S VYASA DEEMED TO BE UNIVERSITY

School of Advanced Studies

Global Sattva City Campus, Mysore Road, Bengaluru-560059

INTERNSHIP REPORT

On

Web Application Security Testing

Submitted in partial fulfilment of the requirements for the 2nd Semester Degree in

Master of Computer Applications

Under the guidance of

Mr. Mohammed Tousif
Founder and CEO of Cynux Era
Dept. of Computer Applications

Submitted By
John J Joseph [USN:2232408003]

DEPARTMENT OF COMPUTER APPLICATIONS

S-VYASA SCHOOL OF ADVANCED STUDIES, BENGALURU

2024-2025

S- VYASA DEEMED TO BE UNIVERSITY

SCHOOL OF ADVANCED STUDIES

GLOBAL SATTVA CITY CAMPUS, MYSORE ROAD, BENGALURU-560059

CERTIFICATE

This is to certify that the Internship work entitled **"Web Application Security Testing"** carried out by **John J Joseph** (USN 2232408003) are Bonafide students of **SVYASA DEEMED TO BE UNIVERSITY** submitted in partial fulfilment for the award of **Master of Computer Applications** in **Computer Science** of **S-Vyasa Deemed To Be University, Bengaluru,** during the year **2024-2025.** This Internship report has been approved as it satisfies the academic requirements in respect of Internship prescribed for **Master of Computer Applications Degree.**

Signature of Guide	**Signature of Dean**
Mr.Mohammed Tousif	**Dr.Sachin Sharma**
Founder and CEO of Cynux Era	**Dean-CSA**

----------	**----------**
Internal Examiner Signature	**External Examiner Signature**

DECLARATION

I, **John J Joseph (USN 2232408003)** student of 2nd semester MCA- Master of Computer Applications**, Svyasa Deemed to Be University-Bengaluru**, declare that our internship entitled **"Web Application Security Testing"** has been carried out by us and submitted in partial fulfilment of the course requirements for the award of Degree in Master of Computer Applications by Svyasa Deemed To Be University, Bengaluru during academic year 2024- 2025.

John J Joseph (USN 2232408003) _____

ABSTRACT

Web applications are integral to modern digital services but are frequently exposed to a wide range of security vulnerabilities. This project focuses on the practical implementation of web application security testing to identify, analyze, and exploit common vulnerabilities using industry-recognized tools and methodologies. The assessment was conducted in a controlled lab environment using Kali Linux, targeting a deliberately vulnerable application.

The testing process followed the standard five-phase penetration testing model: reconnaissance, scanning, vulnerability assessment, exploitation, and reporting. Tools such as Nmap, Sublist3r, Gobuster, OWASP ZAP, Burp Suite, and Amass were utilized to simulate real-world attack scenarios and uncover security weaknesses in the application. Each phase was carefully documented with screenshots and findings.

The report also includes a literature survey of existing frameworks such as the OWASP Top 10 and PTES, and reviews related academic research to establish a strong theoretical foundation. The results demonstrate the importance of proactive security testing in preventing exploitation and enhancing the resilience of web applications. This project aims to raise awareness of web application security risks and emphasizes the need for secure development practices.

ACKNOWLEDGEMENT

The successful completion of this Internship on **" Web Application Security Testing"**

not have been possible without the valuable support and guidance of several individuals.

We are grateful to our institution**, SVYASA Deemed to Be University-Bengaluru with** its ideas and inspiration for having provided us with the facilities, which has made this Internship work a success.

We would like to express our gratitude to **Dr**. **S. Sridhar, Director-Academics, SVYASA Deemed to Be University-Bengaluru,** who is the source of inspiration as well providing an amiable atmosphere to work in with expert supervision.

We would like to express our gratitude to **Dr. Sachin sharma, Dean-Centre for Scientific Advancement (CSA), Svyasa Deemed to Be University-Bengaluru,** for his valuable guidance, continuous support, and encouragement throughout my internship period. Their insightful feedback and mentorship have significantly enriched my learning experience.

I would like to express my sincere gratitude to my internship guide, **Mr. Mohammed Tousif** Founder and CEO of Cynux Era, for their constant encouragement, insightful feedback, and meticulous guidance throughout the Internship development process. Their expertise and patience were instrumental in shaping this Internship.

I am also thankful to **Department of Computer Applications** for providing the necessary resources and infrastructure to carry out this Internship Work.

Furthermore, I would like to acknowledge all faculties of MCA Department for their assistance and support during various stages of the Internship.

By

John J Joseph USN 2232408003

INDEX

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

In today's digitally connected world, web applications are essential for businesses, institutions, and individuals. They handle sensitive data and enable critical operations ranging from online banking and e-commerce to healthcare and communication. However, this increasing reliance on web applications also introduces significant security risks.

Cybercriminals often target web applications to exploit vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), broken authentication, and insecure APIs. As a result, securing these applications through structured **web application penetration testing** has become a fundamental step in cybersecurity defense.

Penetration testing simulates real-world attacks to evaluate an application's resistance to threats and identify weaknesses before malicious actors can exploit them. This project follows a systematic pentesting methodology to assess the security posture of a target web application in a controlled environment.

1.2 Problem Statement

Despite advancements in security awareness and tools, many web applications remain vulnerable due to insecure coding practices, misconfigured servers, and a lack of continuous security assessments. Organizations often struggle to identify vulnerabilities early in the development lifecycle, leaving them exposed to exploitation.

This project addresses this gap by performing a structured penetration test on a web application using industry-standard methodologies and tools. The goal is to uncover critical vulnerabilities, demonstrate exploitation, and suggest remediation steps, thereby improving the overall security of the application.

1.3 Objectives

The key objectives of this project are:

- To understand and implement the standard phases of web application penetration testing.
- To identify potential vulnerabilities in the selected target web application.
- To exploit discovered vulnerabilities in a safe and controlled manner.
- To document findings and recommend security improvements through a formalreport.
- To gain hands-on experience using penetration testing tools such as Nmap, Burp Suite, OWASP ZAP, Gobuster, and others.

1.4 Scope of the Project

This project focuses solely on **web application penetration testing**. The testing is performed on a vulnerable application deployed in a **lab environment** using **Kali Linux** and various open-source tools.

The scope includes:

- Passive and active reconnaissance
- Vulnerability scanning and assessment
- Exploitation of common web vulnerabilities
- Documentation of findings and recommendations

The scope does *not* include:

- Network-level or physical security testing
- Social engineering attacks
- Denial-of-service (DoS) testing on production servers

1.5 Methodology Overview

The project follows a widely accepted pentesting methodology consisting of five phases:

1. **Reconnaissance** – Gathering initial information about the target.
2. **Scanning** – Identifying open ports, services, and directories.
3. **Vulnerability Assessment** – Detecting and analyzing known weaknesses.
4. **Exploitation** – Attempting to exploit vulnerabilities to gain unauthorized access.
5. **Reporting** – Documenting all findings, evidence, and suggested remediations.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

The purpose of a literature survey is to understand the existing body of knowledge in the field of web application security and penetration testing. It helps in identifying key concepts, methodologies, tools, and common vulnerabilities documented by industry experts and researchers. This section explores the academic and professional literature that guided the design and execution of this penetration testing project.

2.2 Overview of Web Application Vulnerabilities

Web applications are frequently targeted by attackers due to their accessibility and complexity. According to the **OWASP (Open Web Application Security Project)**, the most common and critical vulnerabilities include:

- Injection flaws (e.g., SQL Injection)
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Cross-Site Request Forgery (CSRF)
- Using Components with Known Vulnerabilities

OWASP's **Top 10** list is considered the de facto standard in identifying and mitigating risks associated with web applications. This project aligns its assessment strategy with the OWASP Top 10 framework.

2.3 Penetration Testing Standards and Methodologies

Multiple well-established standards and methodologies guide penetration testers in a structured and ethical manner. Some of the most recognized ones include:

- **OWASP Testing Guide (v4 and v5)**: A detailed manual for testing web application security vulnerabilities using a step-by-step approach.
- **PTES (Penetration Testing Execution Standard)**: Covers everything from pre-engagement interactions to post-exploitation and reporting.
- **NIST SP 800-115**: A technical guide that provides best practices for security testing, including web application penetration testing.

These methodologies were studied and adapted to form the five-phase testing model used in this project.

2.4 Review of Tools Used

A variety of open-source and community-trusted tools are used in the industry and in this project. Key tools include:

Tool	Purpose
Nmap	Network discovery and port scanning
Sublist3r	Subdomain enumeration
Gobuster	Directory brute-forcing
Burp Suite	Web vulnerability scanning, intercepting traffic
OWASP ZAP	Active and passive vulnerability scanner
Nikto	Web server scanning
Amass	Asset discovery and reconnaissance

These tools are essential for identifying open services, insecure endpoints, and application misconfigurations.

2.5 Related Research Work

Research in web application security continues to evolve. Studies and whitepapers often focus on:

- The effectiveness of automated vulnerability scanning
- Exploitation techniques for business logic flaws
- The role of machine learning in identifying anomalies
- Challenges in API security testing

Example:

"Analysis of Automated Web Vulnerability Scanners in Detecting OWASP Top 10 Vulnerabilities" – A study comparing tools like ZAP, Burp Suite, and Wapiti showed a 72% detection rate when using combined active and passive scanning methods.

Such findings validate the selection of tools and approaches used in this project.

Summary

This chapter summarized the theoretical and technical background necessary for conducting a web application penetration test. It reviewed the most common vulnerabilities, frameworks such as OWASP and PTES, and the tools used for effective vulnerability detection. This literature survey not only guided the methodology but also ensured that industry standards were maintained throughout the practical implementation of this project.

CHAPTER 3

METHADOLOGY AND IMPLEMENTAION

3.1 Reconnaissance

Objective

The primary objective of the reconnaissance phase is to collect publicly accessible information about the target web applications—**Damn Vulnerable Web Application (DVWA)**, hosted locally, and **testphp.vulnweb.com**, hosted remotely. This phase is fundamental in identifying the technologies in use, mapping the attack surface, and detecting any potentially vulnerable entry points prior to initiating direct engagement or exploitation.

Reconnaissance is typically divided into two categories:

- **Passive Reconnaissance**: The process of collecting information about a target without directly interacting with it. This approach minimizes the risk of detection while still gathering valuable intelligence from public sources.
- **Active Reconnaissance**: Involves direct interaction with the target system to extract detailed technical information such as service banners, open ports, and response headers. While informative, it can trigger alerts or intrusion detection systems and is often performed in later stages such as scanning.

In this phase, only **passive reconnaissance** was conducted on testphp.vulnweb.com to remain within the constraints of indirect interaction.

Passive Reconnaissance

Target: testphp.vulnweb.com

A deliberately vulnerable web application provided by Acunetix for safe security testing and educational purposes.

The following passive reconnaissance techniques and tools were employed:

a. WHOIS Lookup

- **Command Used**:
  whois testphp.vulnweb.com
- **Purpose**:
  This lookup was performed to obtain domain registration details, including the domain owner, registrar, and DNS configuration. These details help identify administrative ownership and any related hosting infrastructure, which can later aid in social engineering or targeted enumeration.

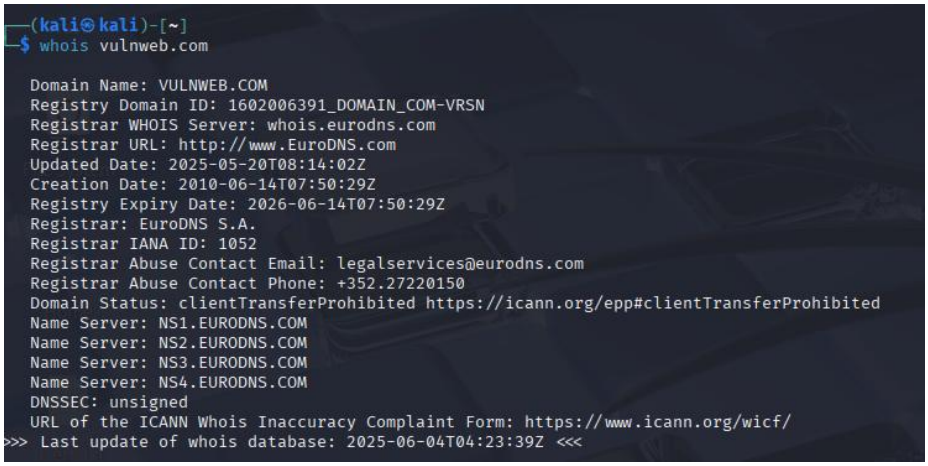


Figure 1.1 whois lookup

b. DNS Enumeration using dig and nslookup

- **Commands Used**:
  dig testphp.vulnweb.com
  nslookup testphp.vulnweb.com
- **Purpose**:
  DNS enumeration reveals critical domain-level information such as A records, CNAMEs, and authoritative name servers. Understanding the DNS configuration assists in uncovering potential subdomains, misconfigurations, and target IPs.

