

OWASP Top 10 Vulnerabilities

Comprehensive Guide to Web Application Security Risks

Prepared by: [GAURAV ROKADE/VAULT-TEC SECURITY]

Date: [OCT 2024]

Overview

The OWASP Top 10 vulnerabilities list represents the most critical security risks to web applications. This guide is designed to help developers, security professionals, and organizations understand these vulnerabilities and implement effective countermeasures.

([OWASP Top 10 Try Hack Me](#))

Contents

1. Introduction
 2. Detailed Explanations of OWASP Top 10 Vulnerabilities
 3. Mitigation Strategies for Each Vulnerability
 4. Best Practices in Web Security
 5. Conclusion
 6. Acknowledgment
-

Contact Information:

GAURAV MAHADU ROKADE

Intern, Vault-Tec Security, Ahmedabad, Gujarat

[linkedin.com/in/vault-tec-security-052948306](https://www.linkedin.com/in/vault-tec-security-052948306)

Aim: To Learn about and exploit each of the OWASP Top 10 vulnerabilities; the 10 most critical web security risks. And provide mitigation to each vulnerability

- Task information with finding and screenshots of all 10 vulnerabilities.
- Mitigation for OWASP top 10 vulnerabilities.

Task 1: Introduction

Web application security risks based on the OWASP Top 10 list. Here's a brief explanation for each:

1. **Broken Access Control**

This occurs when users can access resources or perform actions that they are not authorized to. Proper access control mechanisms, such as role-based permissions, are missing or improperly implemented.

2. **Cryptographic Failures**

These involve weaknesses in the protection of sensitive data due to poor or insufficient encryption. This could lead to data breaches, exposure of sensitive information, or man-in-the-middle attacks.

3. **Injection**

Injection flaws, such as SQL injection or command injection, happen when untrusted data is sent to a command interpreter or database without proper validation. Attackers exploit these to run malicious commands or access unauthorized data.

4. **Insecure Design**

This refers to flaws in the system's architecture or design that do not consider security. Insecure design often results in applications being vulnerable to future attacks because security was not a priority during development.

5. **Security Misconfiguration**

Security misconfiguration occurs when security settings are improperly implemented or left at their default values. This can expose the application to various attacks due to weak security settings, unnecessary features enabled, or outdated error messages revealing sensitive information.

6. **Vulnerable and Outdated Components**

Using outdated or vulnerable libraries, frameworks, or software components can expose applications to known security risks. These components may contain publicly disclosed vulnerabilities that can be exploited by attackers.

7. **Identification and Authentication Failures**

These occur when the system fails to correctly identify or authenticate users. Weak password policies, flawed session management, or inadequate multi-factor authentication can allow attackers to gain unauthorized access.

8. **Software and Data Integrity Failures**

These flaws occur when software or data, such as updates, configurations, or code, are not verified for integrity. This can allow malicious data or code to be introduced into the system.

9. **Security Logging & Monitoring Failures**


When critical security events are not logged or monitored, suspicious activities may go unnoticed, and security incidents may not be detected in time to prevent further damage.


10. **Server-Side Request Forgery (SSRF)**

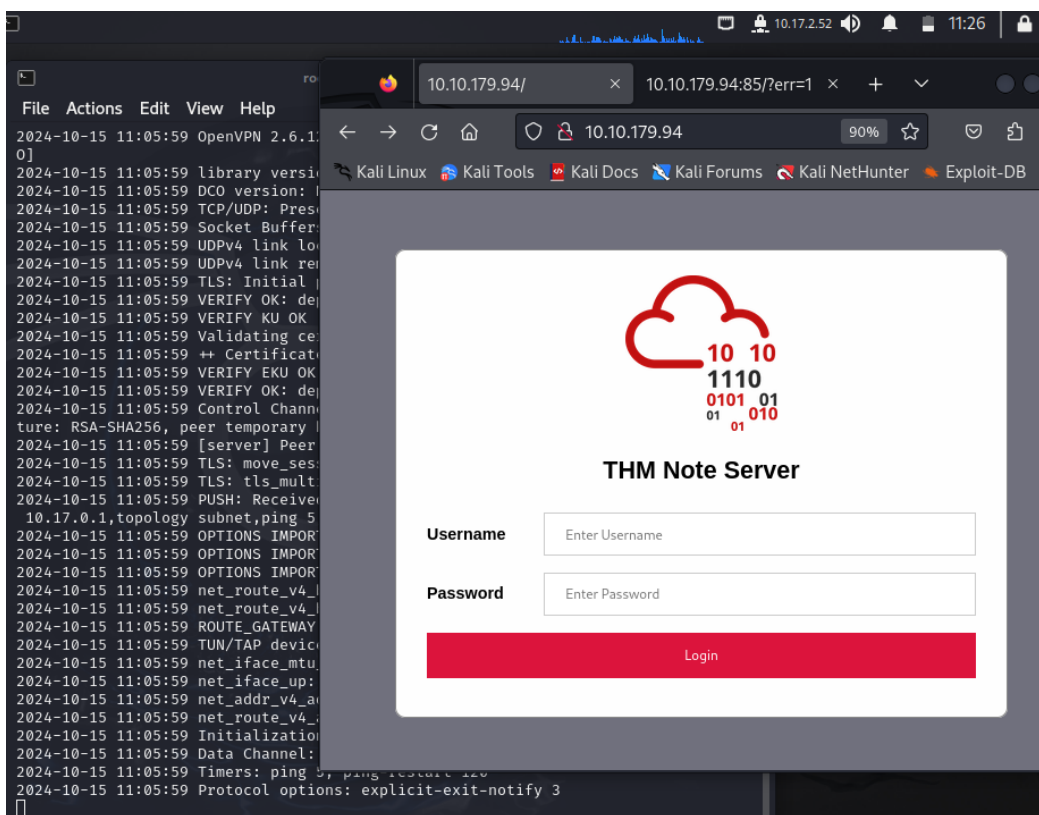
SSRF happens when an attacker tricks the server into making requests to unintended locations, such as internal services or external domains. This can lead to data leaks, unauthorized access to internal resources, or other security breaches.

Task 2: Accessing Machines

Target Machine Information

Title	Target IP Address	Expires
owasp_top10_2021_v1.2	10.10.80.171 	35min 12s





The Machine is run via VPN service inside the kali Linux

Task 3: Broken Access Control (1)

Broken access control happens when a system doesn't properly enforce rules on what users are allowed to access. This means that someone could manipulate the system to access data or functions they shouldn't be able to, like changing a URL to see someone else's information or modifying their permissions. In short, it's when security fails to prevent unauthorized access.

