Table of Contents

Table of Figures

1.0 Introduction

In the ever-evolving landscape of cybersecurity threats, Vulnerability Assessment and Penetration Testing (VAPT) is a critical approach to identifying and mitigating security risks in web applications. This assessment focuses on evaluating the security posture of Damn Vulnerable Web Application (DVWA) which is a deliberately insecure web application designed for security professionals and researchers to test and improve their penetration testing skills. By conducting a comprehensive VAPT, this assessment aims to uncover potential vulnerabilities and assess their exploitability to strengthen overall web application security.

To ensure a structured and effective security evaluation, OWASP ZAP functions as the chosen tool for Vulnerability Assessment (VA) to execute automated checks for OWASP Top 10 risks that affect web applications. During Penetration Testing, Burp Suite and SQLMap act as tools for security assessment to validate SQL Injection (SQLi) and Cross-Site Request Forgery (CSRF) attacks and various other security issues against real-world scenarios. This evaluation within its security assessment uses standardized tools from the industry to deliver thorough security results which generate practical recommendations for defending web applications against cyber-attacks.

1.1 Penetration Testing Purpose

The primary purpose of this penetration testing assessment is to evaluate the security posture of the Damn Vulnerable Web Application (DVWA) by identifying and exploiting vulnerabilities commonly found in real-world web applications. By simulating the actions of a potential attacker, this assessment aims to demonstrate how security weaknesses within the application can be exploited and how organizations can implement necessary countermeasures to mitigate risks. Through this approach, the assessment provides valuable insights into the effectiveness of security controls, allowing organizations to enhance their overall security posture.

A key objective of this penetration test is to identify and exploit vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), Command Injection, and Authentication Bypass to assess their potential impact on a web application. By doing so, this test evaluates the robustness of existing security controls, including authentication mechanisms, input validation, and server configurations. Additionally, the assessment aims to determine the effectiveness of implemented security defenses and identify areas where improvements are needed.

This penetration test is conducted following **ethical hacking principles** and within a **controlled and legal environment** to ensure no real-world systems are affected. By leveraging **Burp Suite and SQLMap** as penetration testing tools, this assessment simulates realistic attack scenarios, allowing security professionals to proactively strengthen the application's security. The insights gained from this test will help organizations build **more resilient web applications** and defend against emerging cybersecurity threats effectively.

1.2 Target Application (DVWA)

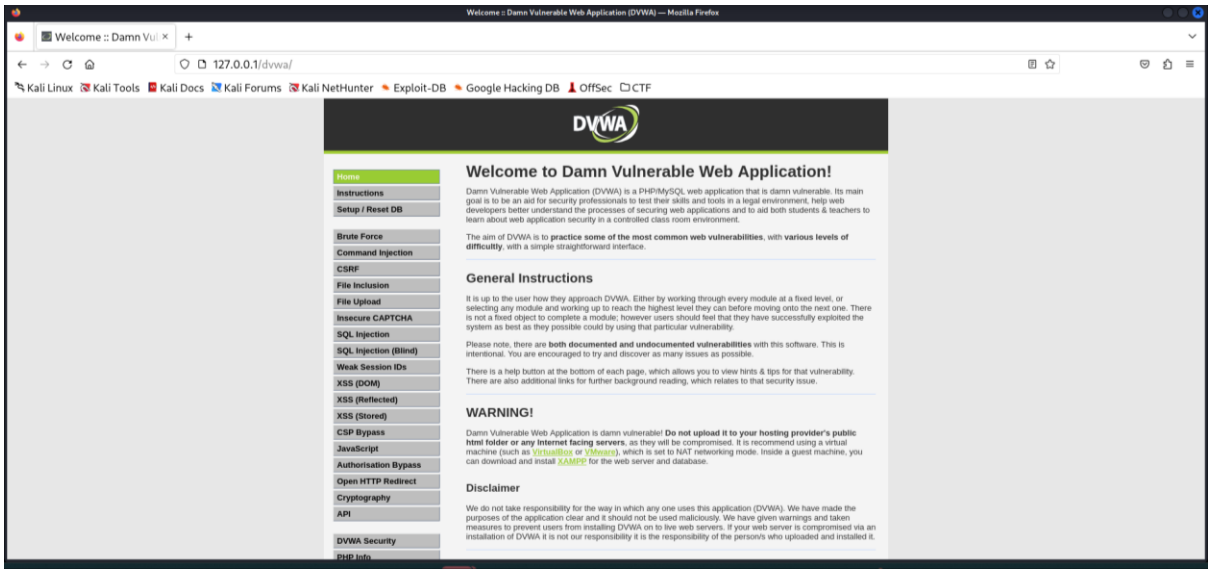


Figure 1: Target Application – DVWA

Damn Vulnerable Web Application (DVWA) is a **deliberately vulnerable web application** designed for cybersecurity professionals, penetration testers, and security researchers to practice and test web application security vulnerabilities. It provides a safe and legal environment to develop and refine penetration testing skills by exploiting commonly known security flaws.

DVWA is a **PHP/MySQL-based web application** that runs on a **Linux-based operating system with Apache as its web server**. It includes multiple security levels—Low, Medium, and High—allowing testers to analyse vulnerabilities under different security conditions. The application is structured to contain various **OWASP Top 10 vulnerabilities**, including **SQL Injection (SQLi), Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Command Injection, and Broken Authentication**.

In this assessment, **OWASP ZAP** will be used for **Vulnerability Assessment (VA)** to identify weaknesses in DVWA, while **Burp Suite and SQLMap** will be used for **Penetration Testing (PT)** to exploit vulnerabilities and evaluate the security impact. The findings from this assessment will provide insights into securing web applications against real-world threats.

1.3 Exploitation Plan, Involvement, Requirements

This Vulnerability Assessment and Penetration Testing (VAPT) assessment is structured to evaluate the security posture of Damn Vulnerable Web Application (DVWA) by identifying and exploiting vulnerabilities commonly found in web applications. The assessment will follow a well-defined methodology, ensuring a comprehensive security evaluation while maintaining an ethical and controlled testing environment.

The assessment will be conducted within a predefined timeframe, allowing sufficient time for reconnaissance, vulnerability assessment, exploitation, and reporting. The selected testing tools, including OWASP ZAP, Burp Suite, and SQLMap, will be utilized to detect and exploit security weaknesses within DVWA.

Start Date	Wednesday, 19 March, 2025, 11:06:31 PM
End Date	Wednesday, 26 March, 2025, 7:31:11 PM
Attacker Machine	The **attacker machine** used in this assessment is **Kali Linux**, a widely used penetration testing distribution that comes pre-installed with numerous security tools. Kali Linux provides an optimal environment for conducting both **vulnerability assessment** and **penetration testing** efficiently.
Target Application	The **target system** is **Damn Vulnerable Web Application (DVWA)**, a deliberately vulnerable web application designed for security testing and ethical hacking practice. It will be hosted in a **controlled environment** using a **Virtual Machine (VM)** running a Linux-based server.
Testing Tools	This assessment utilizes active scanning and exploitation tools such as **OWASP ZAP** which automatically scans the security flaws within web applications, **Burp Suite** intercepts, manipulates, and exploits web requests for vulnerabilities such as XSS, CSRF, and authentication flaws and **SQLMap** that automates SQL injection attacks to extract database information.
Web Server and Database	The **DVWA application** will be hosted on a **Linux-based Apache2 web server** running either within a **virtual machine**. The application also utilizes **MySQL/MariaDB** backend database for database management.

1.3.1 DVWA Environment Setup

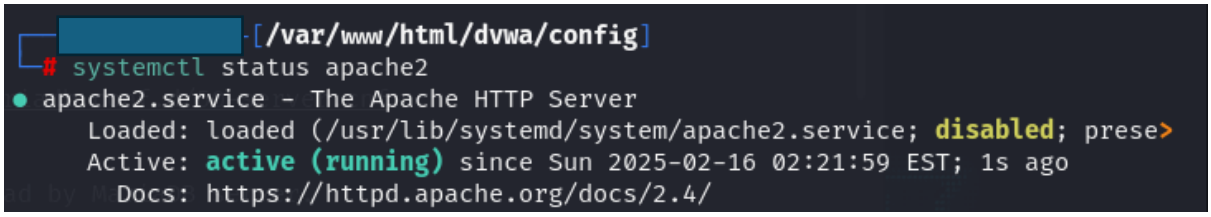


Figure 2: DVWA Setup 1

Figure 3: DVWA Setup 2

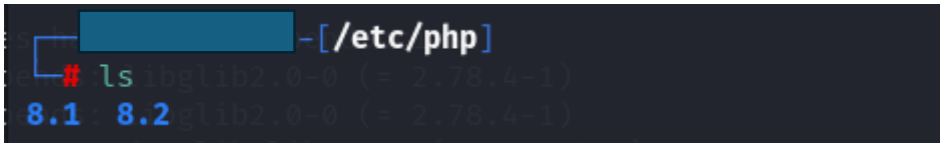


Figure 4: DVWA Setup 3

DVWA (Damn Vulnerable Web Application) is a **PHP/MySQL-based web application** designed for security practitioners to practice **web application vulnerability assessments** and penetration testing. It is intentionally vulnerable, providing a controlled environment to test various security flaws such as **SQL Injection, Cross-Site Scripting (XSS), CSRF, and Command Injection**. Hence, this requires the setup of services such as **php, apache2 and mysql**.

2.0 Vulnerability Assessment

The **Vulnerability Assessment (VA)** phase is a crucial step in evaluating the security weaknesses present in the **Damn Vulnerable Web Application (DVWA)**. This section demonstrates vulnerability detection through the OWASP ZAP security tool to identify weaknesses in web applications as an accepted industry-standard security tool. A complete automated and manual assessment process exists to reveal crucial security flaws that could serve as entry points for attackers if systems maintain their existing vulnerabilities.

The scanning process will include both passive and active scanning techniques to identify vulnerabilities related to injection attacks, authentication flaws, misconfigurations, broken access control, and insecure design. The discovered vulnerabilities will be mapped to the OWASP Top Ten 2021 categories and cross-referenced with Common Weakness Enumeration (CWE) to provide an in-depth understanding of their severity and potential impact on the system.

2.1 OWASP ZAP on Kali Linux

Figure 5: OWASP ZAP (Ashwani K, 2023)

OWASP ZAP (Zed Attack Proxy) is an open-source web application security scanner that is widely used to detect vulnerabilities in web applications. It is designed for security professionals and developers to help them find web application security flaws like SQL injection, cross-site scripting (XSS), broken authentication and security misconfigurations, with a focus on the Open Web Application Security Project (OWASP) Top Ten. It supports **automated and manual testing**, making it an essential tool for **vulnerability assessment (VA)**. By utilizing features such as **passive and active scanning, fuzzing, spider crawling and**

proxy interception, OWASP ZAP provides a comprehensive approach to identifying security weaknesses and improving web application security.