Figure 1.2 : nslookup

c. Shodan Search

- **Platform Used**: https://www.shodan.io
- **Query Used**: testphp.vulnweb.com
- **Purpose**:
  Shodan, a search engine for internet-connected devices, was utilized to determine the open ports, active services, geographic location of the server, and any publicly known vulnerabilities linked to the IP address. This helped assess the external attack surface of the remote host.

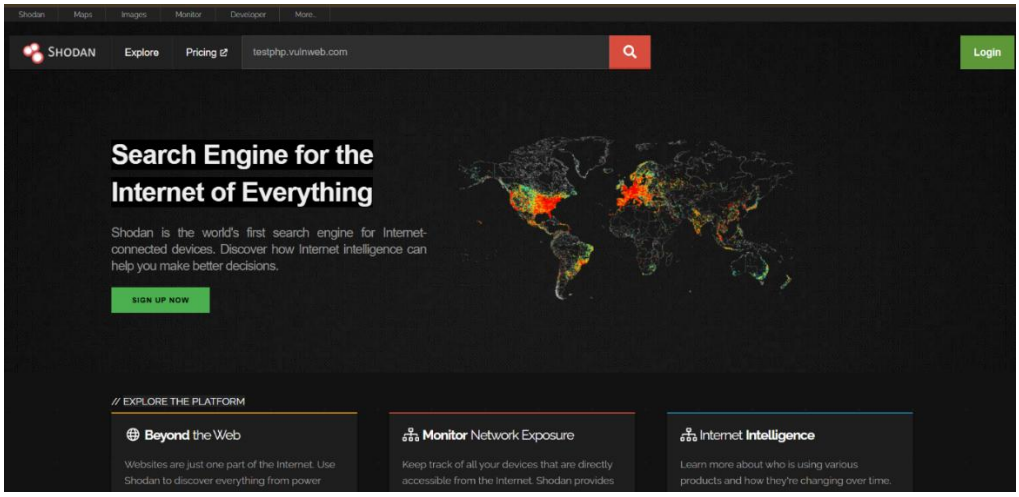


Figure 1.3 shodan website

d. Google Dorking

- **Query Used**:
  site:testphp.vulnweb.com inurl:php?id=
- **Purpose**:

  This technique leverages Google's search indexing to identify potentially vulnerable endpoints, particularly those that include URL parameters susceptible to SQL injection or parameter tampering. As a passive technique, it is valuable for uncovering exploitable patterns without engaging the target directly.

e. Header Analysis using cURL

- **Command Used**:
  curl -I http://testphp.vulnweb.com -H "User-Agent: Mozilla/5.0"
- **Purpose**:
  Header analysis provides insight into the server software, content types, cache policies, and security headers such as X-Frame-Options, X-XSS-Protection, and Content-Security-Policy. Identifying missing or misconfigured headers can inform later exploitation attempts (e.g., for CSRF or clickjacking).

Figure 1.4 curl

f. Technology Fingerprinting using Wappalyzer

- Tool Used: Wappalyzer (Browser Extension)
- Purpose:
  Wappalyzer was used to detect the web technologies and frameworks supporting the target site, such as PHP, Apache, and jQuery. Recognizing the tech stack allows a penetration tester to identify known vulnerabilities specific to those technologies and plan targeted attacks accordingly.

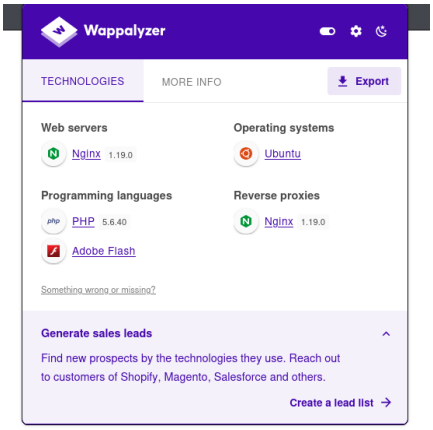


Figure 1.5 wappalyzer

Summary and Transition

The reconnaissance phase provided a comprehensive understanding of the target's exposed infrastructure and web stack without alerting any intrusion detection mechanisms. The insights gained through passive reconnaissance directly informed the selection of tools and techniques used in subsequent phases such as scanning and exploitation.

In the **Exploitation** phase, vulnerabilities inferred during reconnaissance were actively tested. These included:

- **Brute-force attacks and SQL Injection** on testphp.vulnweb.com/test.php
- **Cross-Site Scripting (XSS)** and **Cross-Site Request Forgery (CSRF)** on local platforms such as DVWA and bWAPP

3.2 Scanning & Enumeration

The scanning and enumeration phase involves **actively probing the target system** to identify open ports, services, software versions, directories, and technologies in use. While such actions may trigger IDS/IPS alerts on production systems, they are safe to perform on **intentionally vulnerable environments** such as testphp.vulnweb.com and the locally hosted **DVWA** (Damn Vulnerable Web Application).

A. Nikto Scan

Nikto is a command-line web server scanner used to detect potentially dangerous files, outdated server software, and common configuration issues.

- **Remote Target**

This scans the publicly hosted vulnerable application for exposed directories and misconfigurations.

nikto -h http://testphp.vulnweb.com

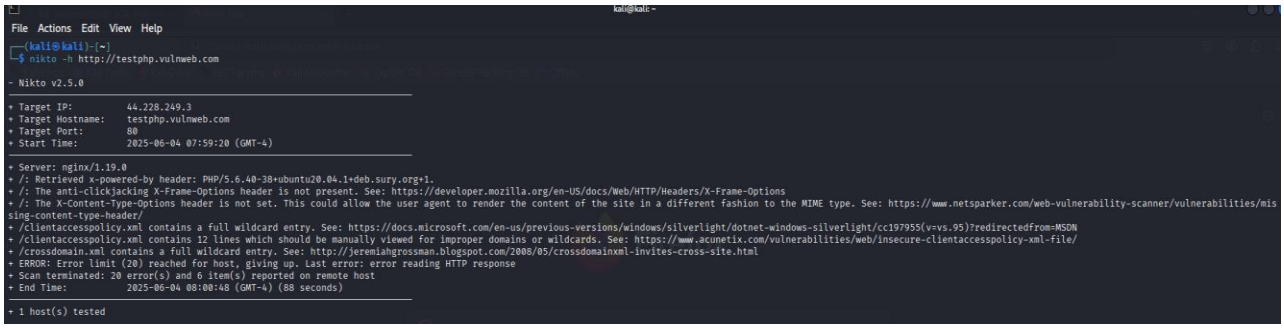


Figure 2.1 Nikto scan website

- **Local DVWA Instance:**

nikto -h http://localhost/dvwa

This performs a similar scan on the local DVWA web server to identify potential weaknesses such as missing headers or open configuration files.

Figure 2.2 Nikto scan hosted

B. Nmap Scan

Nmap (Network Mapper) is a powerful tool for network discovery and security auditing. It can identify open ports, services, and software versions.

- Basic Service and Version Detection (Remote Target)**:**

nmap -sV -sS testphp.vulnweb.com

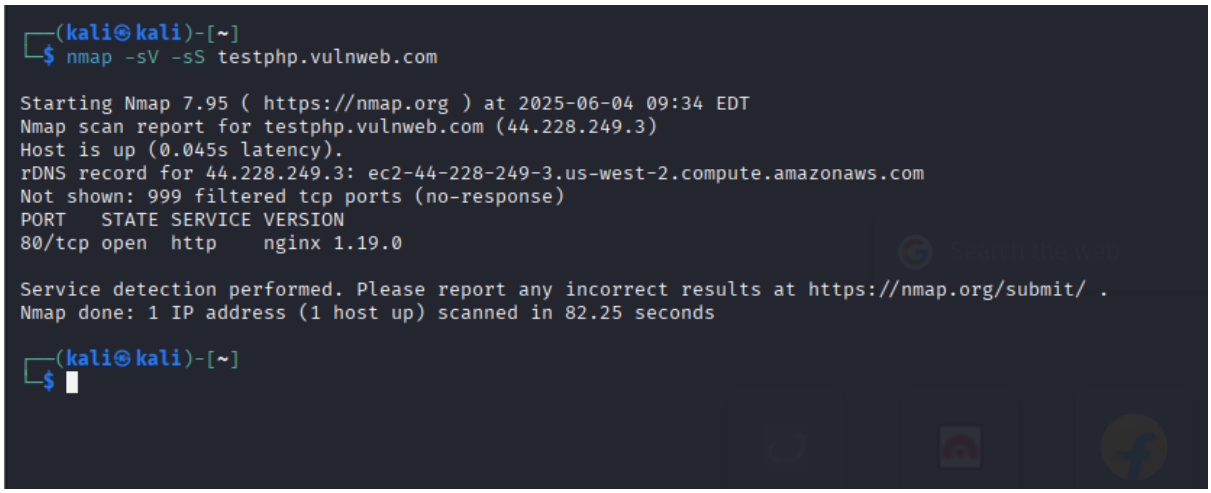


Figure 2.3 nmap scan website

- **Scan Localhost (DVWA):**

nmap -sV -sS localhost

These scans reveal which ports are open, what services are running on them, and the version of each service—valuable information fo

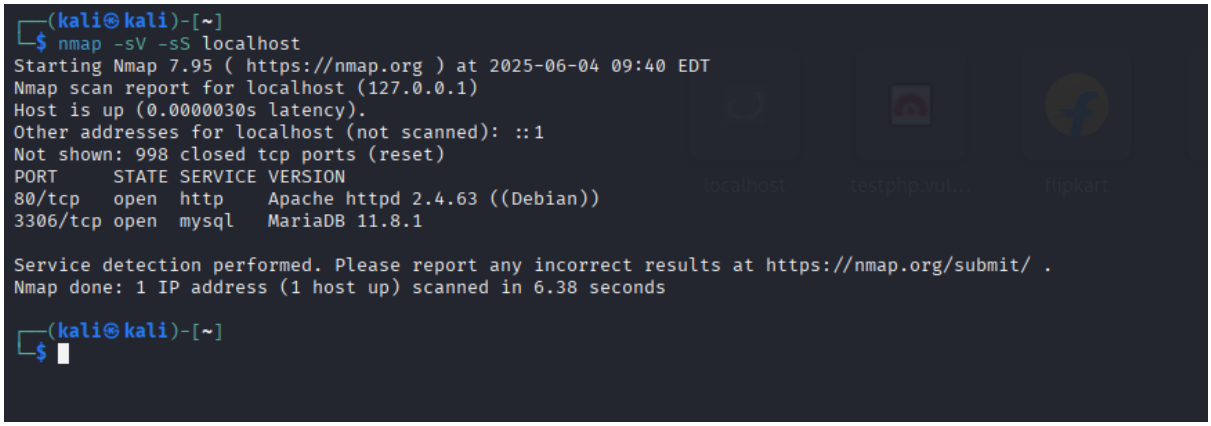


Figure 2.4 nmap scan loachost

- **Enumerate Web Services with Nmap Script:**

nmap --script=http-enum -p 80 testphp.vulnweb.com

This command uses Nmap's scripting engine to identify common directories, admin panels, or configuration files hosted on port 80 (HTTP), enhancing visibility into the web application's structure.

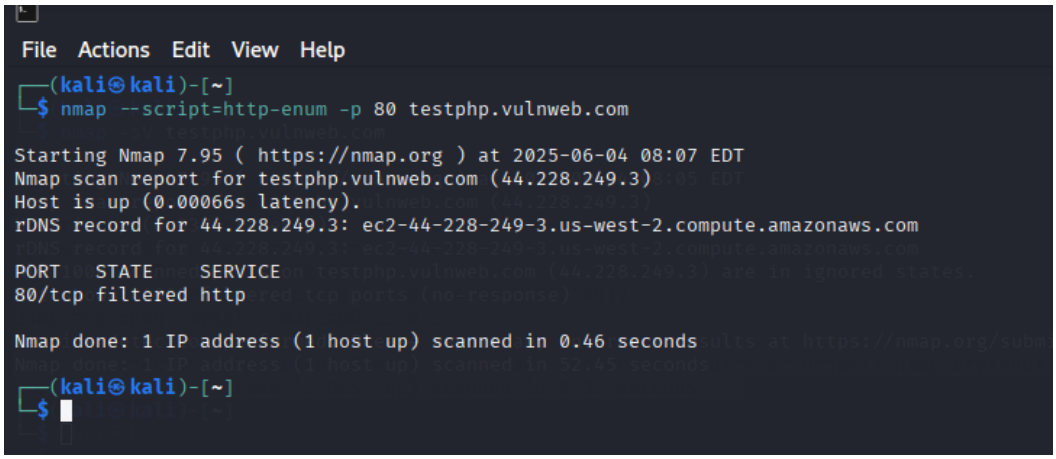


Figure 2.5 nmap enumerate

C. WhatWeb

WhatWeb is a website fingerprinting tool that identifies technologies used on a web server, including frameworks, programming languages, plugins, and web servers.