Mitigations:

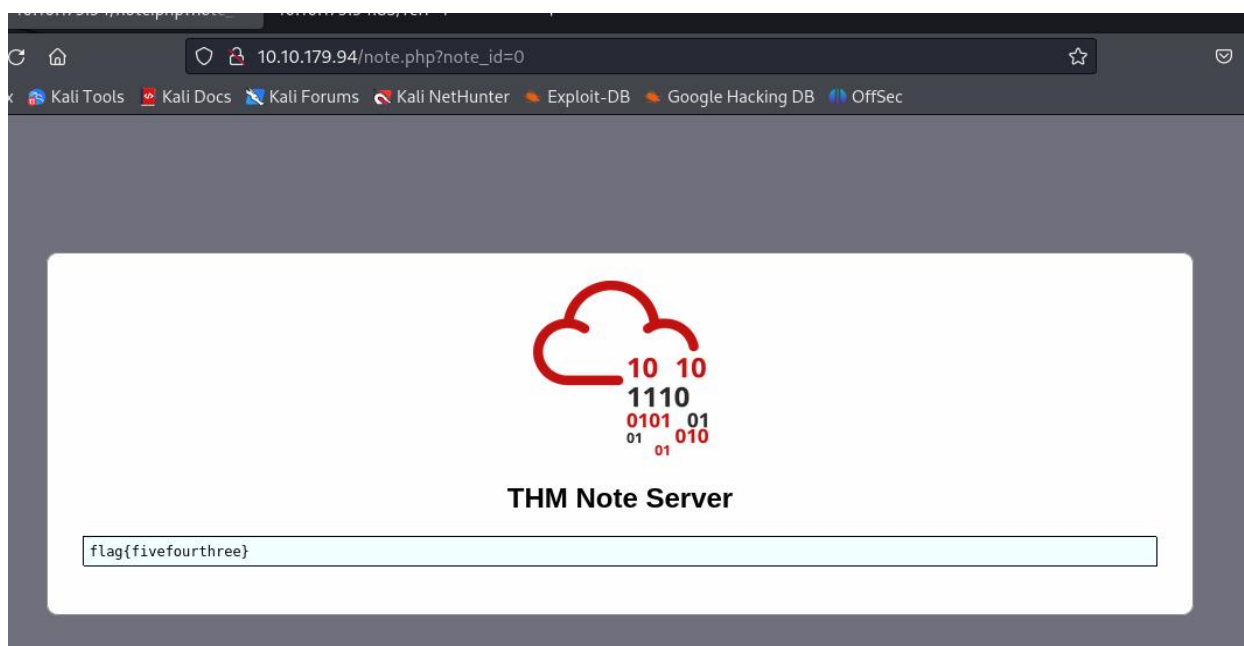
- Role-Based Access Control (RBAC): Define roles with specific permissions and assign users to these roles based on their job functions.
- Attribute-Based Access Control (ABAC): Use attributes (user, resource, environment) to enforce fine-grained access policies.
- Access Control Lists (ACLs): Specify which users or groups have access to certain resources.
- Deny by Default: Ensure that access is denied by default, granting permissions only as necessary.
- Audit and Review: Regularly audit access controls and permissions to ensure they are up-to-date and in line with user roles.

Task 4: Broken Access Control (IDOR Challenge)

Deploy the machine and gone through provide ip - Login with the username **noot** and the password **test1234**.

The URL contains? note Id=1 - change the parameter value, and able to access another user's notes.

flag{fivefourthree}



Task 5: Cryptographic Failures (2)

Cryptographic failures refer to vulnerabilities that arise when cryptographic systems, which are designed to secure data, are improperly implemented or used. These failures can occur due to weak encryption algorithms, poor key management, insecure protocols, or failure to properly secure cryptographic keys. As a result, sensitive data may be exposed, allowing unauthorized access, data breaches, or other forms of cyberattacks.

In short, cryptographic failures happen when encryption doesn't adequately protect information, often due to misconfigurations or weak practices.

Mitigations:

- **Use Industry Standards:** Follow guidelines from organizations like NIST for cryptographic practices.
- **Encryption at Rest and in Transit:** Ensure sensitive data is encrypted both at rest (stored) and in transit (during transmission).
- **Key Rotation:** Regularly rotate cryptographic keys and securely dispose of old keys to minimize the impact of a key compromise.
- **Avoid Custom Cryptography:** Always use established libraries and algorithms; avoid implementing custom solutions.
- **Use Secure Random Number Generators:** For generating cryptographic keys or tokens, use secure random number generators provided by libraries.

Task 6: Cryptographic Failures 1

Flat-file databases, like SQLite, are simpler than server-based databases and store data as files on a computer's disk. While they are easier to set up, especially for smaller applications, they come with potential security risks. If a flat-file database is stored in a web server's root directory, it becomes vulnerable. Users might be able to download and query the database, exposing sensitive data.

In such cases, data exposure can lead to serious security breaches, highlighting the importance of storing sensitive files outside of public web directories. SQLite databases can be queried with the sqlite3 client, which is commonly found on Linux systems.

Task 7: Cryptographic Failures 2

CrackStation exemplifies cryptographic failures by highlighting weaknesses in how passwords are hashed and stored. In secure systems, passwords should be protected through robust cryptographic methods. However, if these methods are poorly implemented, CrackStation can exploit the vulnerabilities to reveal the original passwords.

Here are a few key cryptographic failures that CrackStation can exploit:

1. **Weak Hashing Algorithms:** Older or weak hash functions like MD5 or SHA-1 can be easily cracked because they are fast and lack sufficient security. CrackStation's database contains precomputed hashes for many common passwords using these algorithms, making it easy to reverse them.
2. **Lack of Salt:** If passwords are hashed without a "salt" (a random value added to the password before hashing), attackers can use rainbow tables—precomputed lists of common password hashes—to match hashes and find the original passwords. CrackStation's vast lookup table takes advantage of this cryptographic failure.
3. **Reusing Password Hashes:** When the same password hash is used across multiple users without unique salts, it becomes easier for tools like CrackStation to crack multiple accounts once a single hash is matched.

