**What is a Web Security Threat?**

A web security threat is any potential danger that could harm your website, web applications, or online data.

These threats can lead **to unauthorized access, data breaches, service disruptions, and other malicious activities.**

**1. SQL Injection**

**What is it?**

SQL Injection is a type of attack where malicious SQL code is inserted into a query to manipulate the database and gain unauthorized access.

**How Does It Occur?**

It occurs when user input is not properly sanitized and is directly included in SQL queries.

**How to Overcome It?**

Use prepared statements and parameterized queries to ensure user input is treated as data, not executable code.

**Example**

**Vulnerable Code:**

**PHP**

$username = $\_POST['username'];

$password = $\_POST['password'];

$query = "SELECT \* FROM users WHERE username = '$username' AND password = '$password'";

$result = mysqli\_query($conn, $query);

**Secure Code:**

**PHP**

$stmt = $conn->prepare("SELECT \* FROM users WHERE username = ? AND password = ?");

$stmt->bind\_param("s s", $username, $password);

$stmt->execute();

$result = $stmt->get\_result();

**2. Cross-Site Scripting (XSS)**

**What is it?**

XSS is an attack where malicious scripts are injected into web pages viewed by other users, potentially stealing cookies or session tokens.

**How Does It Occur?**

It occurs when user input is not properly sanitized and is included in the output HTML.

**How to Overcome It?**

Sanitize and encode user input before displaying it in the browser.

**Example**

**Vulnerable Code:**

**PHP**

echo "<div>" . $\_GET['user\_input'] . "</div>";

**Secure Code:**

**PHP**

echo "<div>" . htmlspecialchars($\_GET['user\_input'], ENT\_QUOTES, 'UTF-8') . "</div>";

1. htmlspecialchars($\_GET['user\_input'], ENT\_QUOTES, 'UTF-8'): Converts special characters in the user input to HTML entities.
2. This prevents malicious scripts from being executed if the input contains HTML or JavaScript code.
3. The ENT\_QUOTES flag ensures that both double and single quotes are converted, and 'UTF-8' specifies the character encoding.

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

**What is it?**

CSRF is an attack that tricks the user **into performing actions they did not intend to perform.**

**How Does It Occur?**

It occurs when a malicious website causes a user’s browser to perform an unwanted action on a different site where the user is authenticated.

**How to Overcome It?**

**Use anti-CSRF tokens** to ensure that requests are coming from authenticated users.

**Example**

**Vulnerable Code:**

**HTML**

<form action="/change-email" method="POST">

<input type="email" name="email">

<input type="submit" value="Change Email">

</form>

**Secure Code:**

**HTML**

<form action="/change-email" method="POST">

<input type="hidden" name="csrf\_token" value="<?php echo $\_SESSION['csrf\_token']; ?>">

<input type="email" name="email">

<input type="submit" value="Change Email">

</form>

**4. Distributed Denial of Service (DDoS)**

**What is it?**

DDoS attacks overwhelm your server with traffic, causing it to slow down or crash.

**How Does It Occur?**

It occurs when multiple systems flood the bandwidth or resources of a targeted system.

**How to Overcome It?**

Use rate limiting, web application firewalls (WAF), and DDoS protection services.

**Example**

**Mitigation Strategy:**

* Implement rate limiting to control the number of requests a user can make.
* Use a WAF to filter and monitor HTTP traffic.

A Web Application Firewall (WAF) is a security tool that helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet.

**5. Insecure Deserialization**

**What is it?** Insecure deserialization is a vulnerability that occurs when untrusted data is used to abuse the logic of an application, inflict denial-of-service (DoS) attacks, or execute arbitrary code.

**How Does It Occur?** It occurs when an application deserializes untrusted data without proper validation, allowing attackers to manipulate serialized objects.

**How to Overcome It?** Avoid deserializing untrusted data, use safe libraries, and implement integrity checks.

**Example:** **Vulnerable Code:**

**PHP**

$data = unserialize($\_POST['data']);

**Secure Code:**

**PHP**

// Use JSON instead of PHP serialization

$data = json\_decode($\_POST['data'], true);

**6. Broken Access Control**

**What is it?** Broken access control occurs when users can perform actions they are not authorized to do.

**How Does It Occur?** It occurs due to improper implementation of access control mechanisms, allowing unauthorized access to resources.

**How to Overcome It?** Implement proper access control checks, use role-based access control (RBAC), and regularly test access control mechanisms.

**Example:** **Vulnerable Code:**

**PHP**

// No access control check

if ($\_GET['admin'] == 'true') {

// Perform admin actions

}

**Secure Code:**

**PHP**

// Proper access control check

if ($\_SESSION['role'] == 'admin') {

// Perform admin actions

}

**7. Session Hijacking**

**What is it?** Session hijacking is an attack where an attacker takes over a user’s session by stealing or guessing the session token.

**How Does It Occur?** It occurs when an attacker gains access to a user’s session token, often through XSS attacks, network sniffing, or other vulnerabilities.

**How to Overcome It?** Use secure cookies, implement HTTPS, and regenerate session IDs after login.

**Example:** **Vulnerable Scenario:** An attacker steals a session token through an XSS attack and uses it to impersonate the user.

**Secure Scenario:** Using secure cookies and regenerating session IDs:

**PHP**

session\_start();

session\_regenerate\_id(true); // Regenerate session ID

setcookie(session\_name(), session\_id(), [

'secure' => true,

'httponly' => true,

'samesite' => 'Strict'

]);