- **Command:**

whatweb http://testphp.vulnweb.com

Purpose:
This provides a summary of technologies such as Apache, PHP, JavaScript libraries, and content management systems, helping to tailor further testing based on the tech stack.

Figure 2.6 whatweb

D. Directory Brute Forcing (Gobuster/Dirb)

Gobuster is a directory/file brute-forcing tool used to discover hidden or unlinked directories on a web server by using a wordlist.

- **Command:**

gobuster dir -u http://testphp.vulnweb.com -w /usr/share/wordlists/dirb/common.txt

Explanation:
This command scans the target URL for common directories and files using a predefined wordlist (common.txt). It attempts to access each listed path and reports which ones return a valid response.

On the **DVWA site**, this technique helps uncover internal directories such as /dvwa/config/, /dvwa/database/, or /dvwa/uploads/ that may not be linked on the main website but are accessible and potentially vulnerable. Mapping these directories is crucial for further analysis and exploitation.

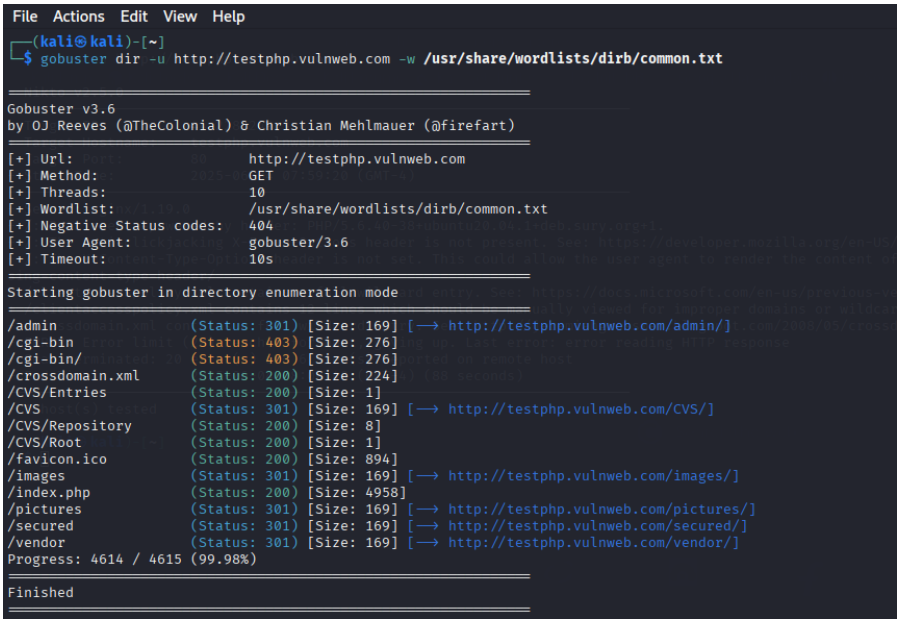


Figure 2.7 gobuster website

Now on dvwa site we can see the directories that are present in the sites. We map them using the gobuster tool that is available to us in the kali OS.

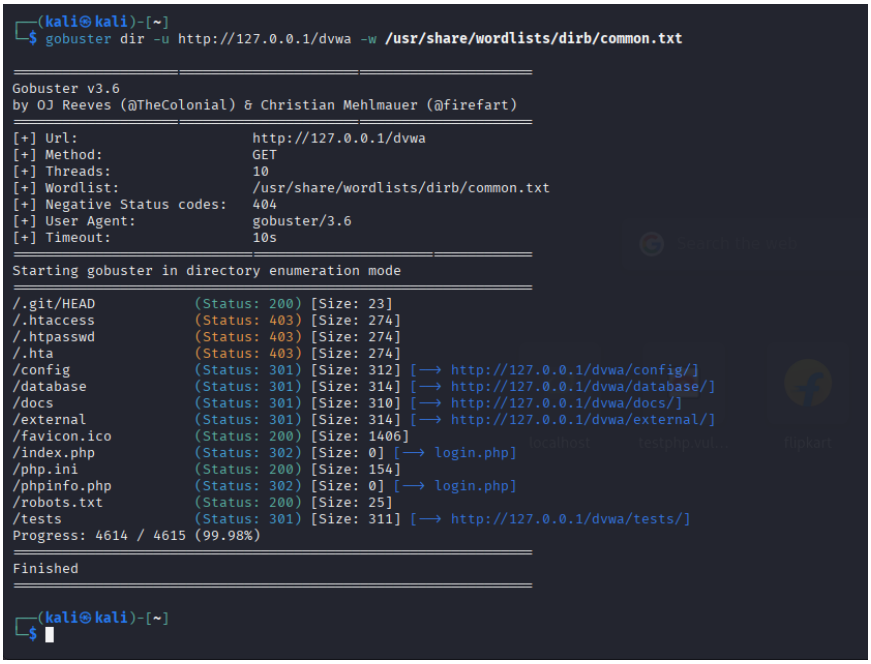


Figure 2.7 gobuster localhost

3.3 Vulnerability Identification

Following the reconnaissance and scanning phases, we analyzed the gathered data to identify potential vulnerabilities in the target environments—**DVWA** (hosted locally) and **testphp.vulnweb.com**. These vulnerabilities were validated and further confirmed through targeted exploitation in the next phase. The findings were mapped against the **OWASP Top 10 vulnerabilities** to understand their potential impact and relevance to real-world scenarios.

1. Missing Security Headers

- **Tool Used:** Nikto
- **Target:** http://localhost/dvwa
- **Findings:**
  - Missing X-Frame-Options → indicates susceptibility to **Clickjacking** attacks.
  - Missing X-Content-Type-Options → allows **MIME-sniffing**, which may lead to content-type confusion attacks.
- **Risk Level:** Medium
- **OWASP Category:** A5 – Security Misconfiguration

2. Exposed Git & Configuration Files

- **Tool Used:** Nikto

- **Target:** http://localhost/dvwa
- **Findings:**
    - Accessible files such as .git/config, .gitignore, .dockerignore were discovered.
    - Open directory indexing on /dvwa/config/ and /dvwa/database/ exposed internal structure.
- **Risk Level:** High
- **Impact:** May reveal credentials, internal application logic, and sensitive configurations.
- **OWASP Category:** A5 – Security Misconfiguration

3. SQL Injection

- **Method:** Manual exploitation on testphp.vulnweb.com
- **Findings:**
    - Login form and input fields were vulnerable to **SQL injection** using payloads such as:

      ' OR '1'='1

    - Error-based responses confirmed backend SQL query manipulation.
- **Risk Level:** Critical
- **Impact:** Unauthorized access to database content, bypassing authentication mechanisms.
- **OWASP Category:** A1 – Injection

4. Cross-Site Scripting (XSS)

- **Tools Used:** DVWA (Low and Medium Security Levels), Manual Testing
- **Target:** http://localhost/dvwa
- **Findings:**
    - Reflected XSS identified in form inputs such as message/comment fields.
    - Payloads like <script>alert(1)</script> were successfully executed.
- **Risk Level:** High
- **Impact:** Could lead to session hijacking, credential theft, or defacement.
- **OWASP Category:** A7 – Cross-Site Scripting

5. Cross-Site Request Forgery (CSRF)

- **Tool Used:** DVWA CSRF Module
- **Target:** http://localhost/dvwa
- **Findings:**
    - Lack of CSRF token validation allowed unauthorized **state-changing requests**.
    - Crafted POST requests could change admin credentials without the user's knowledge.
- **Risk Level:** High
- **OWASP Category:** A8 – Cross-Site Request Forgery

6. Brute-Force Login (Weak Authentication Controls)

- **Tool Used:** DVWA Brute Force Module
- **Target:** http://localhost/dvwa
- **Findings:**
    - No rate limiting or account lockout implemented.
    - Repeated login attempts using a dictionary list successfully revealed valid credentials.
- **Risk Level:** High
- **OWASP Category:** A2 – Broken Authentication

7. Open Directories & File Disclosure

- **Tools Used:** Gobuster, Nikto
- **Targets:** http://localhost/dvwa and http://testphp.vulnweb.com
- **Findings:**
    - Openly accessible directories such as /dvwa/docs/, /dvwa/tests/, and /dvwa/database/ were discovered.
    - Lack of access controls and proper permissions led to disclosure of internal files.
- **Risk Level:** Medium
- **OWASP Category:** A5 – Security Misconfiguration

8. Potential Outdated Software Versions

- **Tools Used:** Nikto, Nmap, WhatWeb
- **Findings:**
    - Detected software: Apache/2.4.63 (on DVWA instance).
    - This version can be cross-referenced with public CVEs to identify any known vulnerabilities.
- **Risk Level:** Medium
- **Recommendation:** Perform CVE analysis using tools like **Nessus** or **SearchSploit**.
- **OWASP Category:** A9 – Using Components with Known Vulnerabilities

Summary

The vulnerabilities listed above were identified using a combination of **automated scanning tools** (Nikto, Nmap, Gobuster, WhatWeb), **browser reconnaissance tools** (such as Wappalyzer), and **manual testing** on DVWA and testphp.vulnweb.com. Each vulnerability was verified through exploitation to confirm its presence and assess its severity. These findings form the foundation for the exploitation phase that follows.

3.4 Exploitation

Brute Force Attack

Objective:
The objective of this penetration test was to assess the strength of the authentication mechanism on the target web application at http://testphp.vulnweb.com/login.php. Specifically, we aimed to determine if the login page is vulnerable to brute force attacks by systematically trying multiple passwords against a known username to gain unauthorized access.

Target URL:
http://testphp.vulnweb.com/login.php

Step 1: Navigating to the Target Website
We accessed the login page and identified input fields for username and password. For this test, we assumed the valid username was already known, focusing on brute forcing the password field.

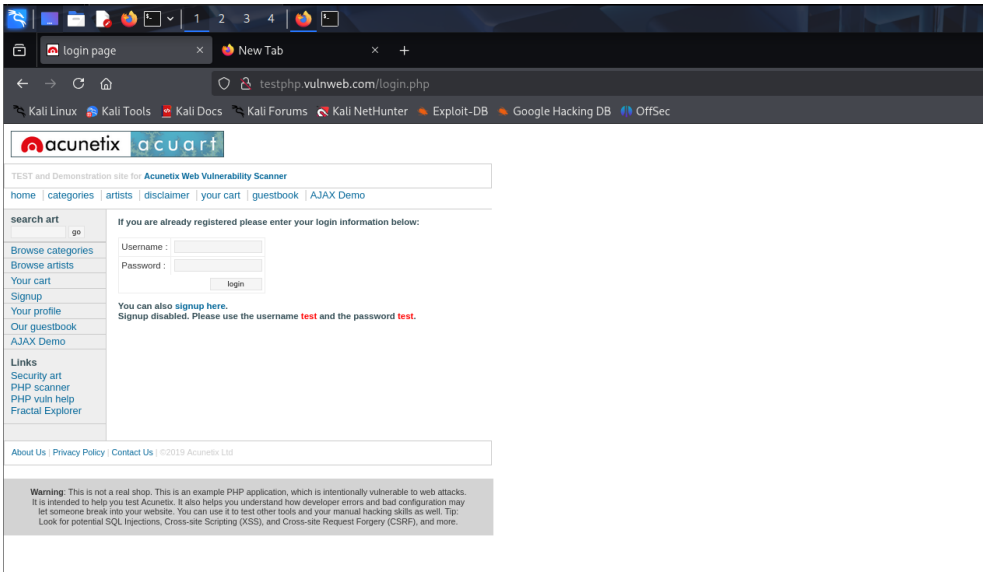


Figure 3.1 Target website

Step 2: Setting Up Burp Suite and Interception

Burp Suite was launched with a new temporary project. Browser traffic was routed through Burp Suite using FoxyProxy, and the Intercept feature in the Proxy tab was enabled to capture HTTP requests before they were sent to the server.