In essence, CrackStation demonstrates the importance of using strong, modern cryptographic techniques (like salted and iterated hashing with algorithms such as bcrypt or Argon2) to avoid these vulnerabilities and prevent sensitive data exposure

CrackStation

Defuse.ca · Twitter

CrackStation Password Hashing Security Defuse Security

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

5f4dcc3b5aa765d61d8327deb882cf99

I'm not a robot

reCAPTCHA

Crack Hashes

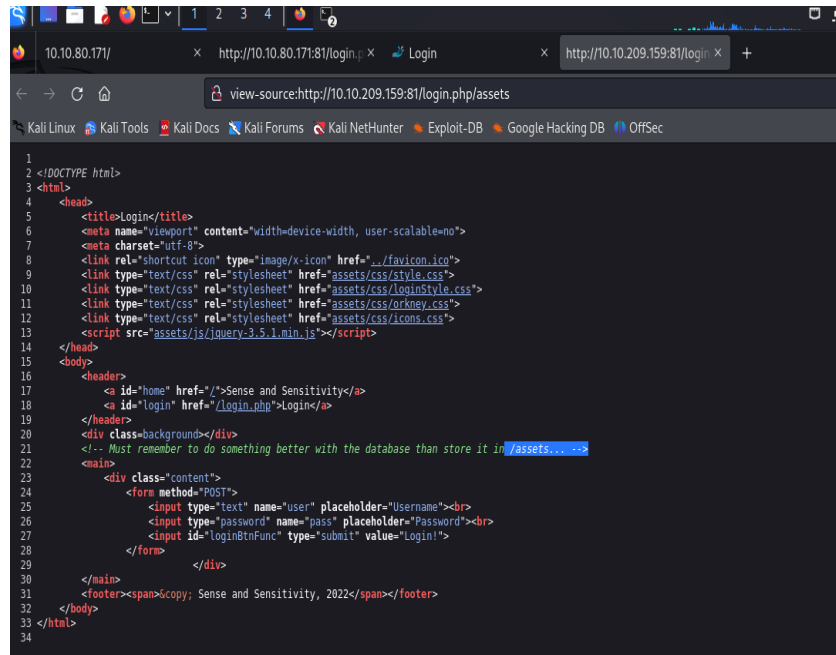
Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: green Exact match, yellow Partial match, red Not found.

Task 8: Cryptographic Failures (Challenge)

I. Name of the mentioned directory



```
1
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <title>Login</title>
6 <meta name="viewport" content="width=device-width, user-scalable=no">
7 <meta charset="utf-8">
8 <link rel="shortcut icon" type="image/x-icon" href="/favicon.ico">
9 <link type="text/css" rel="stylesheet" href="/assets/css/style.css">
10 <link type="text/css" rel="stylesheet" href="/assets/css/loginStyle.css">
11 <link type="text/css" rel="stylesheet" href="/assets/css/orknay.css">
12 <link type="text/css" rel="stylesheet" href="/assets/css/icons.css">
13 <script src="/assets/js/jquery-3.5.1.min.js"></script>
14 </head>
15 <body>
16 <header>
17 <a id="home" href="/">Sense and Sensitivity</a>
18 <a id="login" href="/login.php">Login</a>
19 </header>
20 <div class="background"></div>
21 <!-- Must remember to do something better with the database than store it in /assets/... -->
22 <main>
23 <div class="content">
24 <form method="POST">
25 <input type="text" name="user" placeholder="Username"><br>
26 <input type="password" name="pass" placeholder="Password"><br>
27 <input id="loginBtnFunc" type="submit" value="Login">
28 </form>
29 </div>
30 </main>
31 <footer><span>©copy; Sense and Sensitivity, 2022</span></footer>
32 </body>
33 </html>
34
```

Navigate to the directory you found in question one. What file stands out as being likely to contain sensitive data?

webapp.db

✓ Correct Answer

Use the supporting material to access the sensitive data. What is the password hash of the admin user?

6eea9b7ef19179a06954edd0f6c05ceb

✓ Correct Answer

Crack the hash.

What is the admin's plaintext password?

qwertyuiop

✓ Correct Answer

💡 Hint

Log in as the admin. What is the flag?

THM{Yzc2YjdkMjE5N2VjMzNhOTE3NjdiMjdl}

✓ Correct Answer

Cryptographic Failures, ranked #2 in the OWASP Top 10 (2021), occur when sensitive data isn't properly protected through encryption. Common issues include:

- Use of weak algorithms (e.g., MD5, SHA-1)
- Not encrypting sensitive data
- Poor key management
- Misconfigured TLS/SSL

Task 9: Injection (3)

Injection flaws happen when an application misinterprets user input as commands or queries, leading to attacks like:

- SQL Injection: Attackers manipulate SQL queries to access, modify, or delete database data.
- Command Injection: Attackers execute arbitrary system commands on a server.

Mitigations:

- Input Validation: Enforce strict validation rules on all user inputs to ensure they conform to expected formats.
- Output Encoding: Encode output to prevent execution of injected code, especially in web applications (e.g., HTML encoding).
- Parameterized Queries: Use parameterized queries or prepared statements to separate SQL code from data.
- Stored Procedures: Use stored procedures for database interactions, ensuring they do not execute arbitrary commands.
- Web Application Firewall (WAF): Deploy a WAF to detect and block injection attacks in real time.

Task 10: Command Injection

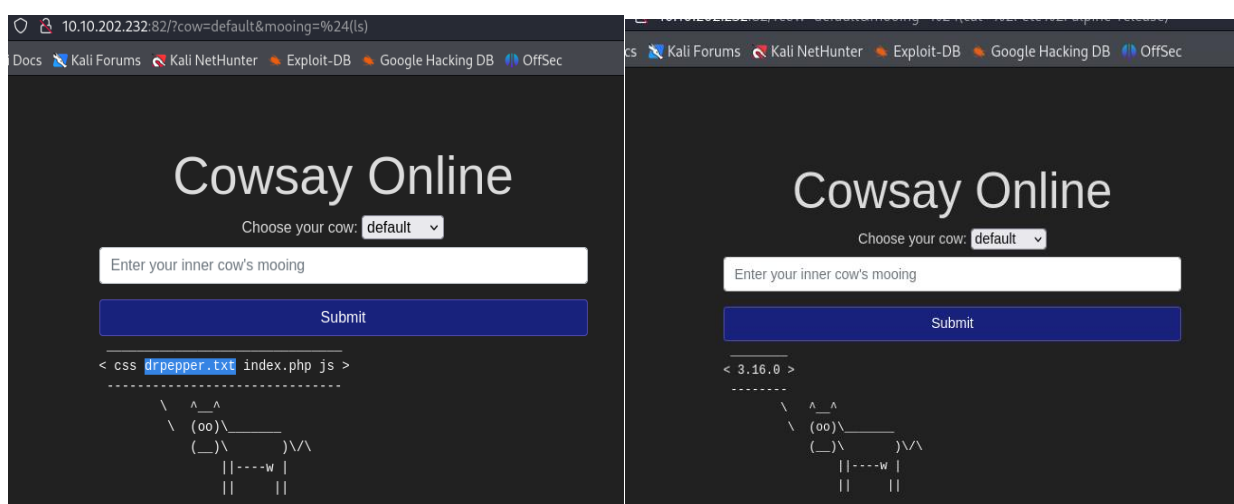
Command injection is a security vulnerability that allows an attacker to execute arbitrary system commands on a server or computer running a vulnerable application. This happens when an application passes user input directly to a system command without proper validation or sanitization.