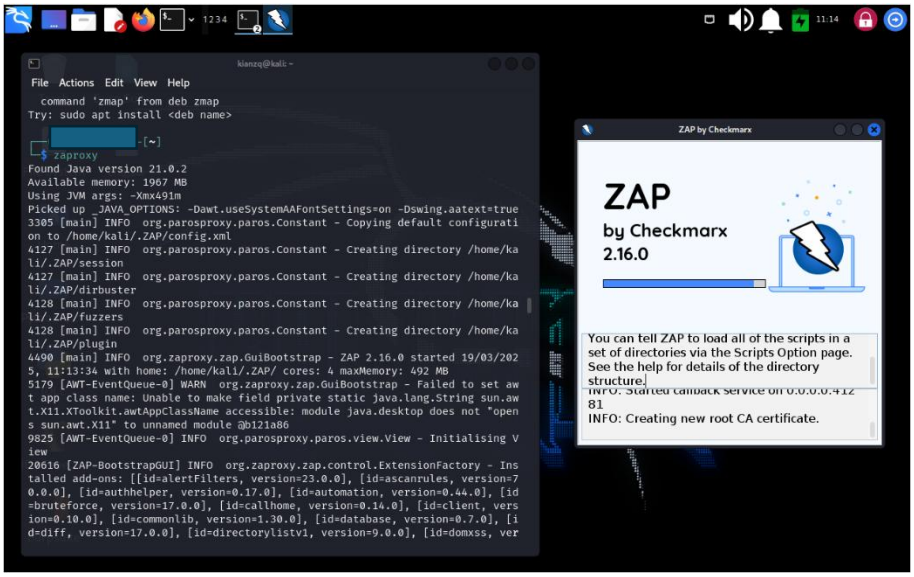


Figure 6: Execute OWASP ZAP through Command

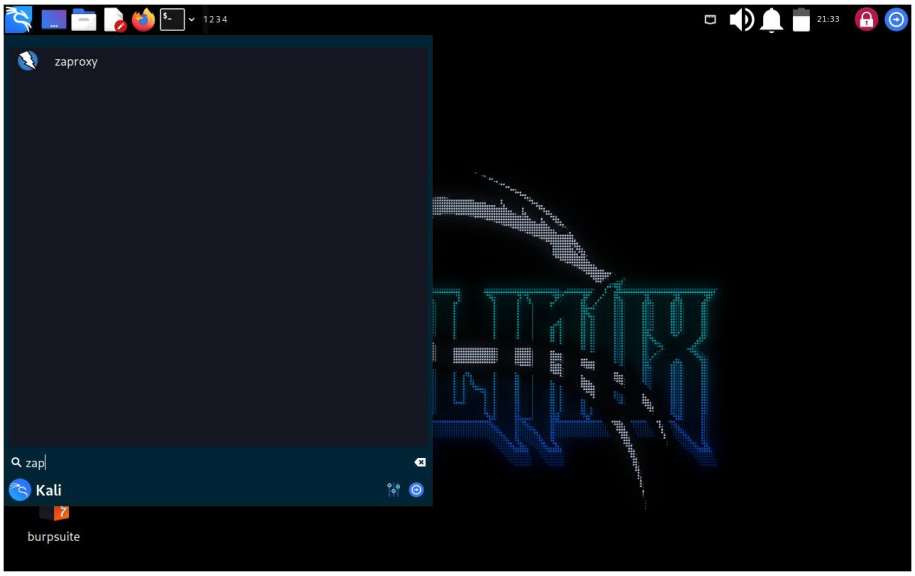


Figure 7: Execute OWASP ZAP through Kali Panel

OWASP ZAP as an open-source tool is given free installation within Linux environments, including the Kali Linux machine, which can be executed through the command ‘zaproxy’ or search application on the Kali Linux panel.

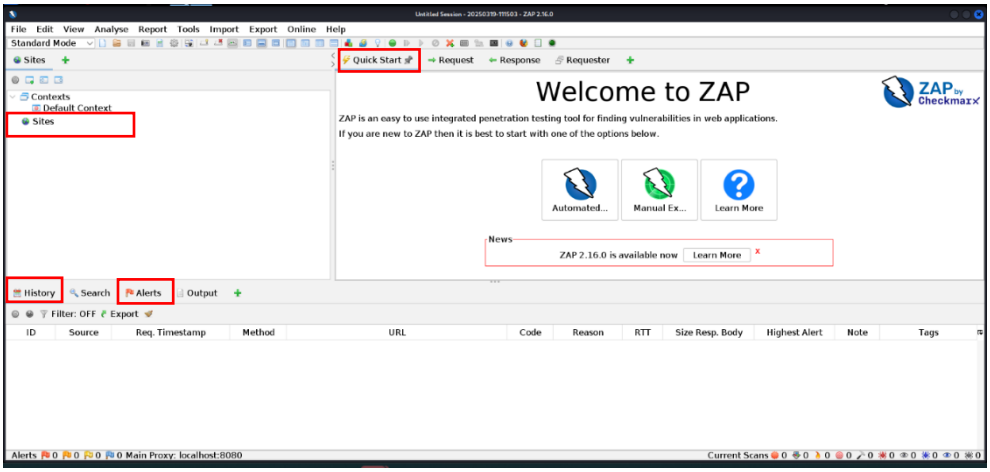


Figure 8: OWASP ZAP Interface

OWASP ZAP is designed with a user-friendly interface that allows easier navigation between features. This includes core features such as Sites that show all discovered directories within the target application, History that shows all scanned paths and URLs and Alerts that show discovered vulnerabilities. At the Quick Start panel, both Automated Scan and Manual Explore features will be shown.

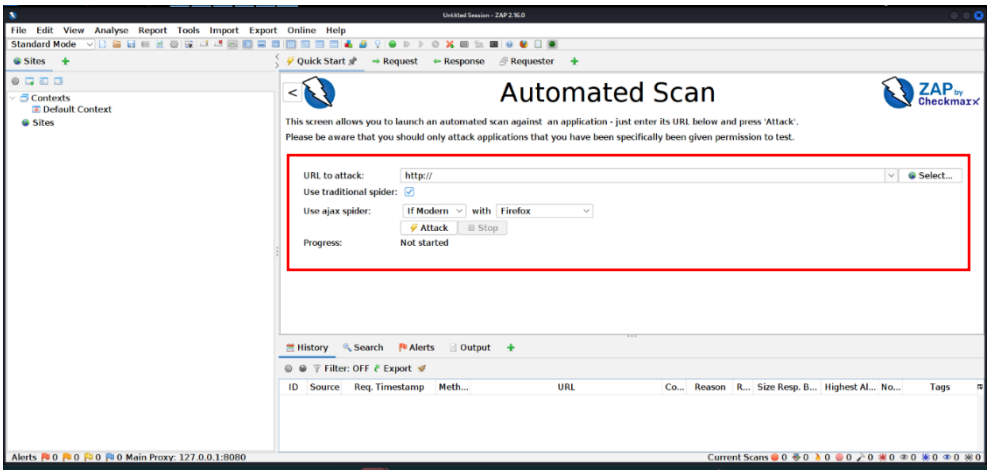


Figure 9: OWASP ZAP Automated Scan

To carry out automated scanning on target applications, the user will be prompted to enter details of the target which include the path or URL to attack, and the option of scanning with or without additional tools such as Spider and AJAX Spider. The attack button will be used to interpret the scanning.

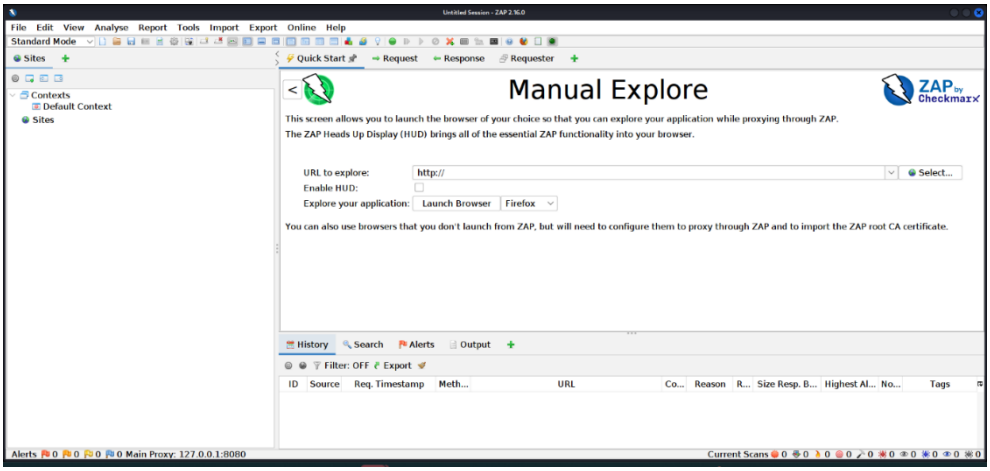


Figure 10: OWASP ZAP Manual Explore

Similar to Automated Scan, the application will prompt the user to enter the preferred scanning target path or URL in the Manual Explore session. This includes the options given with or without Heads Up Display (HUD) and browser to launch the target website or application. Manual exploration of the target application will be interpreted upon clicking on the Launch Browser button. In OWASP ZAP, Manual Explore allows security testers to interact with the web application manually, navigating through different pages and triggering various functionalities. This approach helps identify vulnerabilities that automated scans might miss, ensuring a more comprehensive assessment by simulating real user behaviour and manually testing inputs for security flaws. However, in this assessment, the Manual Explore feature will not be utilized.