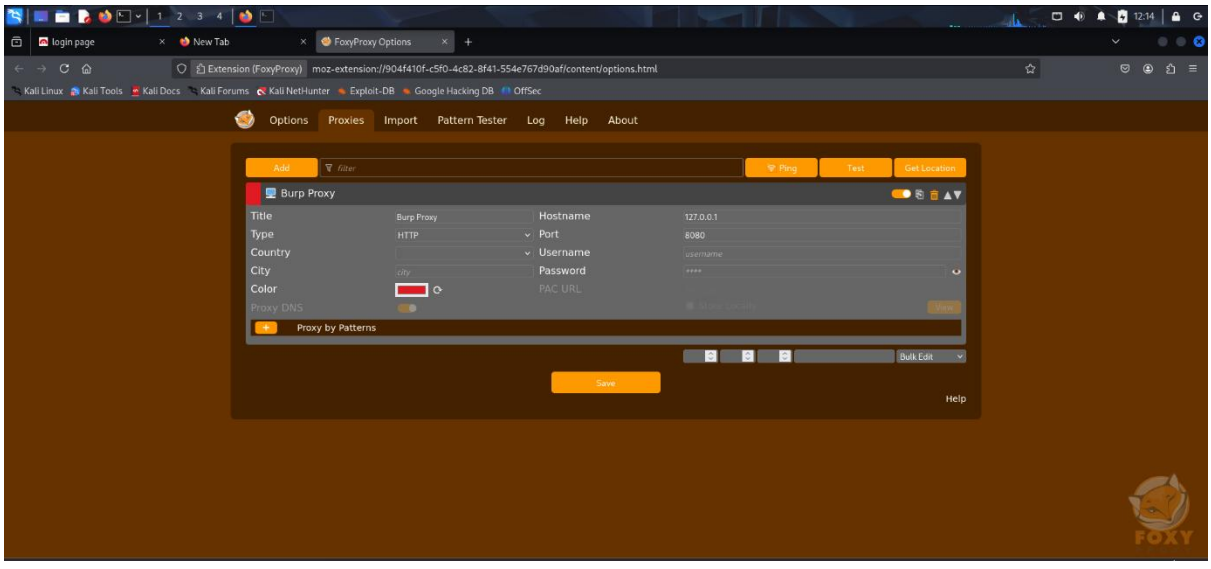


Figure 3.2 Burp suite proxy

Step 3: Capturing the POST Request

A login attempt was made using the known username and an incorrect password. The HTTP POST request containing these credentials was intercepted by Burp Suite, revealing the target URL, headers, and form data necessary for the attack..

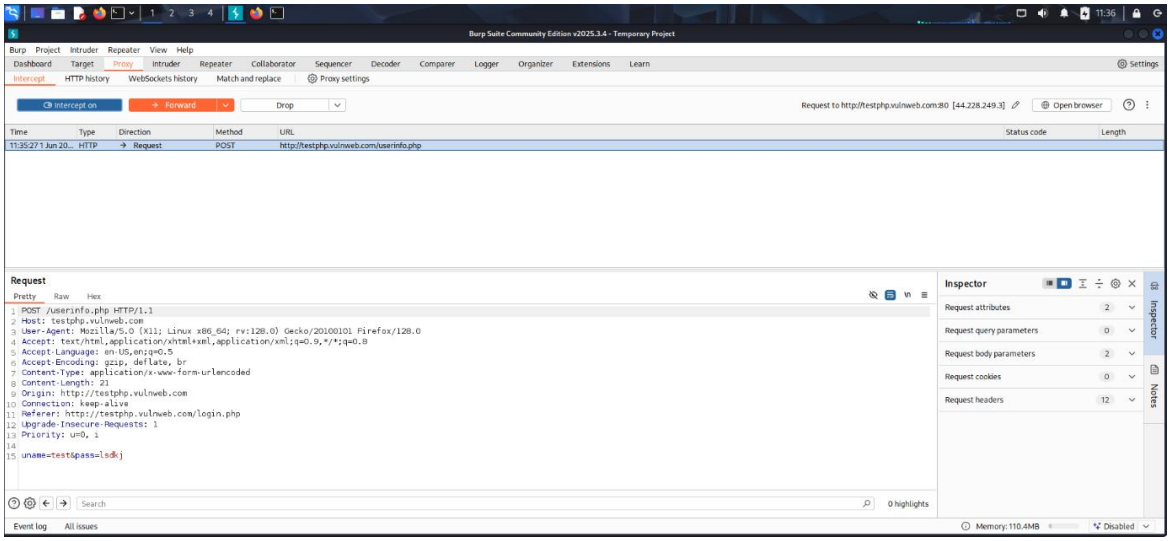


Figure 3.3 capture request

Step 4: Sending the Request to Intruder

The captured request was sent to the Intruder tab. We cleared default payload positions and highlighted only the password parameter to target it exclusively during the attack.

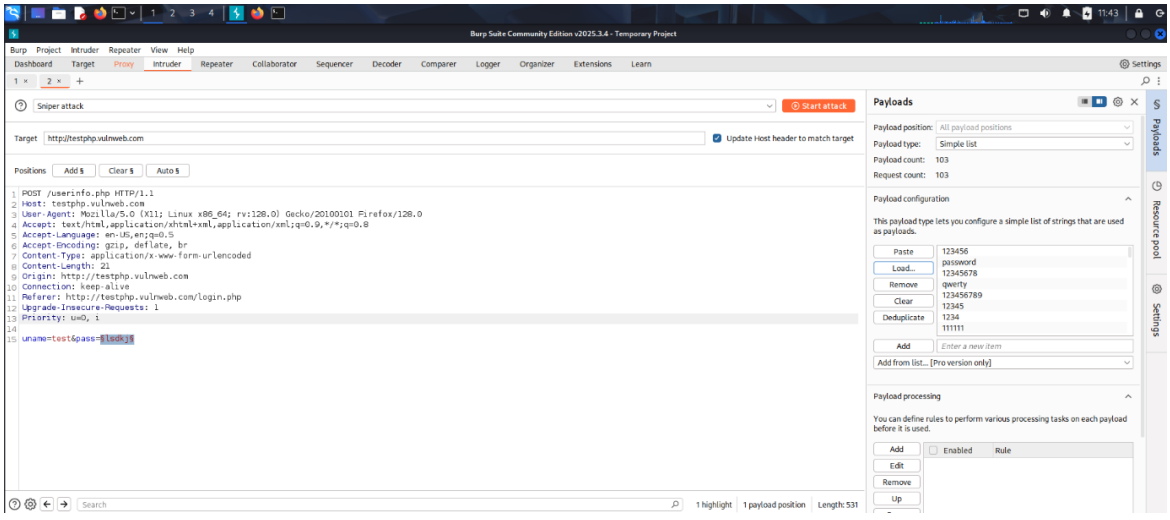


Figure 3.4 request in intruder

Step 5: Configuring the Brute Force Attack

We selected the Sniper attack type, ideal for testing a single parameter. A wordlist of commonly used passwords was uploaded as payloads, allowing Burp Suite to automate login attempts with each password.

Step 6: Launching the Attack

The attack was started and monitored. Responses were analyzed for differences in status codes and content length. A password "test" was identified as successful based on distinct response characteristics.

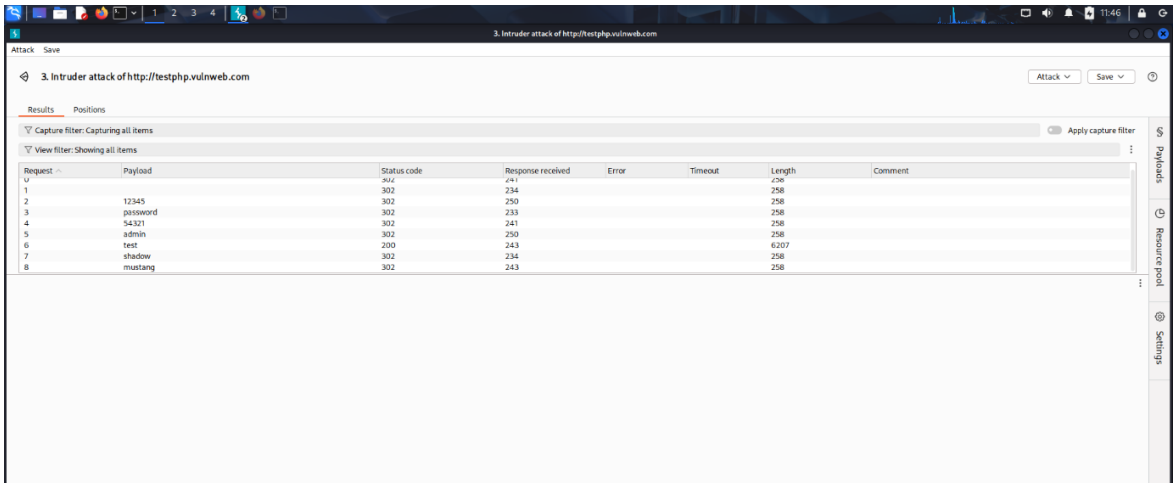
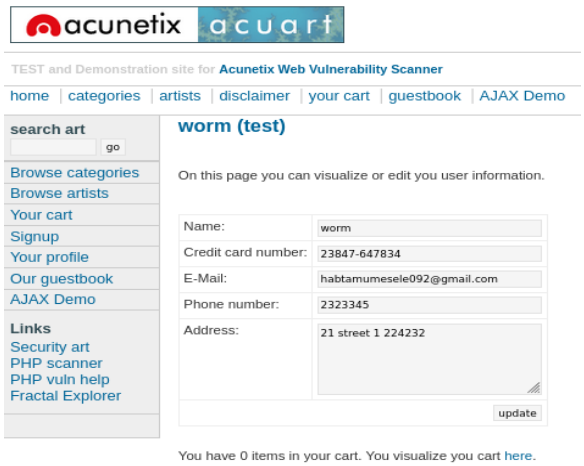


Figure 3.5 Launching Attack

Step 7: Verifying the Credentials

Manual login using the identified password confirmed successful authentication. The intercepted successful request matched the Intruder's findings, verifying the brute force success.



Conclusion:

The login page is vulnerable to brute force attacks due to the absence of protective controls such as rate limiting, account lockouts, or CAPTCHAs. Additionally, consistent server responses made it easier to identify successful logins, showing a lack of response obfuscation. This highlights the need for improved authentication security measures and secure coding practices.

SQL Injection

Objective:

To determine whether the web application is vulnerable to **SQL Injection (SQLi)** by injecting malicious SQL commands into the application's input fields or URL parameters and attempting to access unauthorized data from the backend database.

Testing Approach:

We began our test by targeting a GET request parameter in the URL of the vulnerable web application:

http://testphp.vulnweb.com/listproducts.php?cat=1

To check for SQL Injection, we altered the cat parameter value to an invalid character like an asterisk (*):

http://testphp.vulnweb.com/listproducts.php?cat=*

This resulted in a **SQL syntax error** being displayed by the server, indicating that the application **directly inserts user input into an SQL query without proper sanitization or filtering**. This is a clear sign of a SQL Injection vulnerability.

Using SQLMAP for Automated Exploitation:

To exploit this vulnerability and extract further information from the database, we used **SQLMAP**, an automated SQL injection and database takeover tool. Here's how the process went:

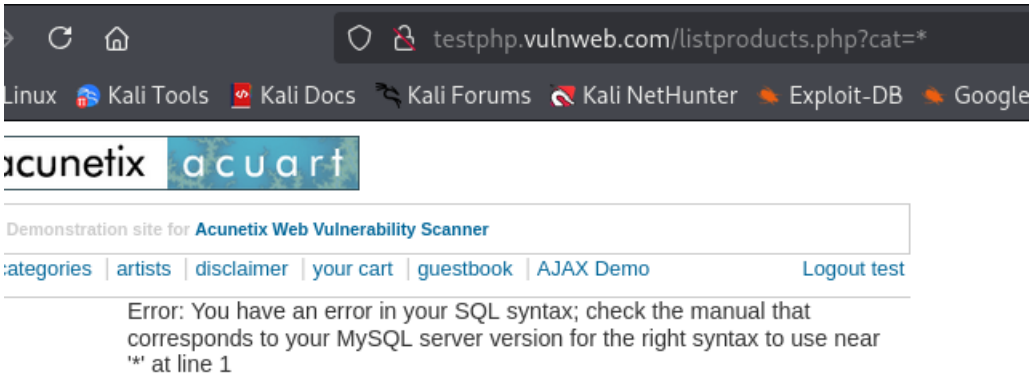


Figure 4.1 check sql injection

Sqlmap –help shows the options that we can use to enumerate the backend of the database

```
Enumeration:
  These options can be used to enumerate the back-end database
  management system information, structure and data contained in the
  tables

  -a, --all           Retrieve everything
  -b, --banner        Retrieve DBMS banner
  --current-user      Retrieve DBMS current user
  --current-db        Retrieve DBMS current database
  --passwords         Enumerate DBMS users password hashes
  --dbs               Enumerate DBMS databases
  --tables            Enumerate DBMS database tables
  --columns           Enumerate DBMS database table columns
  --schema            Enumerate DBMS schema
  --dump              Dump DBMS database table entries
  --dump-all          Dump all DBMS databases tables entries
  -D DB               DBMS database to enumerate
  -T TBL              DBMS database table(s) to enumerate
  -C COL              DBMS database table column(s) to enumerate

Operating system access:
  These options can be used to access the back-end database management
  system underlying operating system

  --os-shell          Prompt for an interactive operating system shell
  --os-pwn            Prompt for an OOB shell, Meterpreter or VNC

General:
```

Figure 4.2 help in sqlmap

Step 1: Database Enumeration

Command used:

sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --dbs

This command tells SQLMAP to scan the target URL for injection points and attempt to enumerate the available databases.