Example:

If an application takes a user input to search files and passes it to a system command like this:

```
find / -name [user input]
```

An attacker could `enter ; rm -rf /` to delete files on the system.



Task 11: Insecure Design (4)

Insecure design refers to vulnerabilities rooted in the application's architecture, not caused by poor implementation or configuration. These flaws often stem from inadequate threat modeling during the planning phase or from shortcuts developers take for testing. For example, a developer may disable OTP validation for convenience during testing but forget to re-enable it in production.

Example:

Instagram once had a vulnerability in its password reset process, where attackers could bypass rate-limiting by using multiple IP addresses to brute-force a 6-digit reset code. This issue resulted from flawed design assumptions rather than implementation errors.

Insecure design vulnerabilities are harder to fix as they require significant redesign efforts. The best prevention is early-stage threat modelling and secure development practices.

The image shows a TryHackMe challenge window on the left and a Kali Linux virtual machine connection window on the right. The TryHackMe window displays a challenge titled 'Insecure Design' with a practical example about a password reset mechanism. It includes a question: 'What is the value of the flag in joseph's account?' and a hint: 'THM[Not_3ven_c4tz_c0uld_sav3_UI]'. The Kali Linux window shows a file explorer with a 'Flag.txt' file.

TryHackMe | OWASP Top 10 - 2021

the ground up and is security harder to do than any other simple code-related vulnerability. The best approach to avoid such vulnerabilities is to perform threat modelling at the early stages of the development lifecycle. To get more information on how to implement secure development lifecycles, be sure to check out the [SSDLC room](#).

Practical Example

Navigate to <http://10.10.179.94:85> and get into joseph's account. This application also has a design flaw in its password reset mechanism. Can you figure out the weakness in the proposed design and how to abuse it?

Answer the questions below

Try to reset joseph's password. Keep in mind the method used by the site to validate if you are indeed joseph.

No answer needed ✓ Correct Answer

What is the value of the flag in joseph's account?

THM[Not_3ven_c4tz_c0uld_sav3_UI] ✓ Correct Answer Hint

Task 12 5. Security Misconfiguration

Task 13 6. Vulnerable and Outdated Components

Task 14 Vulnerable and Outdated Components - Exploit

Kali on GK - Virtual Machine Connection

File Action Media View Help

10.10.179.94/ 10.10.179.94:85 New Tab

Docs Private Cat Images

Flag.txt

Status: Running

Just go with all common colour name guesses names inside forget password option

solution: 'Green'.

Mitigations:

- Threat Modeling: Conduct threat modeling sessions to identify potential vulnerabilities during the design phase.
- Secure Coding Guidelines: Follow secure coding practices and guidelines throughout the development lifecycle.
- Security Testing: Perform regular security testing, including static and dynamic analysis, during development.
- Security Design Reviews: Conduct design reviews focusing on security concerns before implementation.
- Educate Developers: Provide training and resources on secure design principles to development teams.

Task 12: Security Misconfiguration (5)

Security misconfiguration occurs when security settings are improperly configured, leaving systems vulnerable despite using up-to-date software. Common examples include:

- Poorly set permissions on cloud services (e.g., S3 buckets)
- Unnecessary services or features enabled
- Default accounts with unchanged passwords
- Overly detailed error messages
- Lack of HTTP security headers

These misconfigurations can expose sensitive data or lead to other vulnerabilities like command injection or XXE attacks.

Example:

A security misconfiguration in 2015 led to the Patreon hack, where an exposed debug interface in the Werkzeug Python console allowed attackers to execute arbitrary commands.

Preventing this requires disabling debugging features in production and ensuring proper configuration.

For more details, refer to OWASP's entry on Security Misconfiguration.

The image shows two overlapping screenshots. The left screenshot is from a TryHackMe challenge titled 'OWASP Top 10 - 2021'. It contains a series of steps for a task:

- Step 1: Navigate to `http://10.10.179.94:86/console` to access the Werkzeug console. (Correct Answer)
- Step 2: Use the Werkzeug console to run the following Python code to execute the `ls -l` command on the server:

```
import os; print(os.popen("ls -l").read())
```
- Step 3: What is the database file name (the one with the .db extension) in the current directory?
Answer: `todo.db` (Correct Answer)
- Step 4: Modify the code to read the contents of the `app.py` file, which contains the application's source code. What is the value of the `secret_flag` variable in the source code?
Answer: `THM[Just_a_tiny_misconfiguration]` (Correct Answer)

Below the steps are several task cards for other challenges.

The right screenshot shows a Kali Linux terminal window titled 'Kali on GK - Virtual Machine Connection'. It displays the 'Interactive Console' for the Werkzeug application. The console output shows the execution of the provided Python code, listing the contents of the current directory and the contents of `app.py`, which includes the `secret_flag` value.

Mitigations: Security Misconfiguration

- **Secure Configuration Baselines:** Develop and enforce secure configuration baselines for all systems and applications.
- **Automated Configuration Management:** Use tools to automate configuration management and ensure compliance with security policies.
- **Periodic Audits:** Conduct periodic audits and vulnerability assessments to identify misconfigurations.
- **Environment Segregation:** Segregate production, development, and testing environments to reduce the risk of misconfigurations affecting live systems.
- **Disable Unused Features:** Turn off any unused services, ports, and features to minimize the attack surface.

Task 13: Vulnerable and Outdated Components (6)

Vulnerable and outdated components refer to software or programs that haven't been updated and contain known security flaws. Attackers can easily exploit these vulnerabilities because they are well-documented, and ready-made exploits are often available.