2.2 Understanding DVWA Architecture

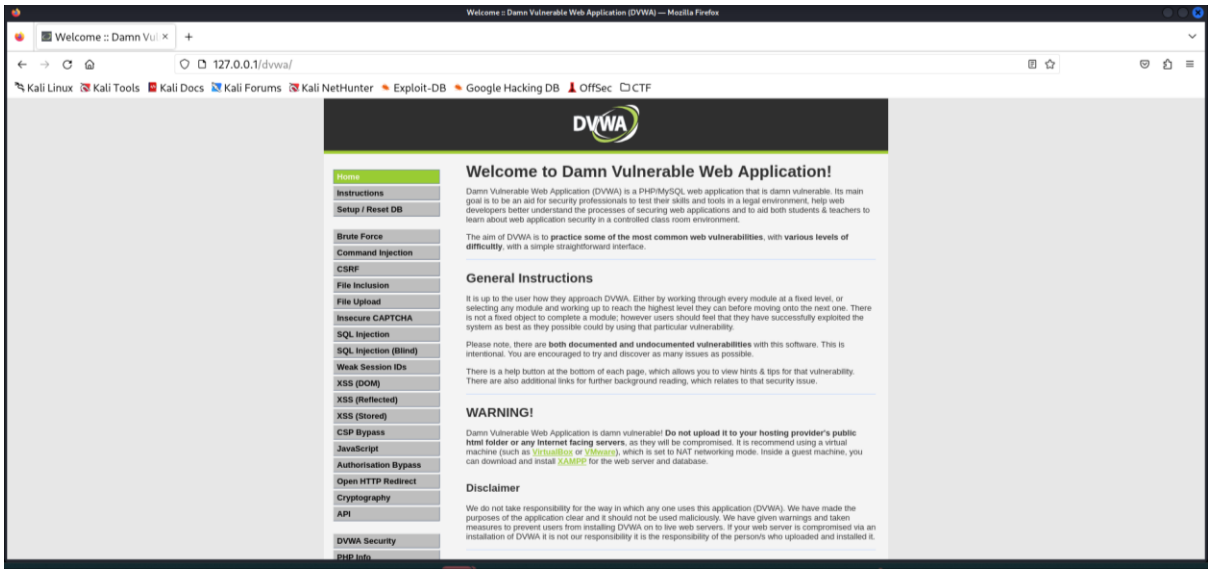


Figure 11: DVWA

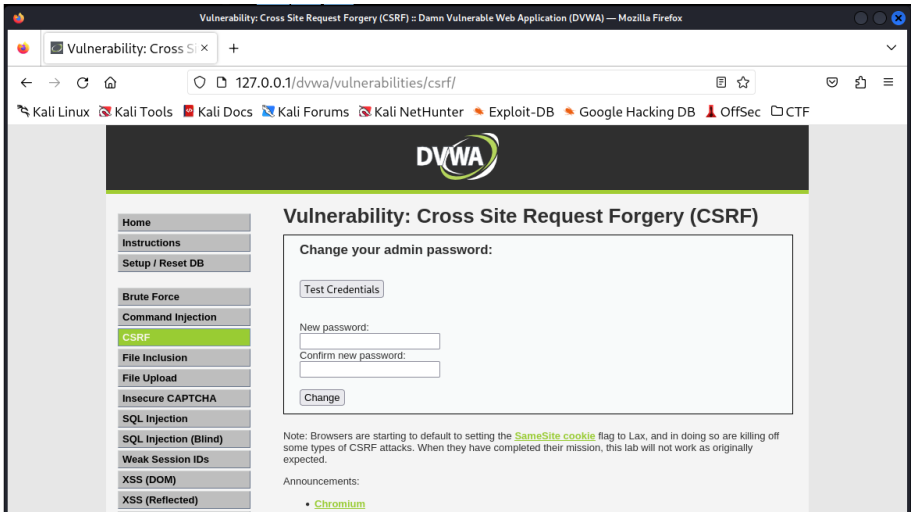


Figure 12: DVWA CSRF

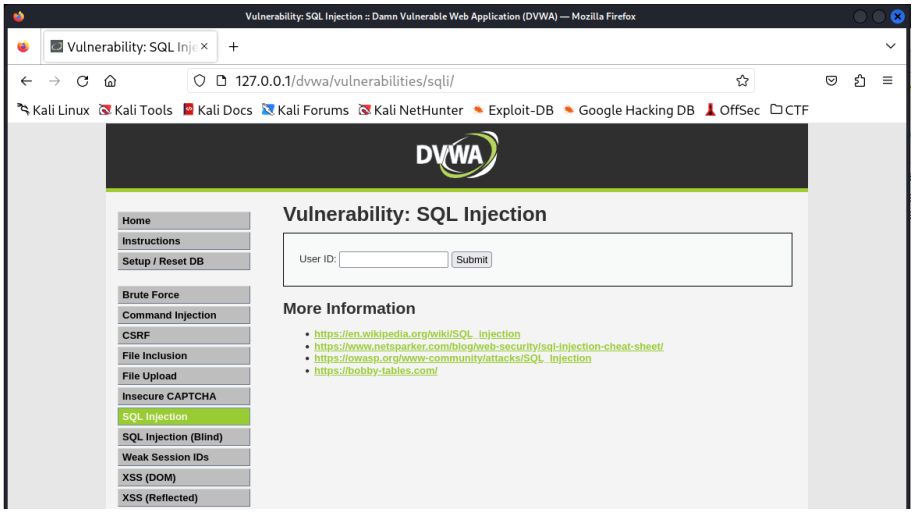


Figure 13: DVWA SQL Injection

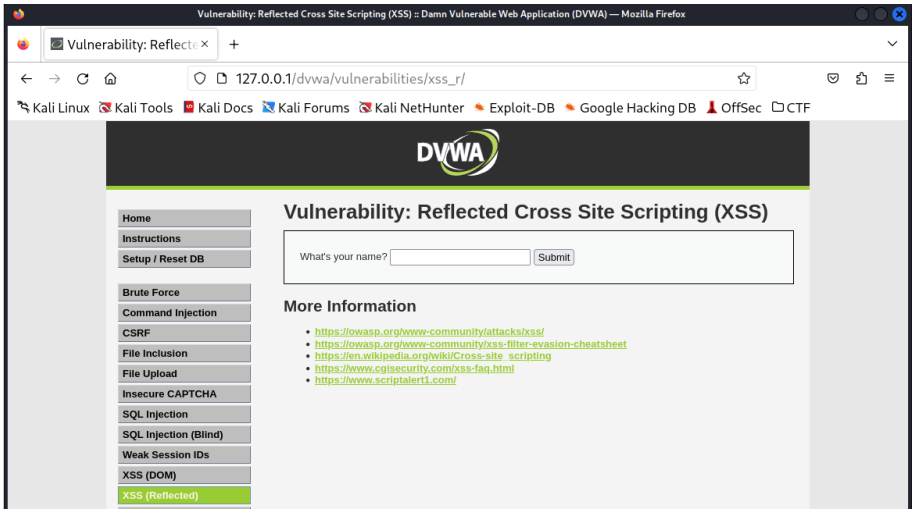


Figure 14: DVWA XSS (Reflected)

DVWA follows a three-tier architecture which is built using **HTML, CSS, JavaScript**, and **PHP-based forms**. By default, it is accessible via web browser with the URL **http://ipaddress/dvwa/**.

Figure 15: DVWA Directory

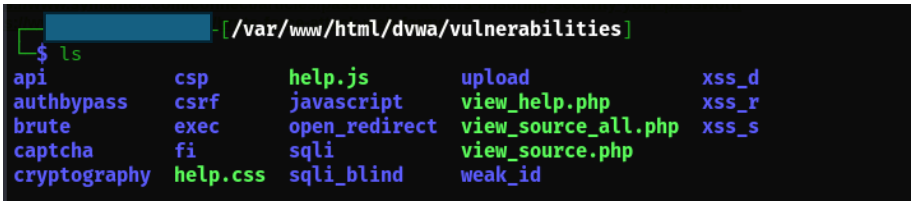


Figure 16: DVWA Vulnerabilities Directory

DVWA is implemented with PHP scripts that are used to handle user interactions, form submissions, and database queries. Based on the figure above, DVWA is designed with multiple individual PHP scripts that hold specific security flaws that allow users to practically test their skills for specific web application vulnerabilities.

Hence, instead of performing a broad scan on **http://ipaddress/dvwa** which includes login pages, dashboards, and non-vulnerable sections, the assessment should directly target specific vulnerable functionalities inside the **/dvwa/vulnerabilities/** directory. Each subdirectory in **/dvwa/vulnerabilities/** corresponds to a particular web security flaw that can be assessed individually. This ensures a more precise and efficient scanning that reduces noise from non-vulnerable paths by directly interacting with critical security flaws with a focus on assessing weaknesses within exploitable components.

2.3 Automated Scan

In OWASP ZAP, Automated Scan is a feature that automatically detects vulnerabilities in a web application by crawling and attacking known security weaknesses without requiring manual interaction. It utilizes techniques such as Active Scanning and Spidering to map the application structure and identify potential threats.

2.3.1 Automated Scan 1 (Broader Scan)

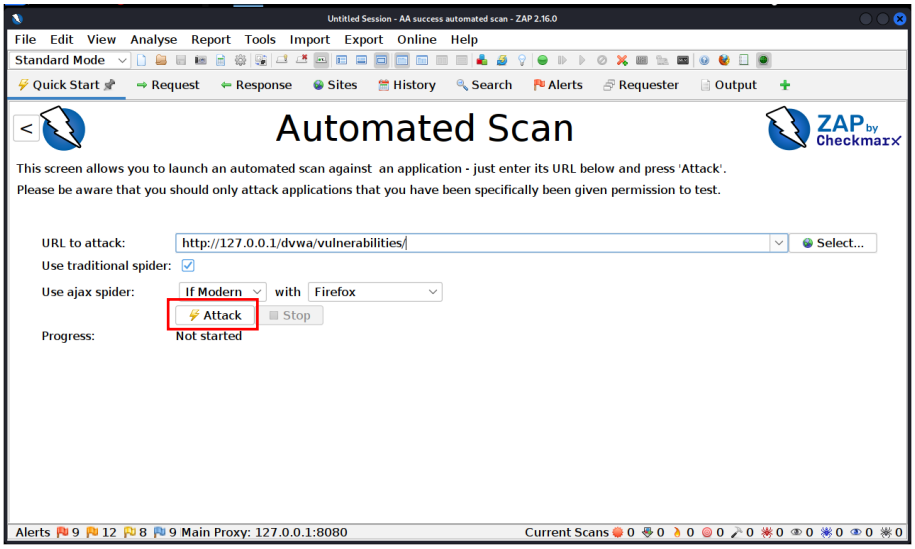


Figure 17: Automated Scan 1

The assessment starts with an automated scan on the target path with the URL **http://127.0.0.1/dvwa/vulnerabilities/** to focus on vulnerability scanning on the **/dvwa/vulnerabilities/** directory. This scanning assessment utilizes the **traditional Spider** tool to crawl over the web application, allowing a more comprehensive scanning to discover hidden URLs which include available links, forms and navigate paths to enumerate the entire website structure.

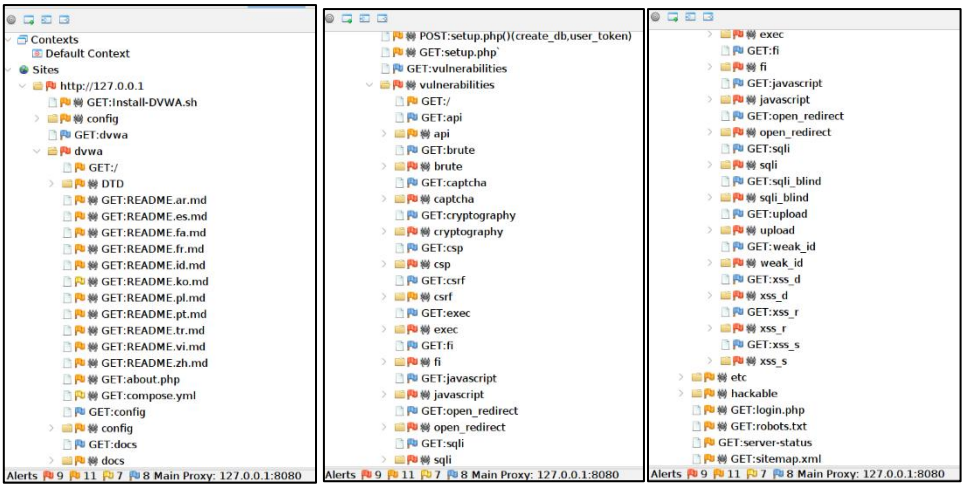


Figure 18: Automated Scan 1 – Sites Discovered

Those discovered paths and directories will be listed in the Site Panel. This includes each of the specific vulnerability directories such as **brute, api, sqli, and xss(s)**.

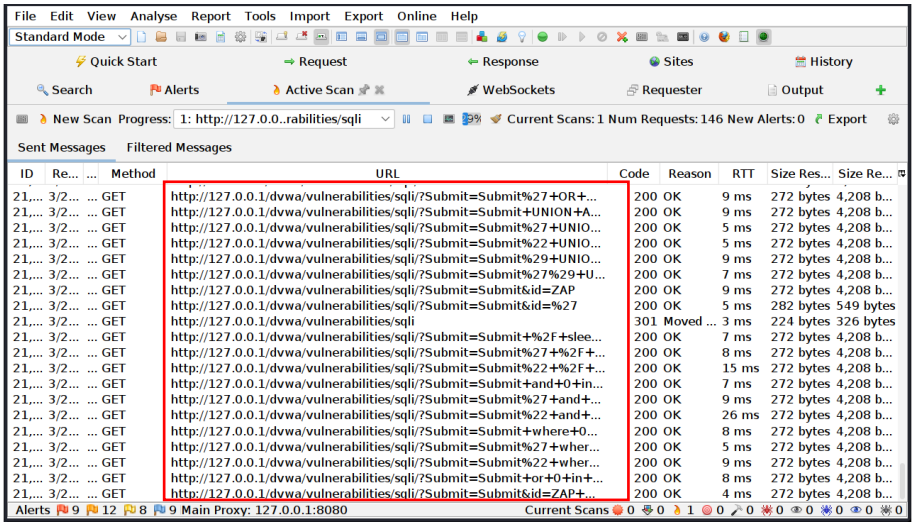


Figure 19: Automated Scan 1.1

During the scanning process, OWASP ZAP utilizes various techniques to actively send malicious requests to test various web application vulnerabilities. This includes **creating different attack payloads** such as SQL Injection, Cross-Site Scripting (XSS) and parameter tampering to modify HTTP requests and input malicious data to trigger vulnerabilities.