Result:

SQLMAP identified and listed two databases:

- accurate – likely the main application database
- information_schema – a default system database containing metadata about other databases

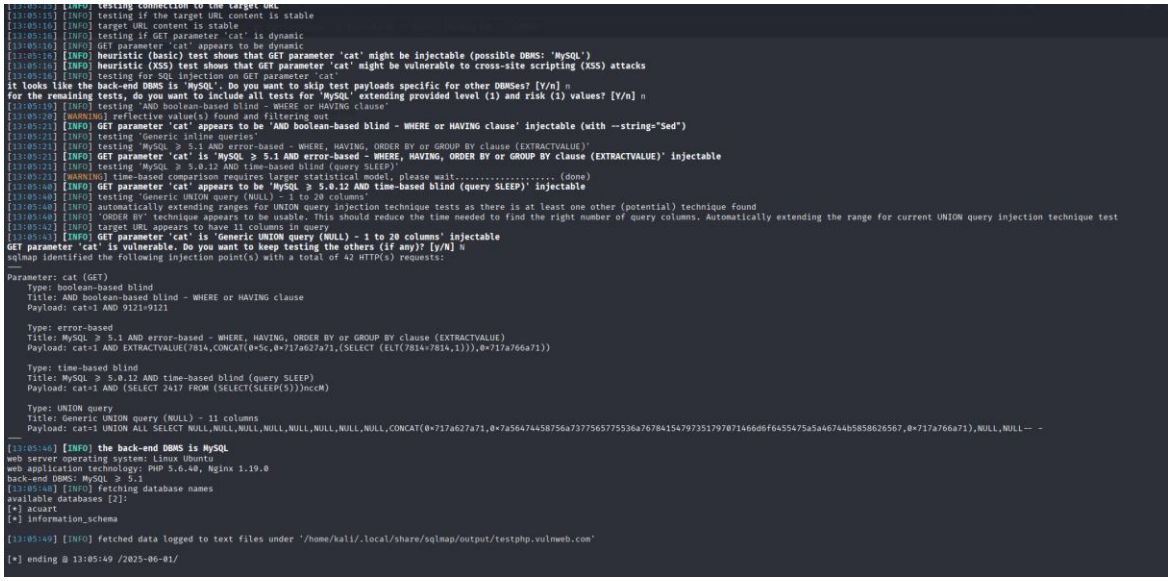


Figure 4.3 database enumeration

Step 2: Table Enumeration in a Targeted Database

Command used:

sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" -D accurate --tables

This command lists all tables within the selected accurate database.

Result:
SQLMAP found **8 tables** in the accurate database. For example:

- artists
- customers
- products
- orders

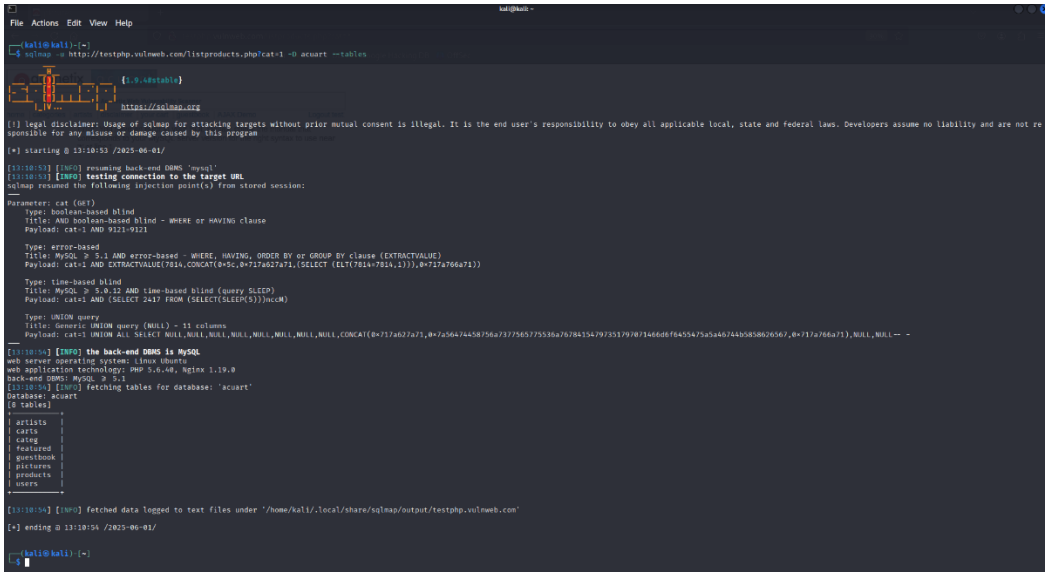


Figure 4.4 DB table

Step 3: Column Enumeration in a Table

We then focused on a table of interest—artists.

Command used:

sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" -D accurate -T artists --columns

This command reveals the structure of the artists table, i.e., the names of the columns it contains.

Result:
SQLMAP returned column names such as:

- id
- name
- email
- genre

.

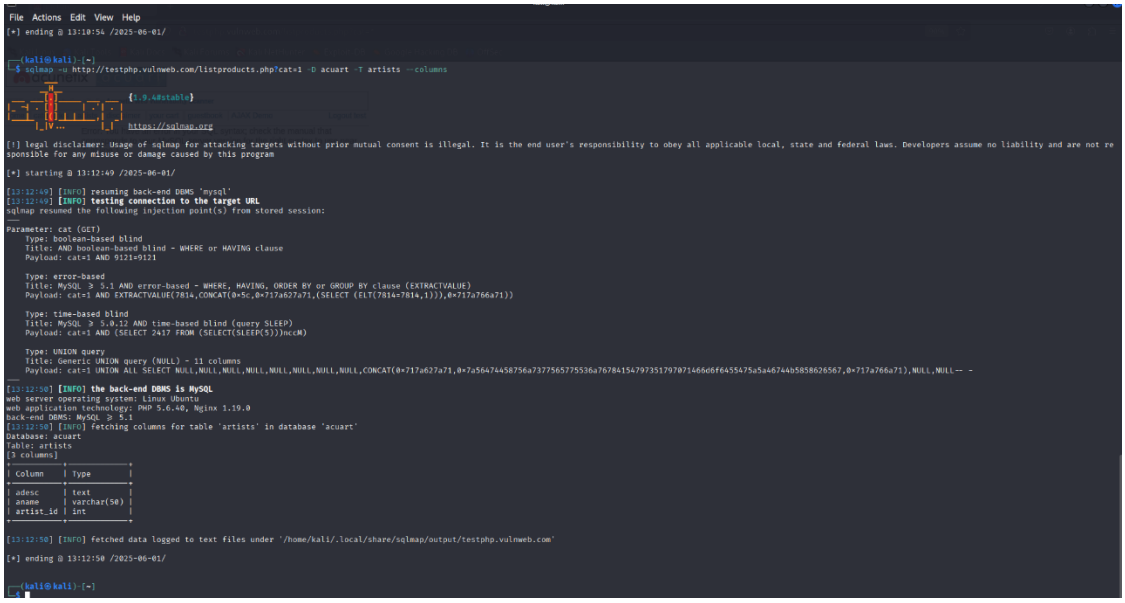


Figure 4.5 enumeration table

Step 4: Data Extraction from Columns

Finally, we attempted to dump the contents of selected columns to demonstrate the impact of the vulnerability.

Command used:

sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" -D accurate -T artists -C name,email --dump

This tells SQLMAP to extract data from the name and email columns in the artists table.

Result:
SQLMAP successfully extracted rows of data containing sensitive information like artist names and their emails. This confirms the SQL Injection vulnerability and shows that an attacker can not only view but also potentially modify or delete database contents.

.

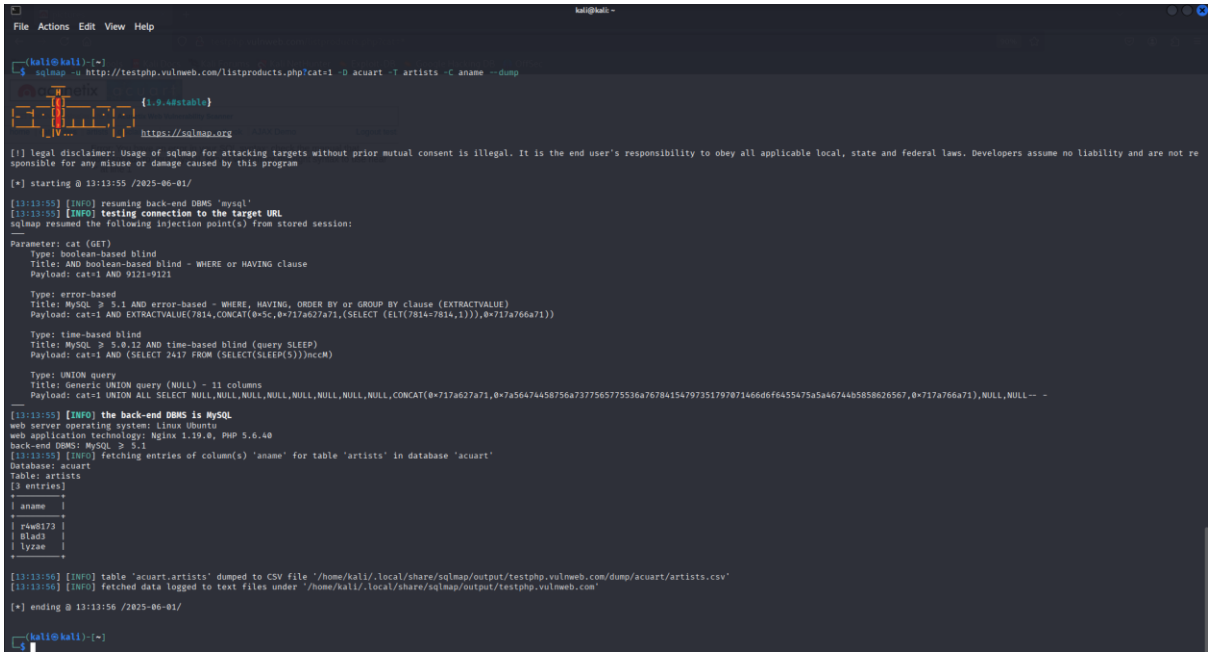


Figure 4.6 Data extraction

Conclusion:

The web application is **critically vulnerable** to SQL Injection. We were able to:

- Identify vulnerable input points
- Access the backend database
- Enumerate databases, tables, and columns
- Extract sensitive data from the database

This demonstrates that an attacker could easily steal, modify, or delete data without proper access. It emphasizes the **need for secure coding practices**, such as:

- Using **parameterized queries** or **prepared statements**
- Validating and sanitizing all user input
- Limiting database permissions
- Employing Web Application Firewalls (WAFs)

Cross-Site Request Forgery (CSRF)

Objective:

To assess the security of the web application against Cross-Site Request Forgery (CSRF) attacks. The goal was to change the password of an authenticated user without their knowledge or consent by exploiting CSRF vulnerabilities at different DVWA security levels (Low, Medium, High).

What is CSRF?

Cross-Site Request Forgery (CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions they do not intend to perform. CSRF exploits the trust that a

web application has in the user's browser. If a user is authenticated on a web application and visits a malicious site in another tab, that site can issue unauthorized requests on behalf of the user.

Step-by-Step Analysis and Exploitation

Step 1: CSRF on DVWA with Low Security Level

At the **low security setting**, DVWA does not implement any CSRF protection mechanisms.