Example:

If a company is using an old version of WordPress (e.g., version 4.6), it is vulnerable to a remote code execution (RCE) attack. Hackers can use existing tools to easily exploit this flaw because the vulnerability is well-known and a fix exists in newer updates.

Keeping software updated is crucial to avoid these types of attacks, as missing a single update can expose the system to serious security risks.

Mitigations:

- **Dependency Management:** Use tools like Dependabot or Snyk to monitor and update dependencies for vulnerabilities.
- **Regular Updates:** Establish a regular schedule for updating software components and libraries.
- **Vendor Security Notices:** Subscribe to security mailing lists for alerts on vulnerabilities in third-party components.
- **Use Software Bill of Materials (SBOM):** Maintain an SBOM to track all components in your software and their versions.
- **Conduct Regular Security Assessments:** Periodically assess all components for vulnerabilities, focusing on those with known issues.

Task 14: Vulnerable and Outdated Components – Exploit

Vulnerable and outdated components are software libraries, frameworks, or applications that have known security flaws or are no longer supported, exposing systems to potential attacks.

Example: Apache Struts Vulnerability

- Background: Apache Struts is a widely-used framework for building web applications.
- Vulnerability: CVE-2017-5638 allows remote code execution due to improper handling of file uploads.

Exploit Scenario:

1. Vulnerable Setup: An application running Apache Struts version 2.3.32 or earlier.
2. Crafting the Exploit: An attacker sends a malicious HTTP request:

```
POST /upload HTTP/1.1
```

```
Host: vulnerable-website.com
```

```
Content-Type: application/x-www-form-urlencoded
```

```
id=%{(#context['com.opensymphony.xwork2.ActionContext'].get('parameters')['id']).toString().toLowerCase()).getClass().getClassLoader().loadClass('java.lang.Runtime').getMethod('exec', new String[]{'cmd.exe'}).invoke(null, new String[]{'calc.exe'})}
```

3. Execution: The crafted request triggers the vulnerability, executing a command (e.g., launching the calculator).

Impact: Successful exploitation can lead to unauthorized access and control over the web server, compromising sensitive data.

Task 15: Vulnerable and Outdated Components – Lab

Find for exploitation id.py file in exploit-dB plat form and run with help of python3

After configuring and yes to the python script locate for given file and get the flag

The screenshot displays a Kali Linux desktop environment. On the left, a web browser window shows a task interface for 'Task 14: Vulnerable and Outdated Components - Exploit'. The interface includes a 'TryHackMe' logo, a 'Woop woop! Your answer is correct' message, and a list of tasks. Task 14 is selected, and the instructions for Task 15 are visible: 'Navigate to <http://10.10.179.94:84> where you'll find a vulnerable application. All the information you need to exploit it can be found online. Answer the questions below. What is the content of the /opt/flag.txt file?'. The answer 'THM{But_its_not_my_fault}' is entered, and a 'Correct Answer' message is displayed. On the right, a terminal window shows the execution of a python3 script. The script attempts to upload a PHP web shell to the target IP address. The output shows the shell being uploaded and verified. The terminal also shows the execution of a command to launch a shell, which results in a 'KeyboardInterrupt' message. The desktop environment includes a taskbar at the bottom with various application icons and a system tray showing the date and time.

Task 16: Identification and Authentication Failures (7)

Authentication and session management are vital for web applications, ensuring that only authorized users can access sensitive data.

1. Authentication Process

- Definition: Verifying user identities, typically via username and password.
- Mechanism: Users provide credentials, which the server verifies. Upon success, a session cookie is issued to maintain user state across requests.

2. Common Flaws

- Brute Force Attacks: Attackers guess usernames and passwords through multiple attempts.
- Weak Credentials: Permitting weak passwords (e.g., "password1") increases vulnerability.
- Weak Session Cookies: Predictable session cookies can be exploited by attackers.

3. Recommendations

- Strong Password Policies: Enforce complex passwords to prevent guessing.
- Account Lockout Mechanisms: Lock accounts after a certain number of failed login attempts.
- Multi-Factor Authentication (MFA): Use additional verification methods for enhanced security.

Mitigations:

- Implement Multi-Factor Authentication (MFA): Require additional verification methods beyond just passwords.
- Secure Password Storage: Use strong hashing algorithms (e.g., bcrypt, Argon2) for storing passwords securely.
- Account Lockout Mechanism: Implement account lockout mechanisms after a predefined number of failed login attempts.
- Session Management Best Practices: Enforce session timeouts and regenerate session tokens after login.
- Use Security Questions Carefully: Avoid using security questions that can be easily guessed or found online.

Task 17: Identification and Authentication Failures Practical

Many times, developers forget to sanitize the input (username and password) provided by users in their applications, making them vulnerable to attacks like SQL injection. However, we will focus on a vulnerability that arises from a developer's oversight but is easy to exploit: the re-registration of an existing user.

Example Scenario

Consider two existing users, Darren and Arthur. If we want to gain access to Darren's account, we can attempt to re-register that username with a slight modification. By entering " darren" (notice the space at the beginning) in the username field, along with the required information like email and password, we can successfully register a new user. This new user will have the same privileges as Darren and will be able to view all the content associated with Darren's account.

Similarly, if we want to access Arthur's account, we can enter " arthur" (again, with a leading space) in the username field during registration. This will create a new user with the same rights as Arthur, allowing us to see all the content associated with Arthur's account.

Register with Darren as your username. You'll receive a message indicating that the user already exists. Now, try registering as " darren" instead. You will find that you are logged in and can see the content present in Darren's account, which, in this case, is the flag you need to retrieve.

You can also try the same steps for Arthur by first attempting to register with Arthur as your username and then using " arthur" to gain access to his account.