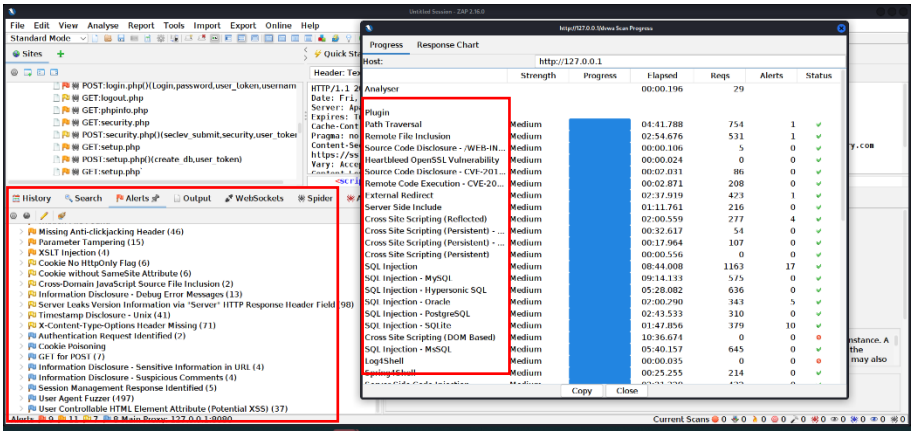


Figure 20: Automated Scan 1.2

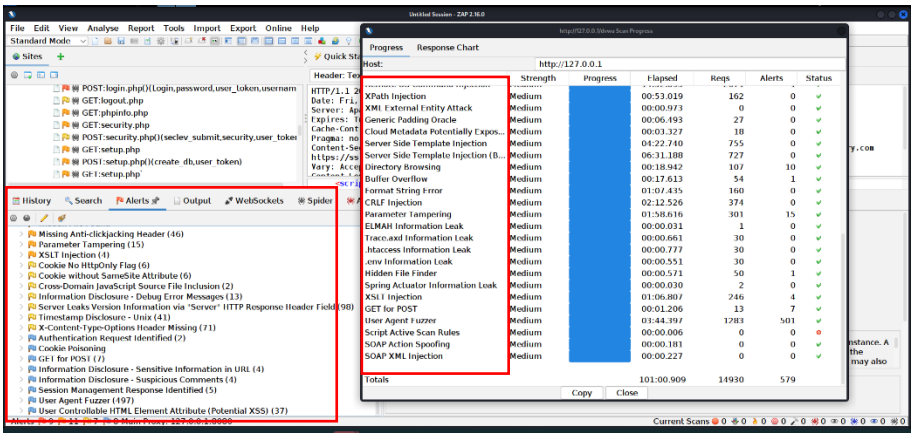


Figure 21: Automated Scan 1.3

Based on the Progress pop-up window, OWASP ZAP scans the entire application efficiently through preconfigured settings with a list of common vulnerabilities as shown in the figure above. This includes the scanning of Path Traversal, Source Code Disclosure, SQL Injection and so on. While in the Alert panel, those vulnerabilities that have been discovered throughout the scanning process will be listed in the Alerts panel with the severity level differentiated by coloured flags.

Automated Scanning Discovery (http://127.0.0.1/dvwa/vulnerabilities/)

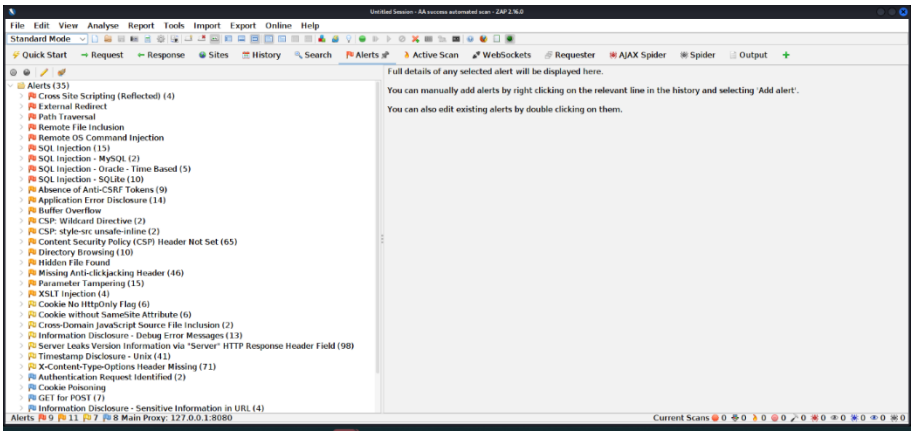


Figure 22: Automated Scan 1.4

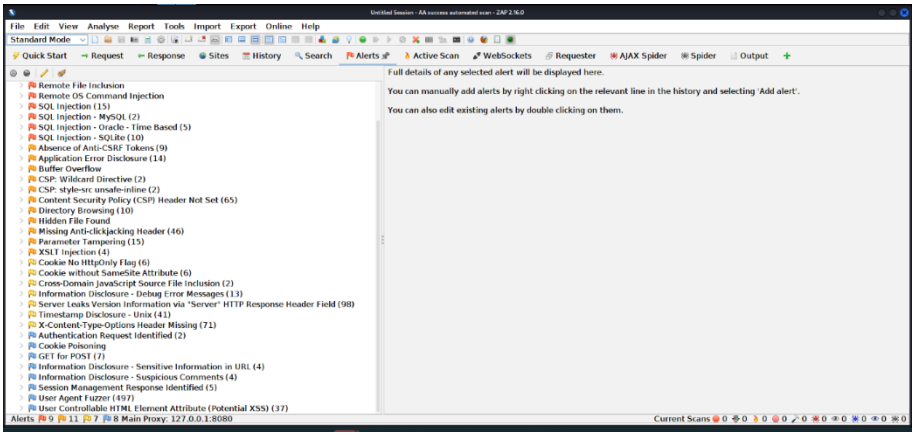


Figure 23: Automated Scan 1.5

Upon complete automated scanning, a total of 35 vulnerabilities are discovered and listed in the Alert panel. This includes **9 red-flagged vulnerabilities** that indicate a high severity level, **11 orange-flagged vulnerabilities** that serve as a medium severity level, **7 yellow-flagged vulnerabilities** that show a low severity level and **8 blue-flagged** with information severity level. Each of these vulnerabilities discovered are listed with the instances and paths where they have been explored.

Example Vulnerability Discovered: Cross-Site Scripting (Reflected)

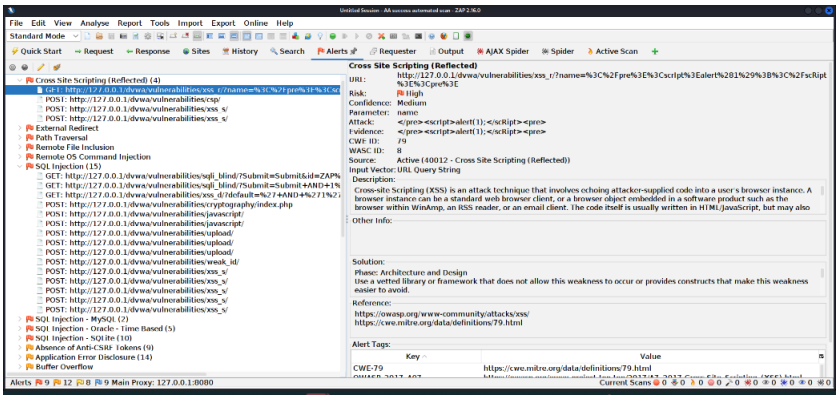


Figure 24: Automated Scan 1.6

For example, Reflected Cross-Site Scripting(XSS) has been discovered in the automated scanning. Based on the instance shown in figure above, the vulnerability is discovered in **http://127.0.0.1/dvwa/vulnerabilities/xss_r/?name=</pre><scrIpt>alert(1);</scRipt><pre>**, which is under Cross-Site Scripting (Reflected) directory with payload **</pre><scrIpt>alert(1);</scRipt><pre>** crafted and included in the **name** parameter to detect the vulnerability.

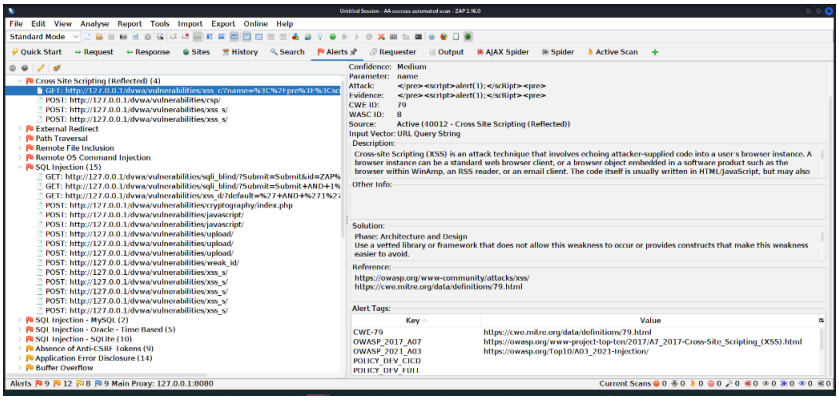


Figure 25: : Automated Scan 1.7

Besides, OWASP ZAP also provides detailed information on the vulnerability with descriptions, solutions and references with **CWE and OWASP Top 10** of the vulnerability. For example, the explanation of Cross-site Scripting (XSS) attack, the steps to mitigate this vulnerability, and **CWE-79 (Improper Neutralization of Input During Web Page Generation (Cross-site Scripting))** indicating the ID of XSS in the standardized Common Weakness Enumeration list, while XSS is listed in the OWASP Top 10 in both 2017 and 2021 as **Injection**.

2.3.2 Automated Scan 2 (Targeted Scan)

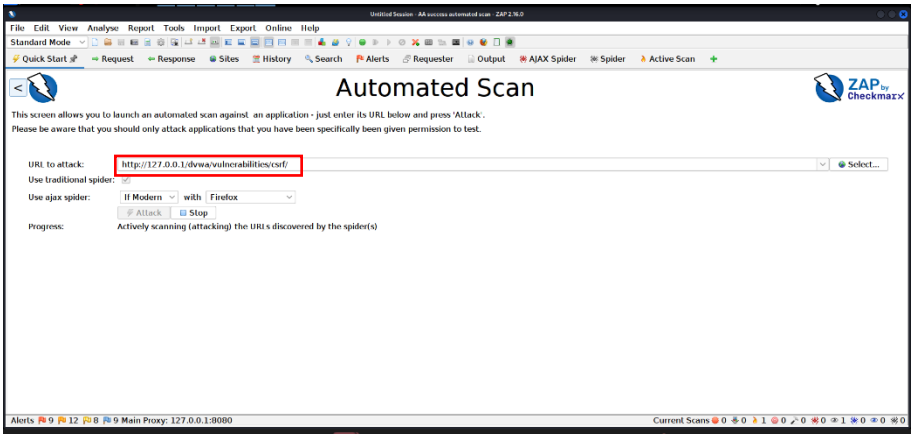


Figure 26: Automated Scan 2

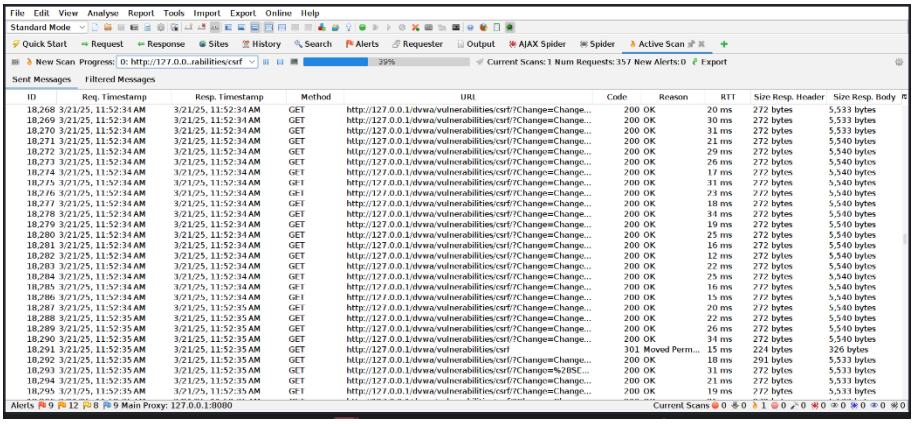


Figure 27: Automated Scan 2.1

After conducting an automated vulnerability scan across a broad range of attack surfaces focusses on **http://127.0.0.1/dvwa/vulnerabilities/**, a more targeted assessment is performed to analyse specific vulnerabilities within the targeted endpoint **http://127.0.0.1/dvwa/vulnerabilities/csrf/**.