Observation:

When attempting to change the password via the web interface, we observed the following GET request in the URL:

http://localhost:9999/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change#

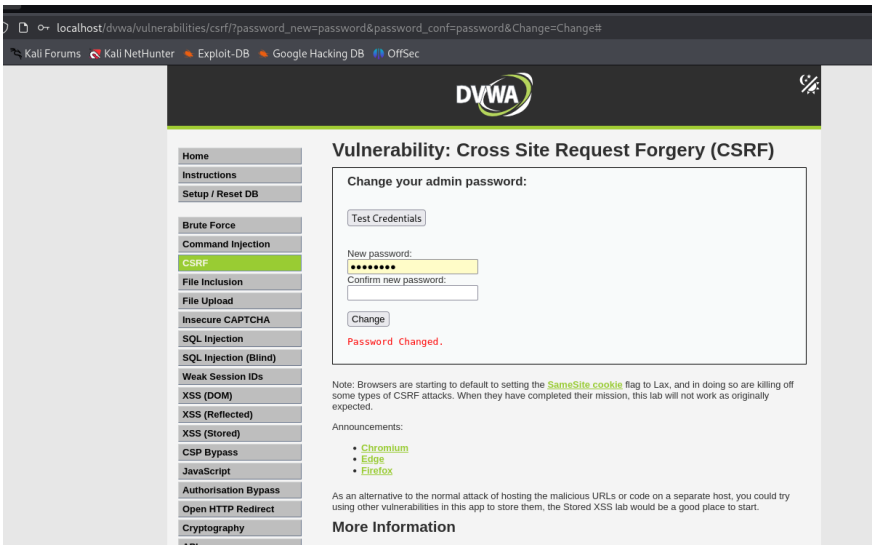


Figure 5.1 low security csrf

This revealed that the password change functionality accepted parameters via a **GET** request, including:

- password_new – the new password
- password_conf – the confirmation password
- Change – a button parameter

Exploit:

An attacker could craft a malicious link like the one above and trick an authenticated user into clicking it:

http://localhost:9999/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change#

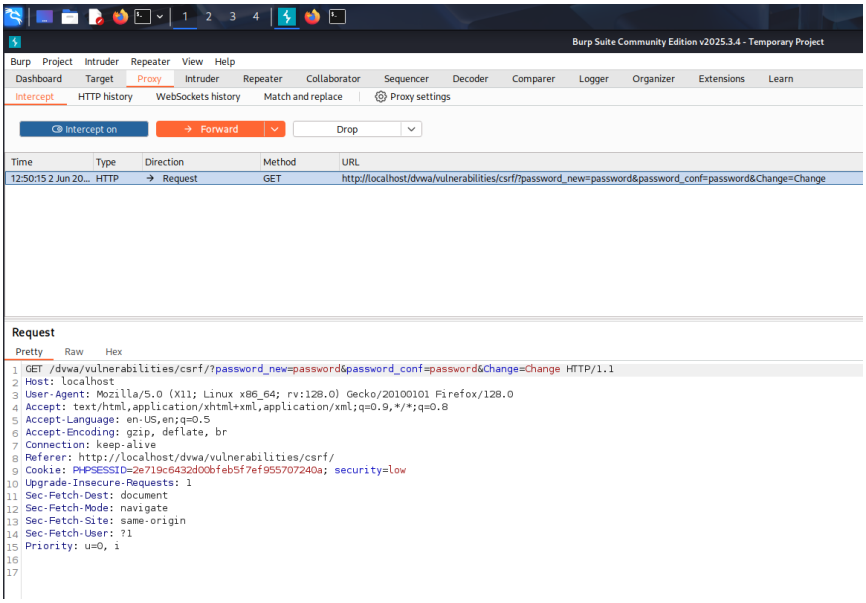


Figure 5.2 burp suite request

If the victim is logged in and clicks this link, their password would be changed to “admin” without any warning or interaction.

Result:

The attack was successful. The credentials were modified, confirming a working CSRF exploit due to the absence of protection mechanisms.

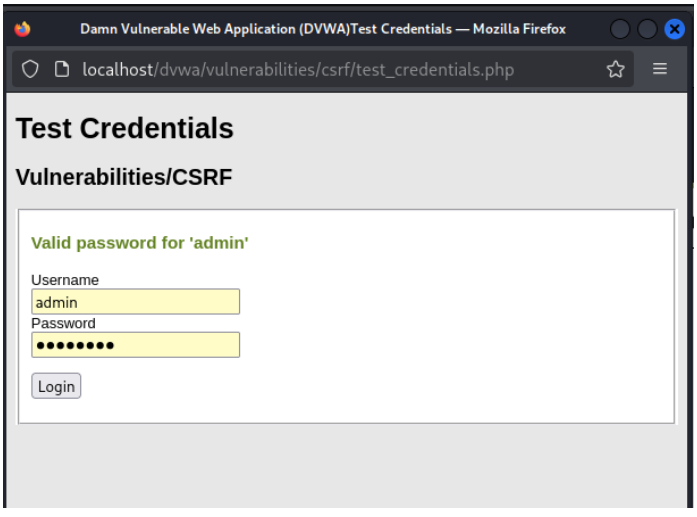


Figure 5.3 succesful log in

Step 2: CSRF on DVWA with Medium Security Level

At the **medium security level**, the application includes a basic check using the **Referer header** to validate the request origin.

Initial Attempt:

Reusing the low-level CSRF payload did not work. The server responded with an error message stating that the request did not appear to be valid.

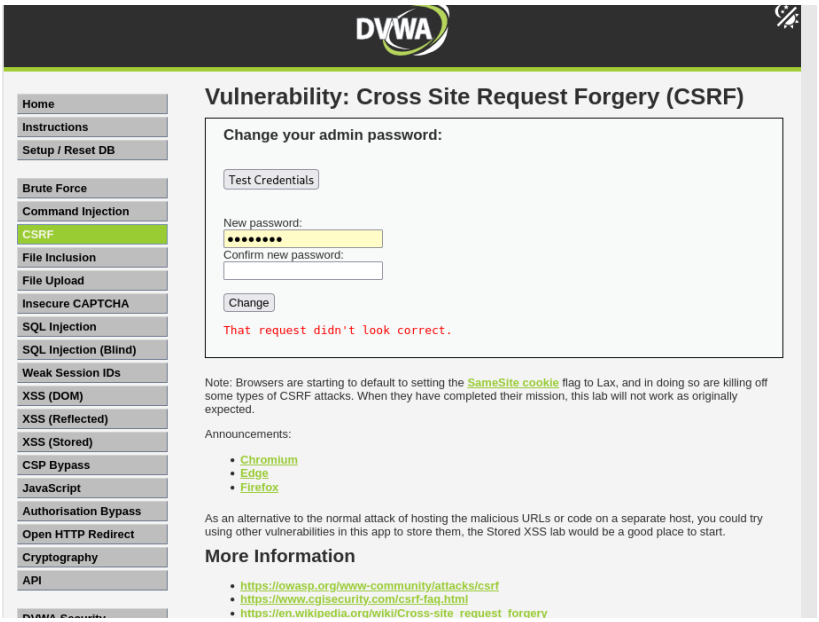


Figure 5.4 medium security csrf

Analysis:

Viewing the source code and analyzing HTTP traffic in **Burp Suite** revealed that the application checks the **HTTP Referer** header to ensure that requests originate from the same site.

```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Checks to see where the request came from
    if( stripos( $_SERVER[ 'HTTP_REFERER' ] ,$_SERVER[ 'SERVER_NAME' ]) !== false ) {
        // Get input
        $pass_new  = $_GET[ 'password_new' ];
        $pass_conf = $_GET[ 'password_conf' ];

        // Do the passwords match?
        if( $pass_new == $pass_conf ) {
            // They do!
            $pass_new = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((trigger_error("[MySQLConverterToo] Fix
            $pass_new = md5( $pass_new );

            // Update the database
            $current_user = dvwaCurrentUser();
            $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . $current_user . "';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_connect_e

            // Feedback for the user
            echo "<pre>Password Changed.</pre>";
        }
        else {
            // Issue with passwords matching
            echo "<pre>Passwords did not match.</pre>";
        }
    }
    else {
        // Didn't come from a trusted source
        echo "<pre>That request didn't look correct.</pre>";
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
}
```

Figure 5.5 code inspection

However, this is a **weak security measure**, as the Referer header can be manipulated.

Bypass:

To bypass this check:

1. The malicious request was intercepted using Burp Suite.
2. The **Referer** header was manually added to simulate a legitimate request:

   http://localhost:9999/vulnerabilities/csrf/
3. The request was forwarded.

Result:

The password change was successful, confirming that the application is still vulnerable to CSRF even at medium security if the attacker can spoof the Referer header.

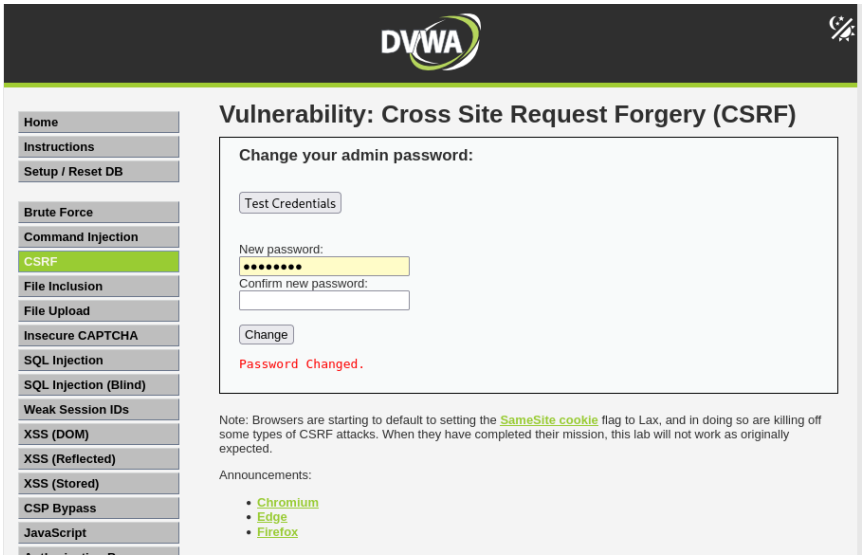


Figure 5.6 succesful log in

Step 3: CSRF on DVWA with High Security Level

At the **high security setting**, DVWA implements a **CSRF token** as a countermeasure. This token:

- Is generated dynamically per session.
- Must be submitted along with the password change request.
- Is validated on the server to ensure authenticity.

Initial Attempt:

Submitting a password change request without a valid CSRF token failed. This confirms the presence of an **anti-CSRF token** mechanism.

Attack Strategy:

To exploit this level, we devised a more advanced attack:

1. **File Upload Vulnerability** was leveraged to upload a malicious HTML file containing the CSRF payload.
2. A file named index.html1 was created with the following functionality:
    - Loads the password change page in an **iframe**
    - Extracts the **anti-CSRF token** using JavaScript
    - Submits a new request including the valid token to change the password

HTML Payload Outline:

html

Copy code

```
<iframe src="http://localhost:9999/vulnerabilities/csrf/" id="frame"
style="display:none;"></iframe>

<script>
  window.onload = function() {
    const iframe = document.getElementById('frame');
    iframe.onload = function() {
      const doc = iframe.contentDocument;
      const token = doc.querySelector('input[name="user_token"]').value;


      fetch('http://localhost:9999/vulnerabilities/csrf/', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        },
        body:
`password_new=admin&password_conf=admin&Change=Change&user_token=${token}`
      });
    };
  };
```

</script>

Execution:

- The malicious HTML file was uploaded to the DVWA server via the file upload feature.
- The uploaded file was accessible at:

http://127.0.0.1:9999/hackable/uploads/index.html1

- When the victim clicked the attacker's link, the malicious page executed in their browser and successfully changed their password using the valid CSRF token.

Result:

The password change was completed silently, even with the CSRF token in place. This demonstrates that CSRF tokens can be bypassed if the attacker can **dynamically retrieve and use them** through vulnerabilities such as file uploads combined with iframe-based scripts.

Figure 5.7 successful uploading

Conclusion:

The experiment demonstrates the escalating complexity of CSRF protection and how:

- **Low-security** implementations are vulnerable to simple GET request exploits.
- **Medium-security** implementations using **Referer headers** are still bypassable.
- **High-security** implementations using **anti-CSRF tokens** are stronger but can be bypassed using **sophisticated multi-step attacks**, especially when combined with other vulnerabilities (e.g., file upload flaws).