The image shows a split-screen view. On the left is the TryHackMe interface for a challenge titled 'OWASP Top 10 - 2021'. The challenge text describes a vulnerability where a user can gain access to another user's account by registering with a username that has a leading space (e.g., " darren"). The challenge includes three questions with their correct answers:

- Question: "What is the flag that you found in darren's account?"
Answer: `fe8079416a21a3c99937fea8874b667` (Correct Answer)
- Question: "Now try to do the same trick and see if you can log in as arthur."
Answer: "No answer needed" (Correct Answer)
- Question: "What is the flag that you found in arthur's account?"
Answer: `d9ac0f7db4fda460ac3edeb75d75e16e` (Correct Answer)

At the bottom of the TryHackMe interface, there are three task cards:

- Task 18: 8. Software and Data Integrity Failures
- Task 19: Software Integrity Failures
- Task 20: Data Integrity Failures

On the right is a Kali Linux virtual machine connection window. It shows a terminal window with the command `curl 10.10.179.94:8088/logged` and the output `d9ac0f7db4fda460ac3edeb75d75e16e`. The terminal window is titled 'Auth hacks' and has a tab for '10.10.179.94:8088/logged'. The terminal window also shows the command `curl 10.10.179.94:8088/logged` and the output `d9ac0f7db4fda460ac3edeb75d75e16e`. The terminal window is titled 'Auth hacks' and has a tab for '10.10.179.94:8088/logged'. The terminal window also shows the command `curl 10.10.179.94:8088/logged` and the output `d9ac0f7db4fda460ac3edeb75d75e16e`.

Task 18: Software and Data Integrity Failures (8)

Integrity refers to ensuring that data remains unmodified and accurate. In cybersecurity, maintaining data integrity is crucial to prevent unwanted changes. For instance, when downloading a software installer, how can you be sure the file hasn't been altered during the download process?

To address this, a hash is often provided alongside the file. A hash is a unique string generated by applying an algorithm to the data, such as MD5, SHA1, or SHA256. After downloading a file, you can recalculate its hash and compare it to the provided hash to verify its integrity.

Example: OpenSSL

For OpenSSL, a widely-used library for secure communications, integrity is critical. When you download the OpenSSL package, it typically includes hash values for verification.

1. Download the OpenSSL installer from the official website.
2. Check the provided hash values (e.g., SHA256) on the website.
3. Calculate the hash on your system:

```
user@linux$ sha256sum openssl-1.1.1k.tar.gz
```

4. *Compare the calculated hash with the hash value listed on the OpenSSL website. If they match, it confirms the file has not been tampered with.*

If the calculated hashes match the published ones, you can confirm the file is intact.

Software and Data Integrity Failures

Integrity failures occur when software or data is used without integrity checks, allowing attackers to modify them, leading to unexpected issues. There are two main types of vulnerabilities:

- Software Integrity Failures: Occur when the software itself is compromised.
- Data Integrity Failures: Happen when the data processed by the application is altered

Mitigations:

- Checksums and Hashes: Use checksums and hashes to verify the integrity of software and data, checking them against known values.
- Code Signing: Sign code and software releases to ensure their integrity and authenticity.
- Implement SRI: Use Subresource Integrity (SRI) for third-party libraries in web applications to ensure they haven't been altered.
- Data Validation and Sanitization: Validate and sanitize all data inputs to ensure they conform to expected formats before processing.
- Regular Integrity Checks: Perform regular integrity checks on critical systems and configurations.

Task 19: Software Integrity Failures

When using third-party libraries, such as jQuery, it's common to include them directly from external servers. For example:

```
<script src="https://code.jquery.com/jquery-3.6.1.min.js"></script>
```

However, if an attacker compromises the jQuery repository, they could inject malicious code into the script. Consequently, users visiting your site would unknowingly execute this malicious code, resulting in a software integrity failure.

To mitigate this risk, modern browsers support Subresource Integrity (SRI), which allows you to include a hash to verify the integrity of the loaded library. If the downloaded file's hash does not match the expected value, the browser will refuse to execute it. Here's how to properly include jQuery using SRI:

```
<script src="https://code.jquery.com/jquery-3.6.1.min.js" integrity="sha256-o88AwQnZB+VDvE9tvIXrMQaPlFFSUTR+nldQm1LuPXQ=" crossorigin="anonymous"></script>
```

The image shows a screenshot of a TryHackMe challenge interface on the left and an SRI Hash Generator tool on the right.

TryHackMe Challenge (Task 19):

- Message: "Woop woop! Your answer is correct"
- Code snippet:

```
<script src="https://code.jquery.com/jquery-3.6.1.min.js" integrity="sha256-o88AwQnZB+VDvE9tvIXrMQaPlFFSUTR+nldQm1LuPXQ=" crossorigin="anonymous"></script>
```
- Text: "You can go to <https://www.srihash.org/> to generate hashes for any library if needed."
- Question: "What is the SHA-256 hash of <https://code.jquery.com/jquery-1.12.4.min.js>?"
- Answer input: `sha256-ZosEbRLbNQzLpnKIkEdrPv7Ioy9C27hHQ+Xp8a4MxAQ=`
- Buttons: "Correct Answer" (green), "Hint" (orange)
- Task list on the left: Task 20 (Data Integrity Failures), Task 21 (9. Security Logging and Monitoring Failures), Task 22 (10. Server-Side Request Forgery (SSRF)), Task 23 (What Next?)

SRI Hash Generator:

- URL: `https://code.jquery.com/jquery-1.12.4.min.js`
- SHA-256 hash: `sha256-ZosEbRLbNQzLpnKIkEdrPv7Ioy9C27hHQ+Xp8a4MxAQ=`
- Generated SRI script:

```
<script src="https://code.jquery.com/jquery-1.12.4.min.js" integrity="sha256-ZosEbRLbNQzLpnKIkEdrPv7Ioy9C27hHQ+Xp8a4MxAQ=" crossorigin="anonymous"></script>
```


Task 20: Data Integrity Failures

Web applications maintain user sessions by assigning session tokens, often stored as cookies in the browser. For instance, a webmail app might store a user's username in a cookie, which can be easily tampered with. If a user changes their cookie to impersonate another user, this leads to a data integrity failure since the application trusts user-modified data.

To prevent this, integrity mechanisms, such as JSON Web Tokens (JWT), can be employed. JWTs contain three parts:

1. Header: Metadata indicating it's a JWT and the signing algorithm (e.g., HS256).
2. Payload: Key-value pairs with data the application wants to store.
3. Signature: A cryptographic hash that ensures the payload's integrity. If the payload is altered, the signature will not match, signaling tampering.

A past vulnerability allowed attackers to bypass JWT signature validation by modifying the header to include `alg: none` and removing the signature. This simple change could let attackers alter the payload, gaining unauthorized access.