Unlike the initial broad-range scan that provides a comprehensive overview of the application's security posture in identifying various potential weaknesses such as **SQL Injection (SQLi), Cross-Site Scripting (XSS), Broken Authentication, and Security Misconfigurations**, this targeted scan on specified endpoint ensures a detailed evaluation of targeted vulnerabilities, CSRF. The specialized testing on CSRF helps determine whether the vulnerability is actively exploitable, ensures a detailed analysis of the targeted vulnerability, and helps to uncover deep-seated security flaws that may not have been fully detected in the initial automated scan.

By combining automated scanning for broad discovery with specified target testing, this approach enhances the accuracy, efficiency, and reliability of the Vulnerability Assessment and Penetration Testing (VAPT) process. This methodology ensures that security weaknesses are thoroughly examined, allowing for effective remediation strategies to be developed.

Automated Scanning Discovery (http://127.0.0.1/dvwa/vulnerabilities/csrf/)

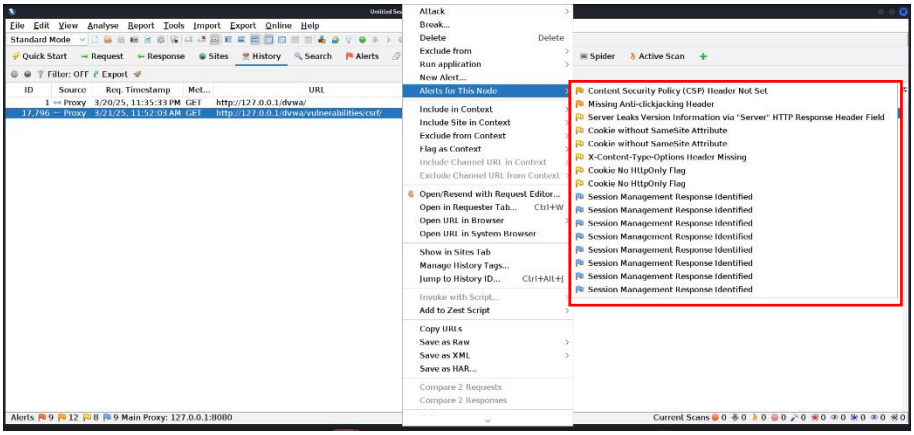


Figure 28: Automated Scan 2.3

Upon complete automated scanning on specified endpoints, a total of 16 vulnerabilities are discovered. This includes **2 orange flagged vulnerabilities** serving as medium severity level, **6 yellow flagged vulnerabilities** showing low severity level and **8 blue flagged** with information severity level. Each of these vulnerabilities discovered is listed with the instances and paths where they have been explored in the Alert Panel together with the result shown in initial scanning.

Example Vulnerability Discovered: Cross-Site Request Forgery (CSRF)

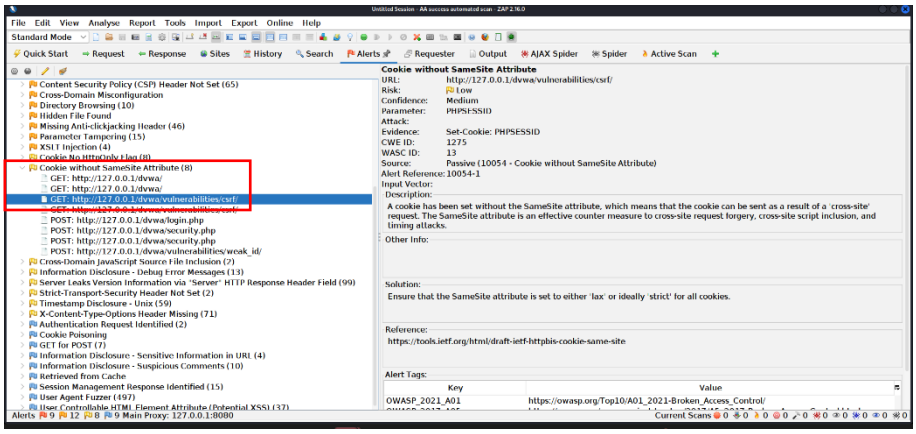
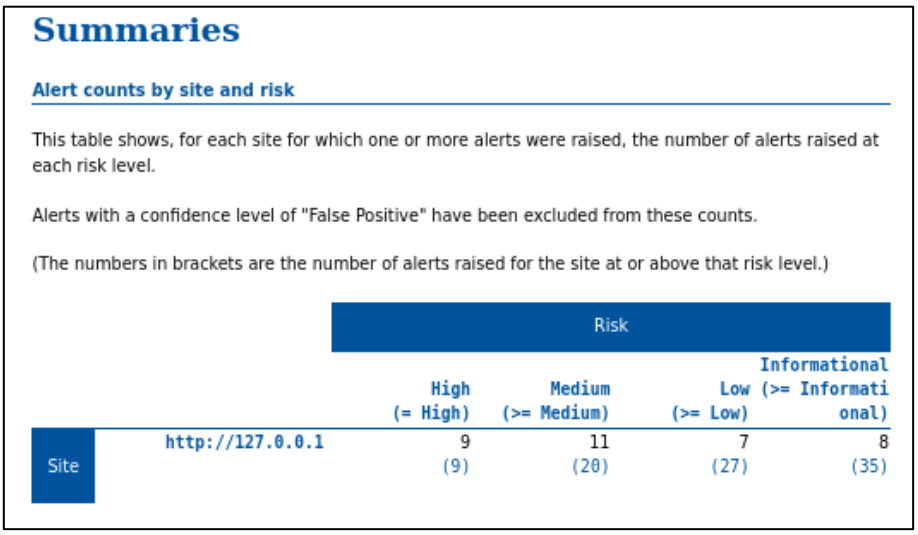


Figure 29: Automated Scan 2.4

According to the result shown in the Figure above, the alert stating the presence of **Cookie without SameSite Attributes** was detected within the CSRF directory and listed together with the vulnerabilities discovered in the initial scan. The presence of this alert signifies a **Cross-Site Request Forgery (CSRF) vulnerability** within the CSRF endpoint, confirming the accuracy of the assessment in alignment with the web application's intended security testing objectives. This finding highlights a potential security risk where an attacker could exploit the absence of the SameSite attribute to perform unauthorized actions on behalf of authenticated users.

According to the result shown, the vulnerability was discovered in **http://127.0.0.1/dvwa/vulnerabilities/csrf/**, with evidence in the request form indicating the presence of PHPSESSID instead of SameSite cookie in the Set-Cookie field. As a reference, the alert was referenced with **CWE ID of 1275 (Sensitive Cookie with Improper SameSite Attribute)** and listed in the OWASP_2021_A01 (**Broken Access Control**).

2.4 Overall Result and Vulnerabilities Discovered



Summaries

Alert counts by site and risk

This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.

Alerts with a confidence level of "False Positive" have been excluded from these counts.

(The numbers in brackets are the number of alerts raised for the site at or above that risk level.)

		Risk			
		High (= High)	Medium (>= Medium)	Low (>= Low)	Informational (>= Informational)
Site	http://127.0.0.1	9 (9)	11 (20)	7 (27)	8 (35)

Figure 30: Overall Result

OWASP Top Ten 2021		Alerts Count based on Risk Level				Total Count
		High	Medium	Low	Informational	
A01	Broken Access Control	1	2	3	2	8
A02	Cryptographic Failures	0	0	0	0	0
A03	Injection	8	2	0	2	12
A04	Insecure Design	0	1	0	0	1
A05	Security Misconfiguration	0	6	3	0	9
A06	Vulnerable and Outdated Components	0	0	0	0	0
A07	Identification and Authentication Failures	0	0	0	0	0
A08	Software and Data Integrity Failures	0	0	1	0	1
A09	Security Logging and Monitoring Failures	0	0	0	0	0
A10	Server-Side Request Forgery	0	0	0	0	0
	Unknown	0	0	0	4	4
		9	11	7	8	35

Figure 31: Overall Result 2

According to the result of alerts shown in the vulnerability assessment on the DVWA application using OWASP ZAP, the findings summarize the number of alerts categorized by risk level, highlighting the overall security posture of the application. The assessment identifies vulnerabilities based on severity, ranging from high-risk threats that pose immediate security concerns to lower-risk issues that may still contribute to potential exploitation. By analysing the distribution of these alerts, critical vulnerabilities such as **Injection (A03)**, **Security Misconfiguration (A05)**, and **Broken Access Control (A01)** based on the **OWASP Top Ten** list are prioritized. This analysis provides valuable insights into areas that require remediation,

ensuring a structured approach to addressing security flaws, mitigating risks, and strengthening the application's defenses against cyber threats.

Among all alerts discovered in the OWASP ZAP vulnerability assessment on DVWA, **Injection (A03)** from the OWASP Top Ten 2021 ranks as the most prevalent vulnerability, accounting for **12 out of the total 35 alerts**. This is followed by **Security Misconfiguration (A05)** with **9 alerts** and **Broken Access Control (A01)** with **8 alerts**, indicating significant security weaknesses in access control and system configurations. Additionally, **Insecure Design (A04) and Software and Data Integrity Failures (A08)** were identified, each with **1 alert**, suggesting lower but still relevant security concerns. This distribution of alerts emphasizes the critical areas that require remediation to enhance the security posture of the application.

A03:2021-Injection

Injection is a type of cybersecurity attack where unauthorized data is sent to a system or application, with the goal of maliciously exploiting vulnerabilities. This attack often takes advantage of insufficiently validated input mechanisms, to carry out unauthorized activities such as data theft, loss of data integrity, denial of service, and hijacking of users' sessions and privileges. (*A03 Injection - OWASP Top 10:2021*, 2021) The most common forms of injection attacks include **SQL Injection (SQLi)**, where attackers manipulate web application's database queries, **Command Injection**, where they trick weak applications into carrying out unapproved system commands, and **Cross-Site Scripting (XSS)** by inserting malicious scripts into web applications. (*OWASP Top 10 Injection Attacks Explained.*, 2023)

A03:2021-Injection				
No.	Alert	CWE ID	OWASP_2021	Risk Level
1	Cross Site Scripting (Reflected)	79	A03	High
2	External Redirect	-	A03	High
3	Remote File Inclusion	-	A03	High
4	Remote OS Command Injection	78	A03	High
5	SQL Injection	89	A03	High
6	SQL Injection - MySQL	89	A03	High
7	SQL Injection – Oracle-Time Based	89	A03	High
8	SQL Injection - SQLite	89	A03	High
9	Buffer Overflow	120	A03	Medium
10	XSLT Injection	91	A03	Medium
11	Cookie Poisoning	565	A03	Information
12	User Controllable HTML Element Attribute (Potential XSS)	20	A03	Information

Figure 32: OWASP Top Ten 2021 - Injection

Injection remains as one of the most critical security that is listed in OWASP Top 10 2021 as it enables attackers to submit untrusted input into an application, result in unintentional command execution, unauthorized access to data, or system compromise. In this assessment, 12 injection vulnerabilities have been found in the Damn Vulnerable Web Application (DVWA) through OWASP ZAP scanning. Among those Injection vulnerabilities, 10 alerts were mapped to Common Weakness Enumeration (CWE) ID, which indicates the identified vulnerability is widely recognized and documented as a severe security threat, emphasizing their severity and potential impact. This includes:

- **Cross-Site Scripting (Reflected)**

  CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting) – The injection of malicious scripts that can be executed within DVWA

can lead to the stealing of session cookies, manipulate of the web application and phishing attacks upon executing.