This underlines the importance of:

- Enforcing **POST-only methods** for sensitive actions
- Implementing **robust anti-CSRF token validation**
- Sanitizing file uploads to prevent **arbitrary script execution**
- Applying **Content Security Policy (CSP)** and same-origin policies

XSS – Cross-Site Scripting

Cross-Site Scripting (XSS) is a client-side vulnerability that allows attackers to inject malicious scripts into websites viewed by other users. These scripts are often used to steal session cookies, manipulate the DOM, or redirect users to malicious pages. XSS occurs when user input is not properly sanitized and is rendered directly into a web page.

Types of XSS:

1. **Reflected XSS** – Malicious script is reflected off a web server and executed immediately in the user's browser.
2. **Stored XSS** – Malicious script is permanently stored on the target server and executed whenever the page is viewed.
3. **DOM-Based XSS** – Malicious script is executed by modifying the DOM environment on the client side without changing the HTTP response from the server.

1. Reflected XSS

In Reflected XSS, the malicious payload is embedded into a URL and sent to a victim. The server reflects the payload back in the HTTP response without proper sanitization, allowing it to be executed in the browser.

DVWA – Reflected XSS:

Low Security:

- **Behavior**: Input is directly reflected without any filtering or sanitization.
- **Payload Used**:

  <script>alert('Reflected XSS')</script>
- **Result**: JavaScript alert box pops up immediately after submission.

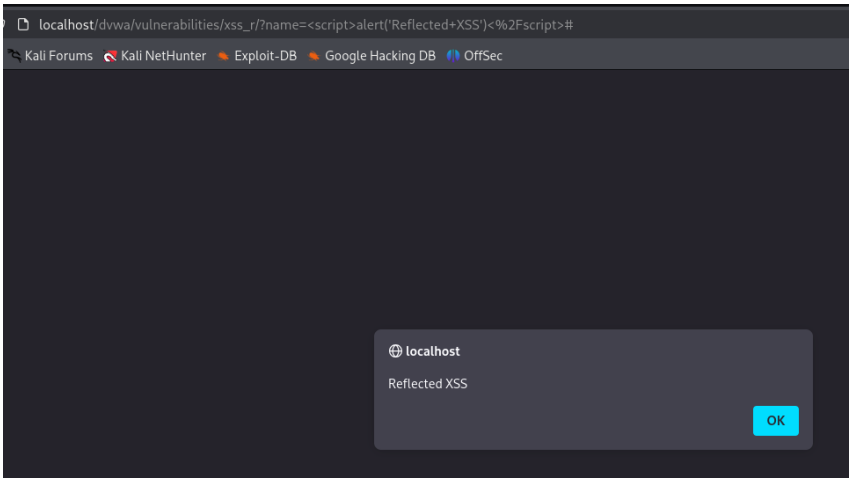


Figure 6.1 low xss reflected

Medium Security:

- **Behavior**: Some sanitization applied, but still vulnerable to alternative payloads.
- **Payload Used**:

  <img src=x onerror=alert('XSS')>
- **Result**: Alert executed successfully via image load error.

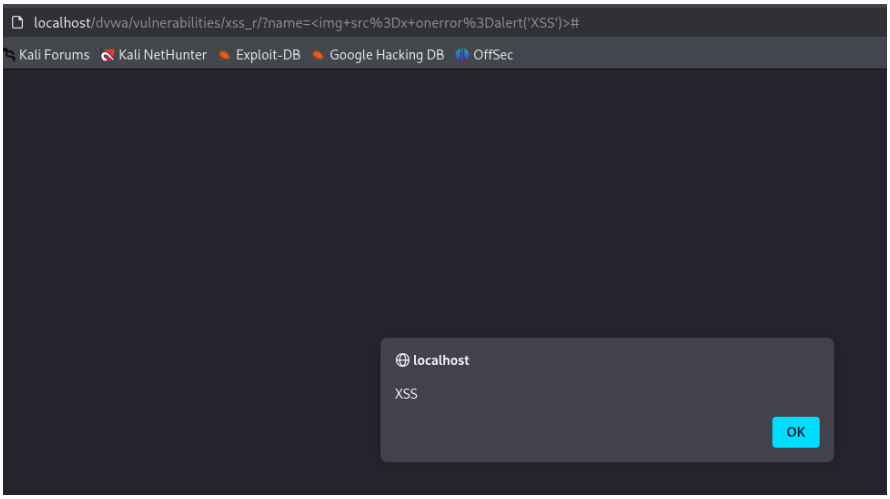


Figure 6.2 medium xss reflected

High Security:

- **Behavior**: Server applies stronger input validation, but can still be bypassed with creative payloads.
- **Payload Used**:

  <img src=x onerror=alert('XSS')>
- **Result**: XSS payload executed by exploiting image error handling.

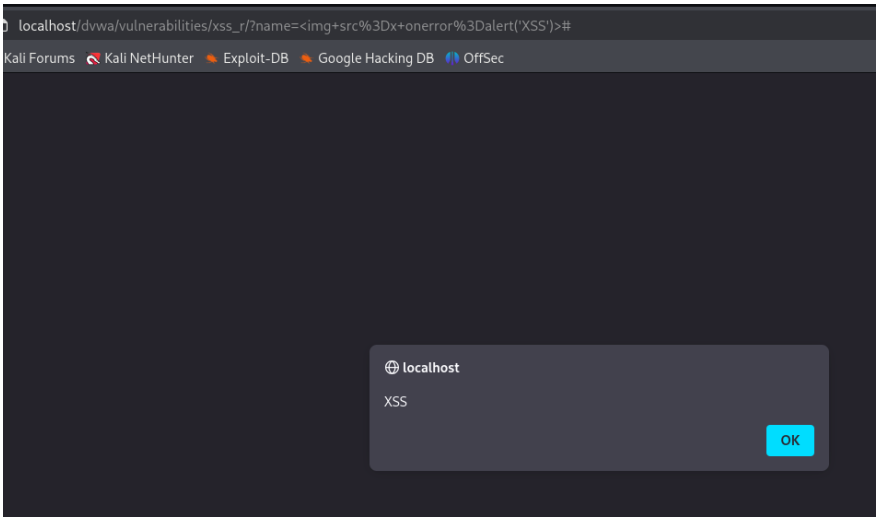


Figure 6.3 high xss reflected

2. Stored XSS

In Stored XSS, the attacker's payload is saved on the server (e.g., in comment or username fields). When another user loads the page, the malicious script executes automatically.

DVWA – Stored XSS:

Low Security:

- **Behavior**: No sanitization of input; scripts are stored and executed upon page load.
- **Payload Used**:

  <script>alert('Stored XSS')</script>
- **Result**: Every user loading the page sees the alert.

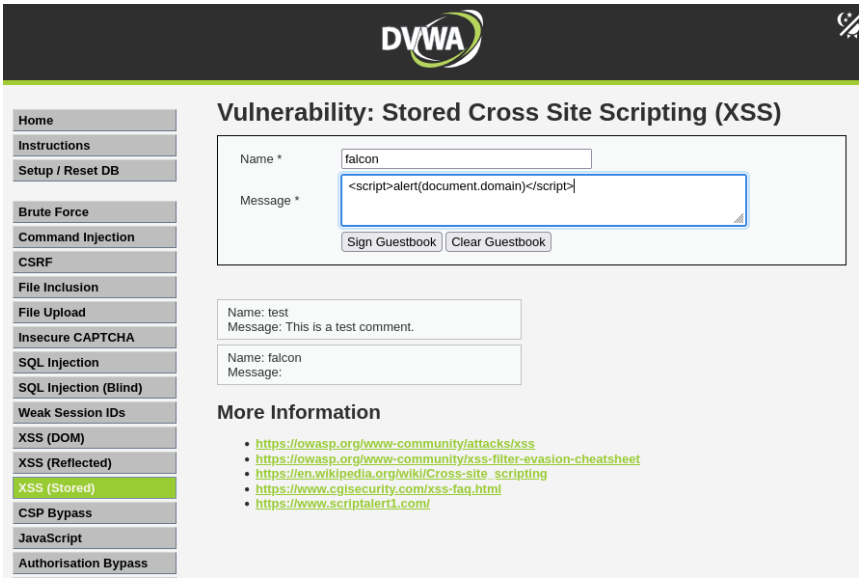


Figure 6.4 low stored

Medium Security:

- **Behavior**: Partial filtering applied, but alternate payloads can still bypass.
- **Payload Used**:

  <img src=x onerror=alert('Stored XSS')>

- **Result**: Payload triggered when the page is reloaded.

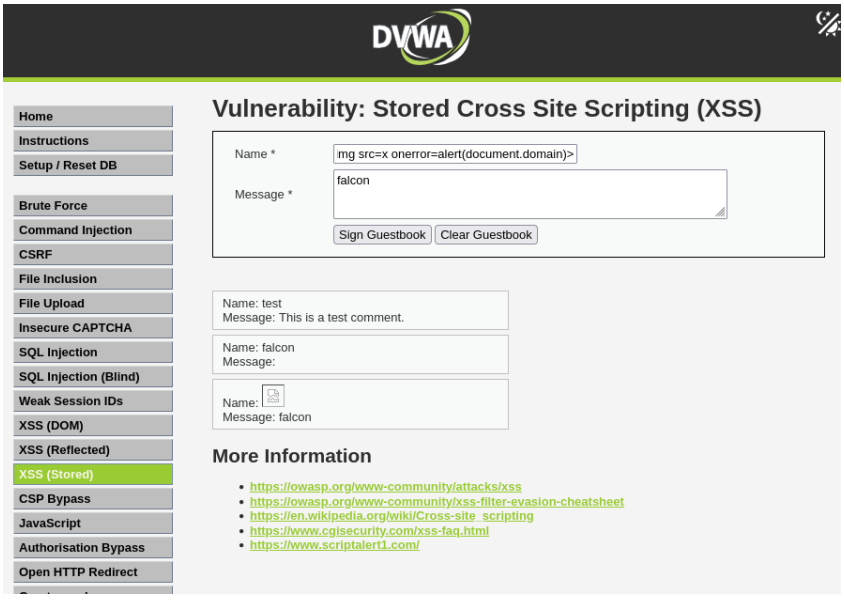


Figure 6.5 medium stored

High Security:

- **Behavior**: Sanitization is applied, but encoded or obfuscated payloads may still execute depending on implementation.
- **Payload Used** (if bypassed):

- **Result**: JavaScript executed if the filter doesn't sanitize SVG tag properly.

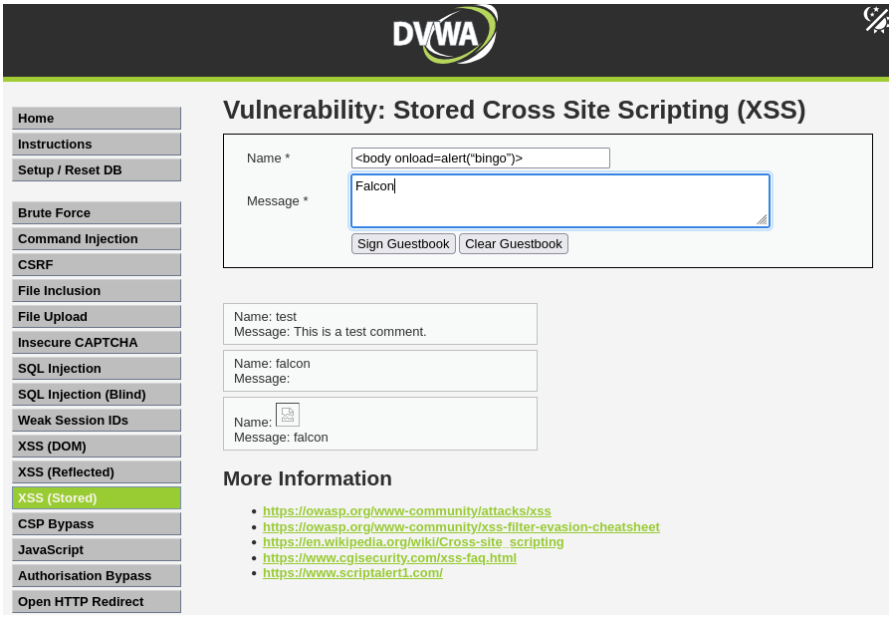


Figure 6.6 high stored

3. DOM-Based XSS

In DOM-Based XSS, the vulnerability arises entirely on the client side. The server returns static pages, and the payload gets executed through insecure handling of user input in JavaScript code.

DVWA – DOM XSS:

Low Security:

- **Payload**:

  http://localhost/dvwa/vulnerabilities/xss_d/?default=<script>alert(1)</script>
- **Execution**: The script runs as soon as the URL is loaded.