Task 21: Security Logging and Monitoring Failures (9)

Importance of Logging in Web Applications

Every user action in web applications should be logged to trace attackers' activities during incidents. Logging helps assess the risk and impact of breaches. Without logs, it's challenging to determine the actions taken by an attacker, leading to significant consequences, including:

- **Regulatory Damage:** If sensitive user information is accessed without a record, application owners may face fines or penalties.
- **Risk of Further Attacks:** An attacker may remain undetected, allowing them to launch additional attacks or steal credentials.

Essential Log Information:

- HTTP status codes
- Time stamps
- Usernames
- API endpoints/page locations
- IP addresses

Logs must be stored securely and backed up in multiple locations. Effective logging is crucial not just post-breach but also for monitoring suspicious activity, such as:

- Multiple unauthorized access attempts
- Requests from unusual IP addresses
- Use of automated tools (indicated by User-Agent headers or request speed)
- Known malicious payloads

The screenshot displays a TryHackMe challenge interface on the left and a terminal window on the right. The challenge, titled 'Task 22: 10. Server-Side Request Forgery (SSRF)', includes a success message and a task description. The terminal window shows a list of log entries with IP addresses, usernames, and timestamps.

TryHackMe Challenge Interface:

Woop woop! Your answer is correct

Just detecting suspicious activity isn't helpful. This suspicious activity needs to be rated according to the impact level. For example, certain actions will have a higher impact than others. These higher-impact actions need to be responded to sooner; thus, they should raise alarms to get the relevant parties' attention.

Put this knowledge to practice by analysing the provided sample log file. You can download it by clicking the **Download Task Files** button at the top of the task.

Answer the questions below

What IP address is the attacker using?

49.99.13.16

✓ Correct Answer ? Hint

What kind of attack is being carried out?

Brute Force

✓ Correct Answer ? Hint

Task 22: 10. Server-Side Request Forgery (SSRF)

Terminal Window:

```
File Edit View
Kali Linux
hosts services networks
File Edit View
200 OK 12.55.22.88 jr22 2019-03-18T09:21:17
200 OK 14.56.23.11 rand99 2019-03-18T10:19:22
200 OK 17.33.10.38 afer11 2019-03-18T11:11:44
200 OK 99.12.44.20 rad4 2019-03-18T11:55:51
200 OK 67.34.22.10 bff1 2019-03-18T13:08:59
200 OK 34.55.11.14 hax0r 2019-03-21T16:08:15
401 Unauthorised 49.99.13.16 admin 2019-03-21T21:08:15
/login
401 Unauthorised 49.99.13.16 administrator 2019-03-21T21:08:20
/login
401 Unauthorised 49.99.13.16 anonymous 2019-03-21T21:08:25
/login
401 Unauthorised 49.99.13.16 root 2019-03-21T21:08:30
/login
```

Mitigations: Security Logging and Monitoring Failures

- Comprehensive Logging: Log all relevant events, including authentication attempts, access to sensitive data, and configuration changes.
- Centralized Log Management: Use centralized log management solutions to aggregate logs for analysis.
- Implement Log Retention Policies: Establish policies for how long logs are retained and ensure they are stored securely.
- Anomaly Detection: Implement anomaly detection systems to identify and alert on unusual patterns in log data.
- Regular Log Reviews: Schedule regular reviews of logs to identify suspicious activity and investigate incidents.

Task 22: Server-Side Request Forgery (SSRF) (10)

Server-Side Request Forgery (SSRF) is a type of vulnerability that occurs when an attacker can manipulate a server into making requests to other internal or external resources. This can allow the attacker to access sensitive information or perform unauthorized actions on behalf of the server.

How SSRF Works

1. Vulnerable Functionality: Typically, SSRF vulnerabilities arise in web applications that accept URLs as user input and fetch data from those URLs. This might involve features like retrieving images, fetching APIs, or even accessing other services.
2. Exploiting the Vulnerability:
 - An attacker can input a crafted URL, tricking the server into making a request that the attacker wouldn't normally be able to make directly.
 - This can lead to accessing internal systems, databases, or services that should not be exposed to external users.

Example of SSRF

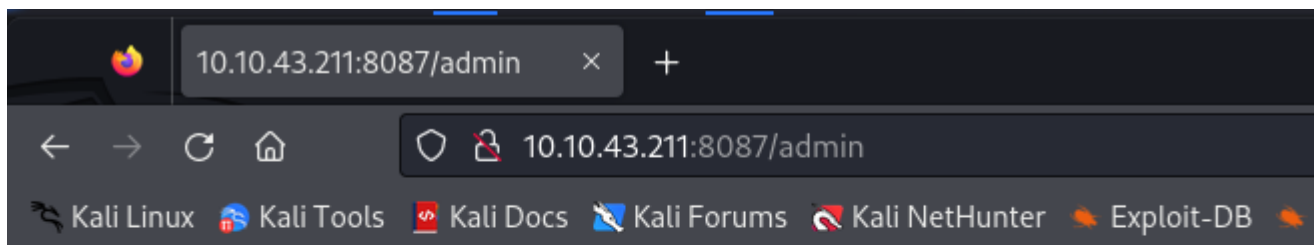
- Scenario: Imagine a web application that allows users to submit a URL to fetch a profile image.
- User Input: A legitimate request might look like this:

```
GET /fetchImage?url=http://example.com/profile.jpg
```