- **External Redirect –** The redirection of parameters without direct code injection. This can lead to manipulation in parameters to be redirected to malicious sites.
- **Remote File Inclusion –** The injection of malicious files into DVWA might lead to code execution on the server to steal sensitive information.
- **Remote OS Command Injection**

  CWE-78: Improper Neutralization of Special Elements Used in an OS Command (OS Command Injection) – The injection of system commands to execute malicious action to the system may potentially compromise the entire system.
- **SQL Injection(s)**

  CWE-89: Improper Neutralization of Special Elements Used in an SQL Command (SQL Injection - SQLi) – Exploit vulnerabilities through injection of queries in MySQL, Oracle, and SQLite databases, allowing unauthorized access, modify and extract of sensitive data through compromising databases.
- **Buffer Overflow**

  CWE-120: Buffer Copy Without Checking Size of Input (Buffer Overflow) – Injection attacks that perform input overflows a memory buffer, leading to memory corruption or code execution of the system.
- **XSLT Injection**

  CWE-91: XML Injection (XSLT Injection) – The injection of Extensible Stylesheet Language Transformations (XSLT) code to perform transformations or data exfiltration to extract sensitive information through XML data processing.
- **Cookie Poisoning**

  CWE-565: Reliance on Cookies Without Validation and Integrity Checking (Cookie Poisoning) – The injection of malicious values into cookies parameters such as user cookies without verification to perform unauthorized actions to steal valuable information.
- **User Controllable HTML Element Attribute (Potential XSS)**

  CWE-20: Improper Input Validation – The injection of payloads or scripts within HTML elements may lead to potential Cross-Site Scripting (XSS) vulnerabilities, increasing the risk of script execution.

Overall, successful injection attacks can result in unauthorized access that allows privileged escalation, data breaches that expose sensitive data, and system compromise that enable attackers to take control of the application as long as session hijacking to steal cookies and inject malicious scripts, posing significant risks to DVWA.

A05:2021-Security Misconfiguration

Security misconfiguration refers to a set of vulnerabilities that arise from incorrectly setting up hardware, network, software, or application components. It encompasses issues where **inadequate or missing application hardening**, **unnecessary features** or **default settings** have remained in place, **security settings** for applications, frameworks, and databases are **not set** to secure values, and **software and components are outdated**. Security misconfiguration can lead to various attack vectors, including but not limited to unauthorized access, data breaches, and system compromises. (*A05 Security Misconfiguration - OWASP Top 10:2021*, 2021)

A05:2021-Security Misconfiguration				
No.	Alert	CWE ID	OWASP_2021	Risk Level
1	Application Error Disclosure	200	A05	Medium
2	CSP: Wildcard Directive	693	A05	Medium
3	CSP: style-src unsafe-inline	693	A05	Medium
4	Content Security Policy (CSP) Header Not Set	693	A05	Medium
5	Hidden File Found	538	A05	Medium
6	Missing Anti-clickjacking Header	1021	A05	Medium
7	Cookie No HttpOnly Flag	1004	A05	Low
8	Server Leaks Version Information via "Server" HTTP Response Header Field	200	A05	Low
9	X-Content-Type-Options Header Missing	693	A05	Low

Figure 33: OWASP Top Ten 2021 - Security Misconfiguration

Security Misconfiguration remains one of the most critical securities that is listed as fifth in OWASP Top 10 2021 as it enables attackers to exploit incorrectly specified security parameters, obtain illegal access, reveal private data, or change system behaviour. In this assessment, 9 security misconfiguration vulnerabilities have been found in the Damn Vulnerable Web Application (DVWA) through OWASP ZAP scanning. All security misconfiguration vulnerabilities were mapped to Common Weakness Enumeration (CWE) ID, which indicates the identified vulnerability is widely recognized and documented as a severe security threat, emphasizing its severity and potential impact. This includes:

- **Application Error Disclosure**

  CWE-200: Exposure of Sensitive Information to an Unauthorized Actor – Application exposes error messages which may contain sensitive information of the application such as internal system details, potentially act as an entry point for attackers to exploit.

- **Content Security Policy (CSP)**

  CWE-693: Protection Mechanism Failure – CSP is not properly implemented within the application such as using wildcard directives (*), allowing inline styles or not implemented, which might lead to potential XSS or injection-based attacks.

- **Hidden File Found**

  CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory – The hidden file that could contain sensitive information is discovered which might expose information such as credentials or application configuration settings.

- **Missing Anti-clickjacking Header**

  CWE-1021: Improper Restriction of Rendered UI Layers or Frames – Missing X-Frame-Options or Content-Security-Policy: frame-ancestors headers that allow the application to be embedded with iframe by external vendors, potentially causing unintended critical actions through clickjacking attacks.

- **Cookie No HttpOnly Flag**

  CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag – The HttpOnly flag was not implemented, which might lead to unauthorized access, stealing or manipulation of session cookies through client-side scripts.

- **Server Leaks Version Information via "Server" HTTP Response Header Field**

  CWE-200: Exposure of Sensitive Information to an Unauthorized Actor – Revealing version information through a server in the response header, potentially allows attackers to identify outdated security flaws to attack the application.

- **X-Content-Type-Options Header Missing**

  CWE-693: Protection Mechanism Failure – The absence of an X-Content-Type-Options Header can lead to exploitation by bypassing security measures to perform critical security attacks.

Overall, the **Security Misconfiguration (A05:2021) vulnerabilities** identified as second critical weaknesses in the **DVWA application**, that could be exploited by attackers to compromise application security, steal sensitive information, or execute unauthorized actions. These findings emphasize the importance of **proper security configurations**, **implementing security headers**, and **restricting access to sensitive files**. Addressing these issues will significantly enhance the overall security posture of the DVWA, reducing potential attack surfaces and mitigating risks effectively.

A01:2021-Broken Access Control

Broken Access Control refers to a security vulnerability where an application fails to enforce access control policies correctly, which allows attackers to access resources without authorization or carry out illegal activities, like altering data, accessing private information, or interfering with an application. This might cause by **improper authorization** when accessing the application and **insecure direct object reference** that reveals a resource's direct reference in the URL within the application. (*OWASP Top 10 Broken Access Control Explained*., 2023) This can lead to unauthorized data exposure, data manipulation, and business disruptions because users are able to act outside of their intended permissions. (*A01 Broken Access Control - OWASP Top 10:2021*, 2021)

A01:2021-Broken Access Control				
No.	Alert	CWE ID	OWASP_2021	Risk Level
1	Path Traversal	22	A01	High
2	Absence of Anti-CSRF Tokens	352	A01	Medium
3	Directory Browsing	548	A01	Medium
4	Cookie without SameSite Attribute	1275	A01	Low
5	Information Disclosure - Debug Error Messages	200	A01	Low
6	Timestamp Disclosure - Unix	200	A01	Low
7	Information Disclosure - Sensitive Information in URL	200	A01	Information
8	Information Disclosure - Suspicious Comments	200	A01	Information

Figure 34: OWASP Top Ten 2021 - Broken Access Control

Broken Access Control remains as one of the most critical security which is listed as first in OWASP Top 10 2021 as it enables attackers to exploit incorrectly enforced authentication and restriction, gain unauthorized access to sensitive data, modify permissions, or execute privileged functions. In this assessment, 8 Broken Access Control vulnerabilities have been found in the Damn Vulnerable Web Application (DVWA) through OWASP ZAP scanning. All Broken Access Control vulnerabilities were mapped to Common Weakness Enumeration (CWE) ID, which indicates the identified vulnerability is widely recognized and documented as a severe security threat, emphasizing its severity and potential impact. This includes:

- **Path Traversal**

  CWE-22: Improper Limitation of a Pathname to a Restricted Directory (Path Traversal) - Allow access to restricted directories and files stored outside the intended web root folder, which may lead to exposure of sensitive information such as credentials, application source code, system logs or configuration files in a hidden directory.

- **Absence of Anti-CSRF Tokens**

  CWE-352: Cross-Site Request Forgery (CSRF) – The lack of Cross-Site Request Forgery (CSRF) protection enables attackers to trick authenticated users into unintended action on privilege escalation, unauthorized account modifications as well as forced transactions leading to financial or data loss.

- **Directory Browsing**

  CWE-548: Exposure of Information Through Directory Listing – Able to access critical directories through browsing and navigation which might expose sensitive information such as sensitive files, source code, or system configurations.

- **Cookie without SameSite Attribute**

  CWE-1275: Sensitive Cookie with Improper SameSite Attribute - Missing SameSite attributes in cookies increases the risk of CSRF attacks by allowing third-party websites to initiate unauthorized requests on behalf of a user, leading to unauthorized session hijacking or privileges escalation to steal sensitive information.

- **Information Disclosure**

  CWE-200: Exposure of Sensitive Information to an Unauthorized Actor – Revealing sensitive information such as detailed error messages, Unix timestamps, sensitive information in URLs, and suspicious developer or debugging comments in source code. This might help an attacker discover potential security weaknesses through information given to exploit the application and perform malicious actions.

Overall, the **Broken Access Control** vulnerabilities discovered in DVWA reveal multiple serious weaknesses that allow unauthorized users to access and modify sensitive data and system functionalities. The path traversal and directory browsing expose critical system files which include sensitive information, while **CSRF protections** and **SameSite cookie attributes** leave the application vulnerable to session hijacking and unintended actions. Various information disclosure issues serve as entry points that reveal application information which include the weaknesses of the application to attackers to exploit and cause further attacks. These discovered vulnerabilities and weak access control risks require robust mitigation strategies implementation which include **proper access control enforcement, secure cookie management, and implementation of security headers and input validation**. Strengthening authentication mechanisms and reducing unnecessary information exposure can significantly enhance the security posture of the application.

A04:2021-Insecure Design

Insecure design refers to flaws and weaknesses in the architecture and overall blueprint of a system that leads to vulnerabilities, which can be exploited by attackers. Unlike implementation flaws that arise from coding mistakes, insecure design issues stem from poor security considerations during the planning, development, and design phases of a system. It encompasses missing or ineffective security controls that are not adequately integrated into the software development lifecycle. This includes weaknesses such as improper input validation, improper secure communication between client and server, and improper session management handling. (*OWASP Top 10 Insecure Design Explained.*, 2023) These vulnerabilities often lead to exploitable weaknesses that attackers can leverage to compromise the confidentiality, integrity, or availability of an application. (*A04 Insecure Design - OWASP Top 10:2021*, 2021)

A04:2021-Insecure Design				
No.	Alert	CWE ID	OWASP_2021	Risk Level
1	Parameter Tampering	472	A04	Medium

Figure 35: OWASP Top Ten 2021 - Insecure Design

Insecure Design is listed as one of the **OWASP Top Ten** 2021 vulnerabilities, emphasizing weaknesses in application architecture and security controls rather than coding errors or misconfigurations. Unlike other vulnerabilities that arise from implementation flaws, insecure design occurs due to poor security considerations during the planning and design phases of software development. According to the result of this assessment, one Insecure Design vulnerability has been found within the Parameter Tampering Alert in the Damn Vulnerable Web Application (DVWA) through OWASP ZAP scanning. This alert is also referenced with a CWE ID of CWE-472.

- **Parameter Tampering**

  CWE-472: External Control of Assumed-Immutable Web Parameter – Allow manipulation of exchange parameter between client and server to modify parameter such as query parameter, form fields, cookies and headers to gain unauthorized access or bypass security mechanisms.