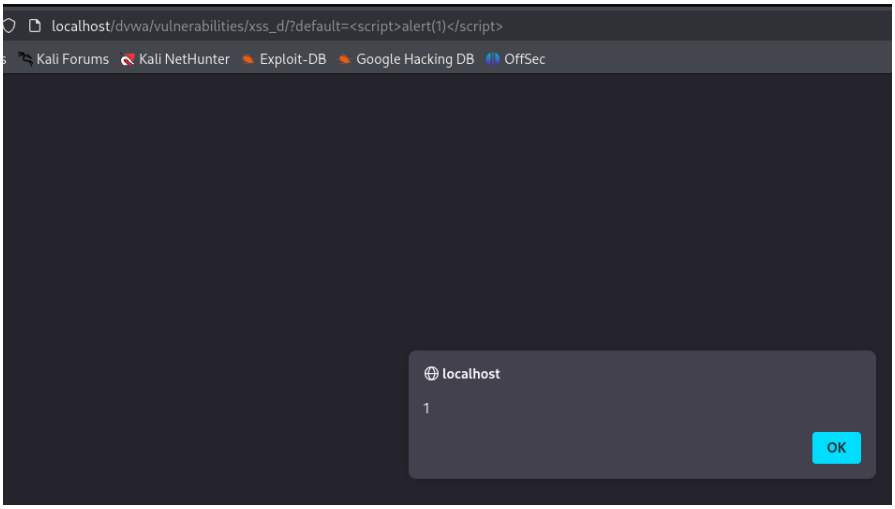


Figure 6.7 low dom

Medium Security:

- **Payload**:

http://localhost/dvwa/vulnerabilities/xss_d/?default=English#<script>alert(1)</script>

- **Execution**: Requires a page reload; the payload in the URL hash triggers the alert.

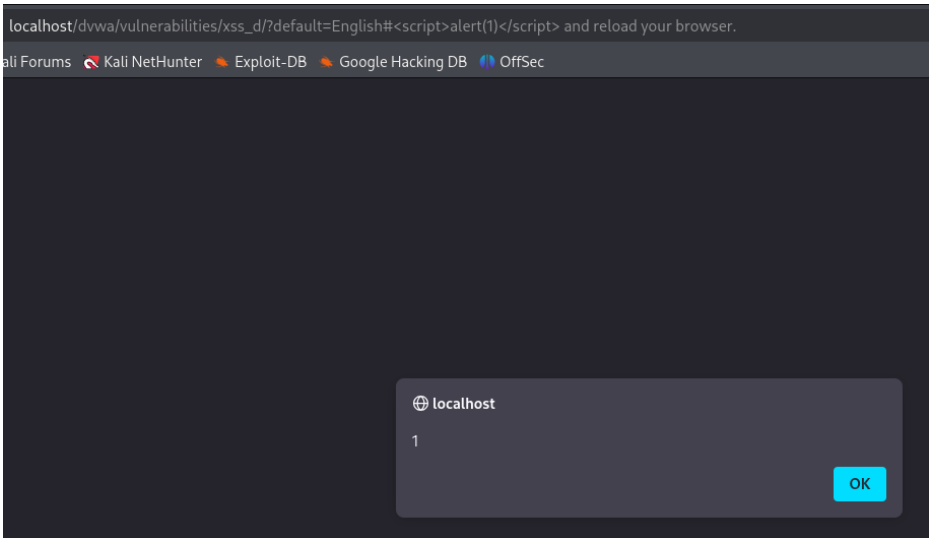


Figure 6.8 medium dom

High Security:

- **Payload**:

http://localhost/dvwa/vulnerabilities/xss_d/?default=English#<script>alert(document.cookie)</script>

- **Execution**: On reload, this script displays the victim's cookies, showing the ability to steal session tokens or sensitive data.

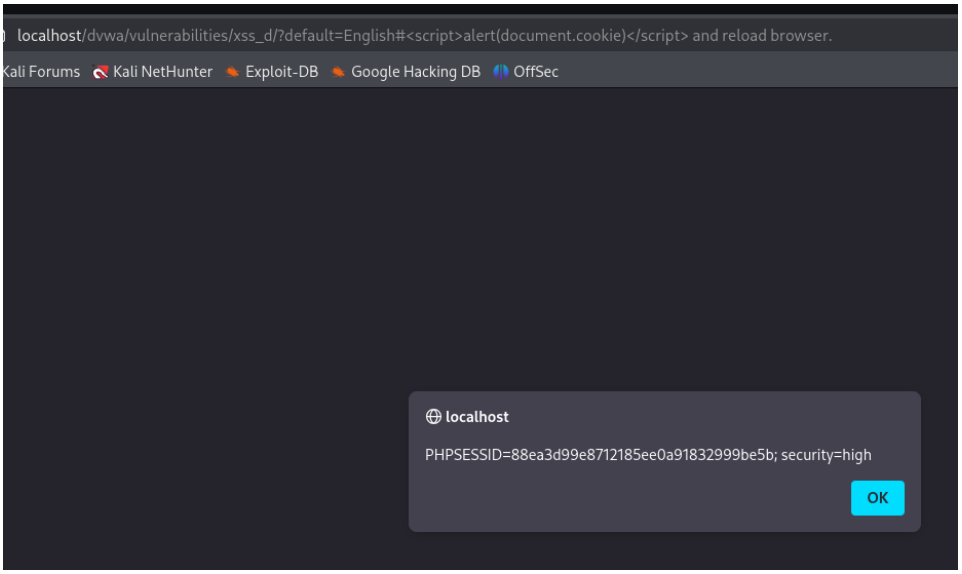


Figure 6.9 high dom

Summary

XSS Type	Low Security	Medium Security	High Security
Reflected	<script>alert()</script>	<img src=x onerror=alert()>	<img src=x onerror=alert()>
Stored	<script>alert()</script>	<img src=x onerror=alert()>	<svg onload=alert()> (if applicable)
DOM-Based	Payload in URL parameter	Payload in hash (#)	Payload with document.cookie access

Conclusion

XSS vulnerabilities in DVWA demonstrate the importance of input validation and output encoding. While low-security settings are trivially vulnerable, even medium and high settings can often be bypassed with slightly modified payloads. Proper defense involves using secure development practices, such as context-aware escaping, Content Security Policy (CSP), and frameworks with built-in XSS protection.

3.5 Post-Exploitation

The objective of the **Post-Exploitation** phase is to assess the impact of vulnerabilities previously exploited in the web applications. This phase involves simulating real-world attacker behavior after successful exploitation, such as credential extraction, session hijacking, unauthorized data access, and privilege escalation.

1. SQL Injection – testphp.vulnweb.com

Exploit Description:

We exploited an SQL Injection vulnerability in user input fields that allowed interaction with the backend database without proper authorization.

Post-Exploitation Action:

Using this access, we extracted sensitive data, including usernames and password hashes from the database. We then attempted to access protected areas of the application using these credentials.

Outcome:

This vulnerability confirmed that attackers could bypass authentication and view confidential user data, which could lead to full system compromise if it were a live application.

2. Cross-Site Scripting (XSS) – DVWA (localhost)

Exploit Description:

We found a reflected XSS vulnerability in DVWA, where user input was reflected on the page without being properly sanitized.

Post-Exploitation Action:

We simulated stealing session cookies from a victim by injecting malicious scripts. These session tokens allowed us to impersonate legitimate users without needing passwords.

Outcome:

This vulnerability showed that attackers could hijack user sessions, bypassing authentication and accessing user-specific data or features.

3. Cross-Site Request Forgery (CSRF) – DVWA (localhost)

Exploit Description:

DVWA was vulnerable to CSRF attacks, where malicious requests could be made on behalf of authenticated users without their knowledge.

Post-Exploitation Action:

We demonstrated this by successfully changing a user's password while they were logged in, without their interaction.

Outcome:

This highlighted a major risk, as attackers could perform unauthorized actions like modifying credentials, changing settings, or even deleting data.

4. Brute Force Login – DVWA and testphp.vulnweb.com

Exploit Description:

We attempted to guess login credentials using brute-force techniques, targeting the login pages.

Post-Exploitation Action:

We managed to discover weak username-password combinations and gain unauthorized access to user accounts.

Outcome:

The application lacked security measures like account lockout or rate limiting, making brute-force attacks effective. This could be used to compromise multiple accounts.

5. Directory Listing & Information Disclosure – DVWA (localhost)

Exploit Description:

Several directories in DVWA were exposed due to improper server configuration. Directory indexing was enabled, revealing internal files and folders.

Post-Exploitation Action:

We accessed files like .git/config, .dockerignore, and /database/, which exposed sensitive data about the application's internal structure and environment.

Outcome:

These findings demonstrated that attackers could collect useful reconnaissance data, review application logic, or find credentials stored in misconfigured files.

The post-exploitation analysis provided clear evidence of how each exploited vulnerability could escalate to a more severe compromise. In real-world scenarios, these would pose significant security risks including:

- Unauthorized data access
- Session hijacking
- Account takeovers
- Exposure of internal application data
- Full system compromise

3.6 Reporting & Remediation

1. Final Report Writing

We compile **everything you've done** in a **professional report** format. It includes:

- **Executive Summary**: High-level overview for non-technical stakeholders.
- **Technical Details**: Step-by-step details of recon, scanning, exploitation, post-exploitation.
- **Vulnerabilities Found**: Clearly listed with CVSS scores if applicable.
- **Impact Assessment**: What could happen if the vulnerabilities were exploited.
- **Proof of Concept**: Screenshots or summarized outputs (no raw code).
- **Tools Used**: Mention tools like Nmap, Nikto, Burp Suite, DVWA, etc.
- **Mitigation Recommendations**: How to fix or reduce the risk.

2. Remediation Suggestions

You give the target system's admin or client a **to-do list** to fix the vulnerabilities. For example:

- Apply input validation to stop SQLi.
- Enable X-Frame-Options headers to prevent clickjacking.
- Enforce password policies to stop brute-force attacks.
- Restrict directory access in Apache settings.

CHAPTER 4

CONCLUSION

4.1 Conclusion

This project explored the critical domain of **web application security testing** through a structured penetration testing process. In today's digital landscape, where web applications play a central role in delivering services and data, ensuring their security has become more essential than ever. This report aimed to demonstrate how real-world vulnerabilities can be identified and analyzed using industry-standard techniques and tools in a simulated environment.

The tests were performed in a secure and ethical lab setup using **Kali Linux**, where simulated attacks were launched against a purposely vulnerable web application. This practical experience highlighted not only how security flaws manifest but also how easily they can be overlooked during development if security is not embedded from the start.

Among the significant findings was a **CSRF (Cross-Site Request Forgery) vulnerability**, which was exploited to perform unauthorized actions on behalf of authenticated users. This vulnerability underscored the importance of implementing proper **anti-CSRF tokens** and validating requests on the server side. Additionally, **directory listing** and **server information leakage** were discovered, showcasing how misconfigurations can provide attackers with valuable information for launching further attacks.

The project has proven to be highly valuable in applying theoretical knowledge in a real-world context. It reinforced several important security principles, such as **the need for layered defenses**, **input validation**, **secure session management**, and the importance of **security-aware development practices**. It also highlighted how penetration testing is not just a one-time task but should be a recurring part of the software development lifecycle (SDLC).

In conclusion, this project has successfully demonstrated how penetration testing can serve as a proactive approach to identifying and mitigating vulnerabilities in web applications. The knowledge gained through this exercise will contribute significantly to future endeavors in the cybersecurity domain, whether in professional practice or academic research. With the increasing sophistication of cyber threats, regular security testing, developer training, and adherence to best practices are indispensable in safeguarding today's digital assets.

REFERNCES

1. OWASP Foundation. (n.d.). *OWASP Testing Guide v4*. Retrieved from https://owasp.org/www-project-web-security-testing-guide/
2. Scarfone, K., & Mell, P. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)*. National Institute of Standards and Technology (NIST). https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf
3. Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). Wiley Publishing.
4. Nmap. (n.d.). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Retrieved from https://nmap.org/book/
5. Kali Linux. (n.d.). *Kali Tools*. Offensive Security. Retrieved from https://www.kali.org/tools/
6. PortSwigger Web Security Academy. (n.d.). *Learning Path: Web Security*. Retrieved from https://portswigger.net/web-security
7. OWASP Foundation. (n.d.). *Cross-Site Request Forgery (CSRF)*. Retrieved from https://owasp.org/www-community/attacks/csrf
8. Kottakota, S., & Beebe, N. (2010). "Web Application Penetration Testing: A Case Study". *International Journal of Cyber-Security and Digital Forensics*, 1(3), 191–197.