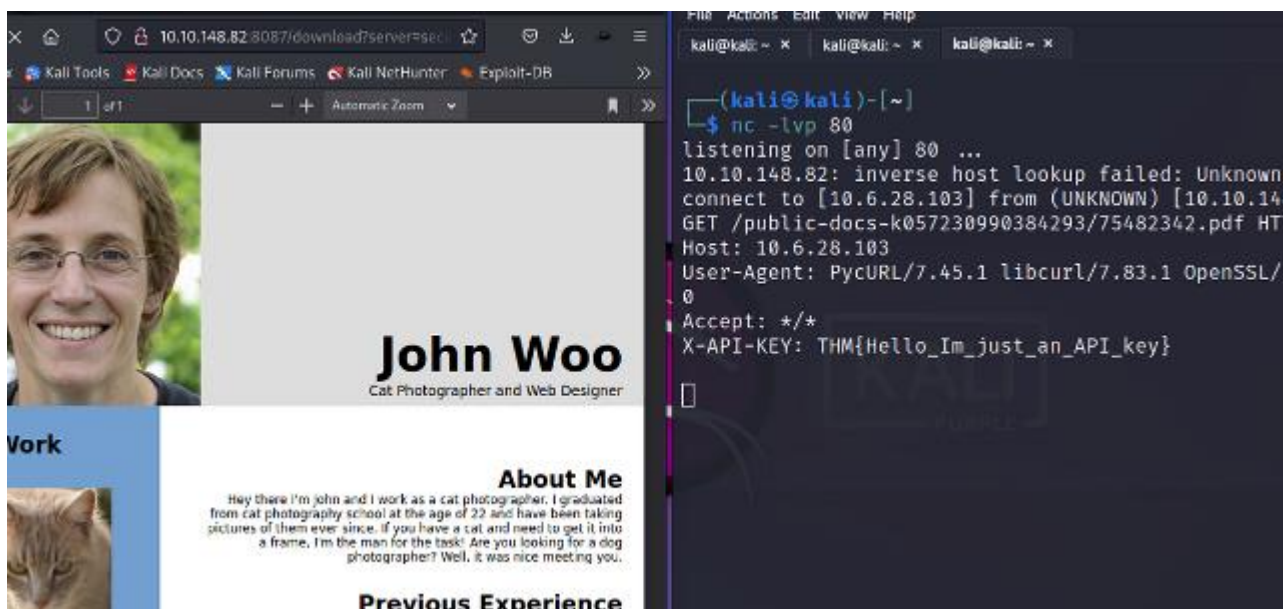
- Attack Vector: An attacker could instead submit:

```
GET /fetchImage?url=http://localhost/admin
```

- Result: The server processes the request and fetches data from its own internal admin interface (e.g., `http://localhost/admin`), potentially exposing sensitive information such as administrative controls, configuration files, or even internal APIs.



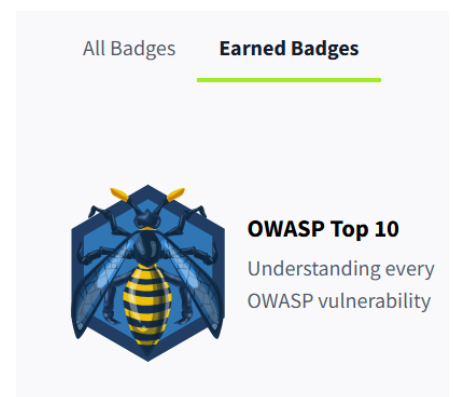
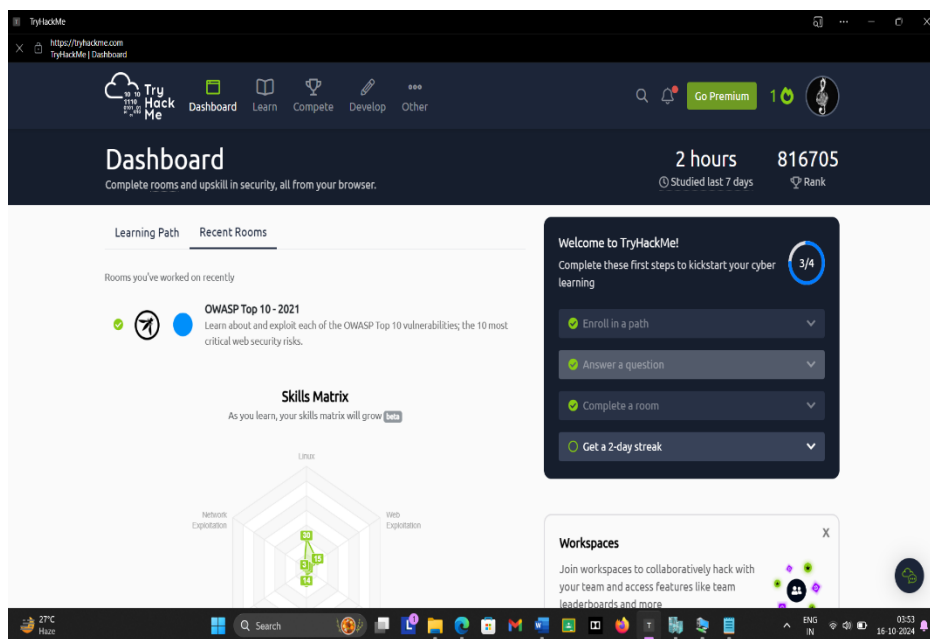
Admin interface only available from localhost!!!



Mitigations:

- Input Validation: Validate and sanitize user input to restrict the URLs and IP addresses users can access.
- Whitelist Trusted Domains: Implement whitelisting for internal requests, allowing only requests to known, safe domains.
- Network Segmentation: Use network segmentation to limit the server's ability to reach sensitive internal resources.
- Disable Unused Protocols: Restrict or disable protocols that are not necessary for the application (e.g., FTP, file access).
- Monitor and Alert: Implement monitoring and alerting for unusual outbound requests that may indicate SSRF attempts.

Task 23: What Next?



Conclusion:

This document provided a comprehensive overview of the OWASP Top 10 vulnerabilities, along with real-world examples, findings, and recommended mitigation techniques. Each vulnerability was examined with a focus on its potential risks and impact on web applications. With this knowledge, developers, system administrators, and security professionals can better understand the security challenges they face in their applications and take appropriate action to minimize these risks.

By following the mitigation strategies for each vulnerability, you can significantly enhance your application's security posture. It's essential to stay vigilant, keep systems and components updated, and implement security best practices at every stage of the software development lifecycle. Regular assessments, security training, and adherence to industry standards will help ensure ongoing protection against these evolving threats.

The screenshots and findings provided in this document serve as a practical demonstration of how these vulnerabilities manifest in real-world scenarios. By addressing these risks early, organizations can build resilient and secure systems that protect both user data and organizational assets.

Next Steps

- Regularly review your application for vulnerabilities.
- Implement continuous monitoring for detecting suspicious activity.
- Stay informed about the latest security threats and updates.

Security is a continuous process, and mitigating these vulnerabilities will go a long way in building a secure and robust system.

Acknowledgment

I would like to express my gratitude to TryHackMe for providing an in-depth, hands-on exploration of the OWASP Top 10 vulnerabilities. This experience has been invaluable in developing a practical understanding of essential cybersecurity threats and how to mitigate them.

Working through the OWASP Top 10 has not only broadened my knowledge of prevalent vulnerabilities but also enhanced my skills in identifying and addressing security flaws. I am thankful for the opportunity to engage in such a comprehensive and challenging learning environment.

Thank you, Vault-Tec Security, for offering me this platform to learn, grow, and demonstrate my skills. I look forward to applying these experiences in future endeavors.