Overall, the Insecure Design vulnerabilities found on the DVWA indicate serious security defects in the application's architecture and consequently leave it at risk of several different kinds of attacks. The attackers may also manipulate some of the vital functionalities without proper access control mechanisms and no safe coding techniques in place. Also, the validation

procedures are insufficient, leading to extra danger of exploitation and permitting them to sneak beyond authentication and carry out destructive actions. Moreover, the application lacks security design principles like appropriate Threat Modelling and Risk Assumption, which include application is also vulnerable to popular attack vectors like injection attacks, authentication bypass, and privilege escalation. In order to address these design flaws, strong access control procedures, utilizing safe development frameworks, and incorporating security best practices into the software development lifecycle (SDLC) should be implemented. By focusing on security at the design level, organizations can drastically reduce vulnerabilities and improve the overall security posture of their web applications.

A08:2021-Software and Data Integrity Failures

Software and Data Integrity Failures refer to vulnerabilities and incidents where the integrity of software and data is compromised. This includes failures when applications do not properly validate the authenticity or integrity of software dependencies, updates, or transmitted data, leaving them susceptible to unauthorized modifications or malicious tampering. This can happen due to various reasons such as the use of untrusted or modified software components, lack of integrity checks, insecure deserialization, improper digital signature verification and lack of secure update mechanisms. (A08 Software and Data Integrity Failures - OWASP Top 10:2021, 2021) In this case, malicious code can be injected by the attackers to change application behaviour and gain unauthorized access to sensitive data. This can lead to accessing other users' data or it can even compromise the whole system. These vulnerabilities pose a serious risk to organizations, as they undermine the trustworthiness and security of applications, potentially affecting the confidentiality, integrity, and availability of critical systems.

A08:2021-Software and Data Integrity Failures				
No.	Alert	CWE ID	OWASP_2021	Risk Level
1	Cross-Domain JavaScript Source File Inclusion	829	A08	Low

Figure 36: OWASP Top Ten 2021 - Software and Data Integrity Failures

Software and Data Integrity Failures remains as one of the OWASP Top Ten 2021 vulnerabilities, highlighting the risks associated with insufficient mechanisms to ensure the integrity of software updates, critical data, and CI/CD pipelines. Unlike traditional security vulnerabilities that stem from coding errors or misconfigurations, Software and Data Integrity Failures occur due to the lack of proper validation, cryptographic controls, and verification mechanisms when handling sensitive data, software dependencies, or updates. According to the result shown in this vulnerability assessment on DVWA using OWASP ZAP, Cross-Domain JavaScript Source File Inclusion alert has been found referenced to Software and Data Integrity Failures with a CWE ID of CWE-829 indicating the **Inclusion of Functionality from Untrusted Control Sphere**.

- **Cross-Domain JavaScript Source File Inclusion**

  CWE-829: Inclusion of Functionality from Untrusted Control Sphere – Loading of JavaScript files from an untrusted domain without proper validation. This might cause the inject malicious code or manipulate application behaviour, leading to security threats.

Overall, DVWA's Software and Data Integrity Failures expose the lack of controls to ensure the integrity of data processing, dependency management, and software updates. In particular, the program is prone to supply chain attacks, malicious code injection and unauthorized modification and this compromises the supply chain integrity checks and validation of software components. These flaws can be used by attackers to modify program behaviours, compromise sensitive information, or create backdoors that stay open over time. In addition, the lack of cryptographic integrity checks, secure update procedures, and digital signatures increases the chances of code execution of unauthorized code in the application environment. In order to guarantee the authenticity and reliability of all software components, and put strict integrity verification procedures into place, organizations can mitigate these risks by forming a secure software supply chain management basis and incorporating cryptographic controls. This includes improving security resilience and preventing exploitation of their applications by giving software and data integrity top priority throughout the development process.

3.0 Penetration Testing

In this penetration testing section, Burp Suite and SQLMap will be utilized to exploit Cross-Site Request Forgery (CSRF) and SQL Injection (SQLi) vulnerabilities within the target web application. These attacks are among the most critical security risks affecting modern web applications.

CSRF attacks occur when an attacker tricks an authenticated user into unknowingly executing unauthorized actions on a web application. To mitigate these risks, many web applications implement CSRF protection mechanisms such as referer-based validation and anti-CSRF tokens. This assessment will evaluate the effectiveness of these security measures by analysing how the application processes requests and whether it properly enforces CSRF protection. Using Burp Suite, we will intercept and modify HTTP requests to determine if referer-based validation can be bypassed and whether the application correctly validates CSRF tokens before processing sensitive actions, such as changing user credentials.

Additionally, SQL Injection (SQLi) will be tested using SQLMap, an automated SQL injection tool that identifies and exploits database vulnerabilities. This assessment will focus on determining whether user input fields, such as login forms and search bars, are vulnerable to SQLi attacks. SQLMap will be used to analyse database interactions, extract sensitive information such as user credentials, and assess privilege escalation possibilities within the database.

By leveraging Burp Suite and SQLMap, this penetration test aims to evaluate the resilience of the target application against CSRF and SQL Injection attacks, identify security weaknesses, and provide actionable insights for strengthening the overall security posture of the system.

3.1 Burp Suite and SQLMap Usage

3.1.1 Burp Repeater & Burp Proxy

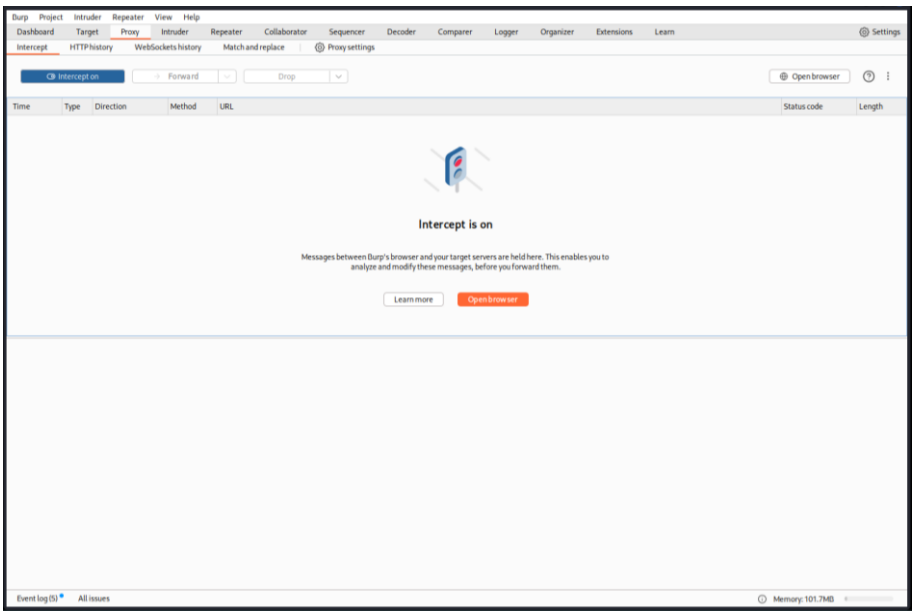


Figure 37: Burp Suite

Burp Suite provides essential features for detecting Cross-Site Request Forgery (CSRF) vulnerabilities in web applications. The **Proxy** tool plays a crucial role in this process by intercepting and capturing HTTP requests between the browser and the target application. This allows penetration testers to analyse how authentication mechanisms handle user requests, including whether the application enforces proper CSRF protection measures such as anti-CSRF tokens or referer-based validation. If an application fails to include these protections, it becomes susceptible to CSRF attacks, where an attacker can trick a logged-in user into performing unintended actions. (*Burp Suite Tutorial (Part 1): Introduction to the Burp Suite Proxy | Cybrary*, 2021)

The **Repeater** tool further aids in CSRF testing by allowing testers to manually modify and resend HTTP requests. By removing CSRF tokens from requests or modifying referer headers, testers can assess whether the application properly validates these security mechanisms. If a sensitive action, such as changing a password or transferring funds, can still be executed without these protections, it indicates a critical vulnerability that needs to be addressed. (*Part 3: Using Burp Suite Repeater More Efficiently | Cybrary*, 2021)

3.1.2 Automated SQL Injection Exploitation

Figure 38: SQLMap

SQLMap is a powerful automated tool used for identifying and exploiting SQL injection vulnerabilities in web applications. One of its key capabilities is **automated SQL injection testing**, where it systematically analyses input fields to determine if they are vulnerable to SQL injection attacks. By injecting malicious SQL payloads, SQLMap can evaluate how the application interacts with the underlying database and whether it exposes sensitive information.

Another critical feature of SQLMap is **database fingerprinting**, which helps identify the type and version of the database management system (DBMS) in use. This information allows testers to tailor their attacks based on the specific database vulnerabilities. Additionally, SQLMap supports **data extraction**, enabling penetration testers to retrieve critical database contents such as user credentials, stored data, and system configurations if a vulnerability is successfully exploited. The tool also includes **privilege escalation** techniques, which assess whether an attacker can elevate their access to higher database privileges, potentially leading to complete system compromise. (*SQLMAP | Bugcrowd*, 2022)

By leveraging **Burp Suite Proxy and Repeater** for **CSRF detection** and **SQLMap** for **SQL Injection exploitation**, penetration testers can systematically uncover and analyse critical security weaknesses in web applications. These tools help organizations strengthen their security posture by identifying vulnerabilities before malicious actors can exploit them.

3.2 CSRF (Without CSRF Prevention Measures)

1. Understanding Source Code

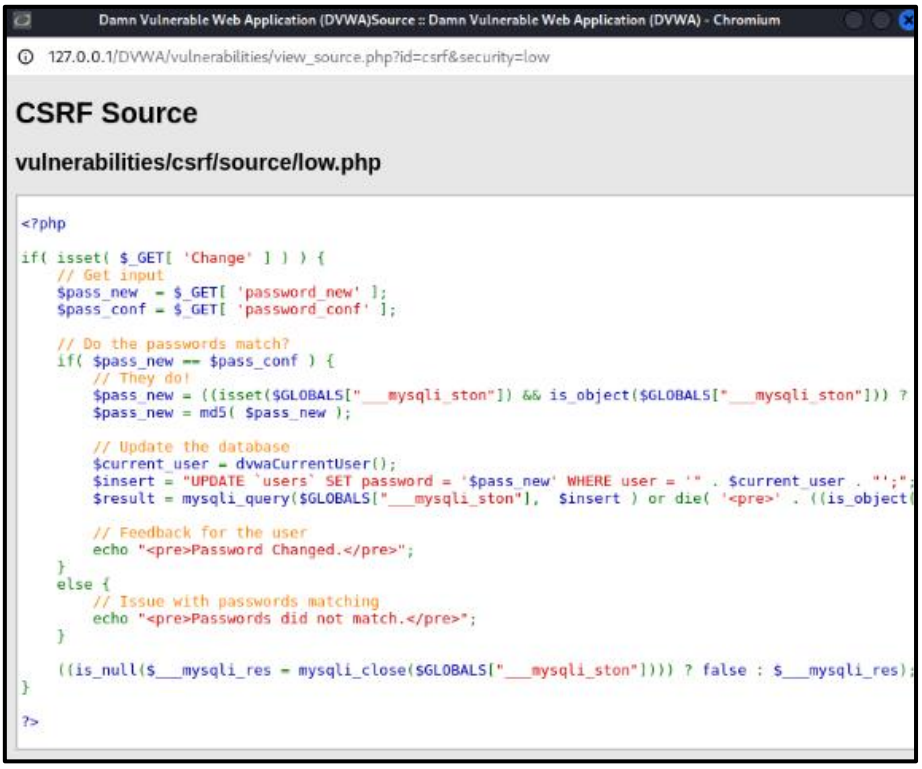


Figure 39: Exploit CSRF 1 – Source Code

An understanding of the web application architecture is essential before starting the actual penetration testing process. This includes the viewing or source code such as investigating the PHP code of this section. Based on the PHP code given, this section primarily functions to change the password of an authenticated user account. This form allows the checking of passwords which seems implemented without any CSRF prevention components. This form allows the checking of matching new passwords and confirmed passwords without any checking on user validation and CSRF attack prevention mechanisms implemented such as generating a CSRF token and including a referrer header in the request.

Hence, penetration testing can be carried out with a focus on exploiting CSRF attacks through changing passwords through sharing links with the specified new password and confirmed password parameters.

2. Test on Original Credential (admin)

Credentials	
Username	admin
Password	admin

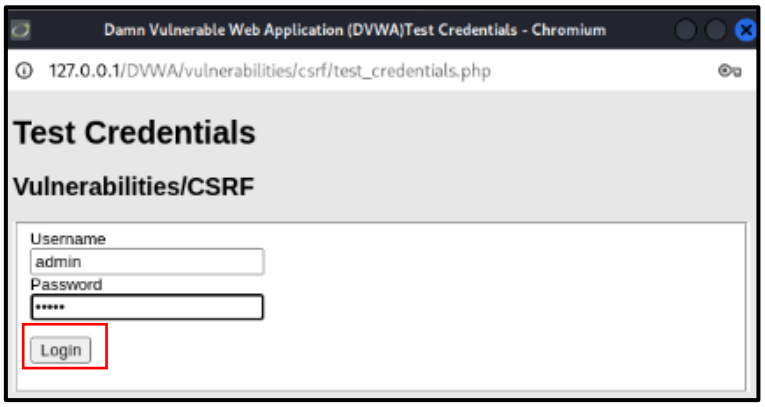


Figure 40 : Exploit CSRF 1.1 – Test Credential

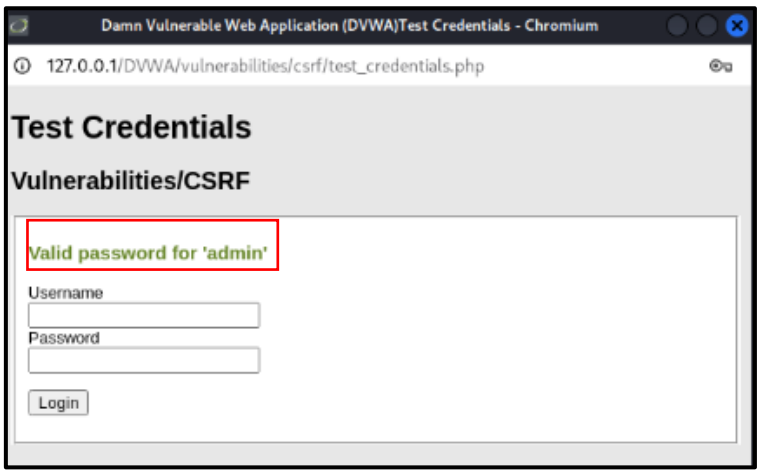


Figure 41 : Exploit CSRF 1.2 – Test Credential

The figure above shows proof of the **admin** account if primarily set with a password **admin**.

3. Change Password to password

Credentials	
Username	admin
Updated Password	password

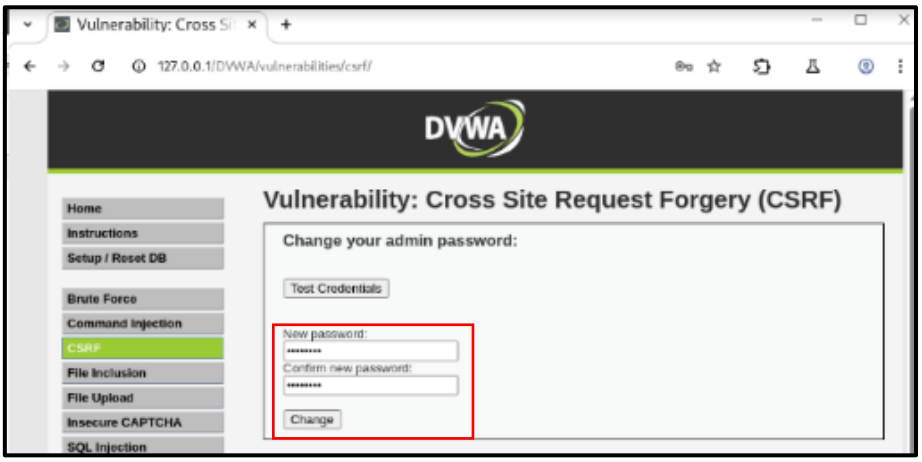


Figure 42 : Exploit CSRF 1.3 – Change Password

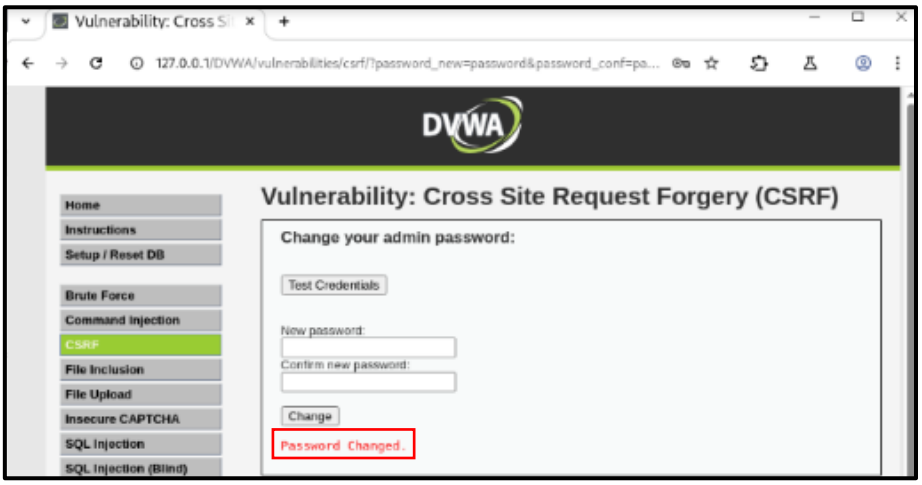


Figure 43: Exploit CSRF 1.4 – Success Change Password

Next, test the password-changing function by changing the original password to the password. The password has been successfully changed and updated.

4. Change Password on New Window

Credentials	
Username	admin
Updated Password	password 12

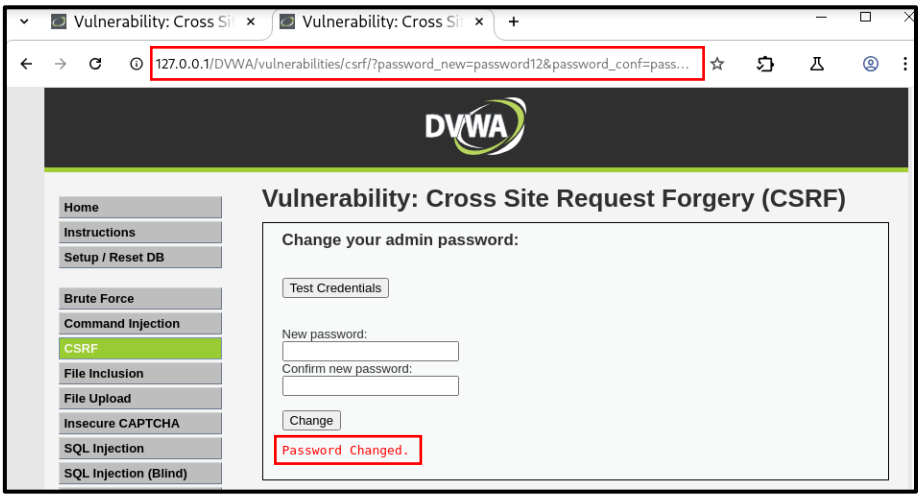


Figure 44: Exploit CSRF 1.5 – Password Changed

Testing on changing the password by attempting the same request but a different parameter sent on another browser. This includes copying and pasting the URL of the changing password request to a new window and attempting to act as another user to change the password by using the same request.

Original URL	http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change
Modified URL	http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=password12&password_conf=password12&Change=Change

This testing utilizes **password12** as the new password that is altered in the URL sent on the new window. Upon launching, the status with the "**Password Changed**" message is shown. This indicates the new password has been successfully changed and updated.

5. Modify Request Parameter to Change Password

Credentials	
Username	admin
Updated Password	password 123

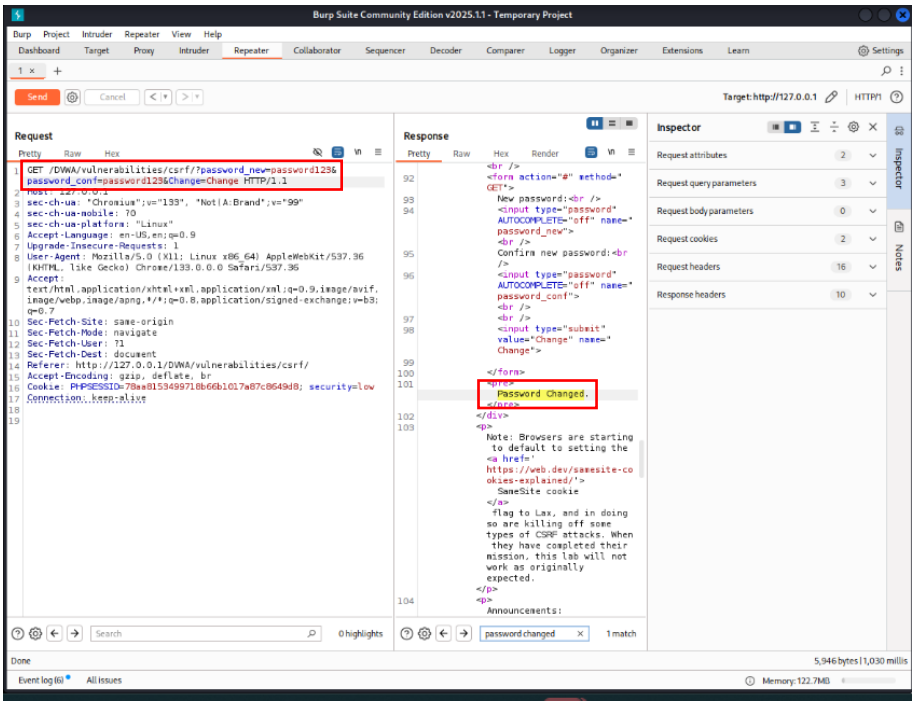


Figure 45 : Exploit CSRF 1.6 – Request Packet

This also can be tested using Burp Suite to further exploit the vulnerability by testing the changing of parameters in requests. This can be carried out by changing the parameter in the previous successful password-changing request, opening the same request in the Burp Repeater and modifying the parameter as per below.

Original API (parameter field)	/DVWA/vulnerabilities/csrf/?password_new=password12&password_conf=password12&Change=Change
Modified API (parameter field)	/DVWA/vulnerabilities/csrf/?password_new=password123&password_conf=password123&Change=Change

Based on the response shown, the message indicates the successful changing of the password using the same request but a modified parameter.

6. Password Changed Successfully

Testing Credentials	
Username	admin
Password	password 123

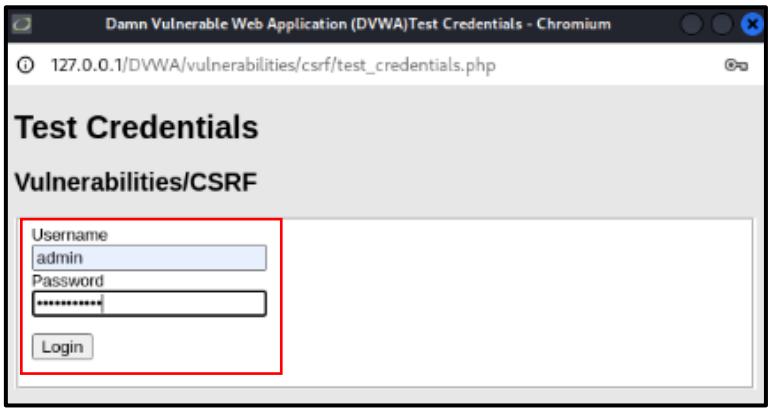


Figure 46 : Exploit CSRF 1.7 – Test Credentials

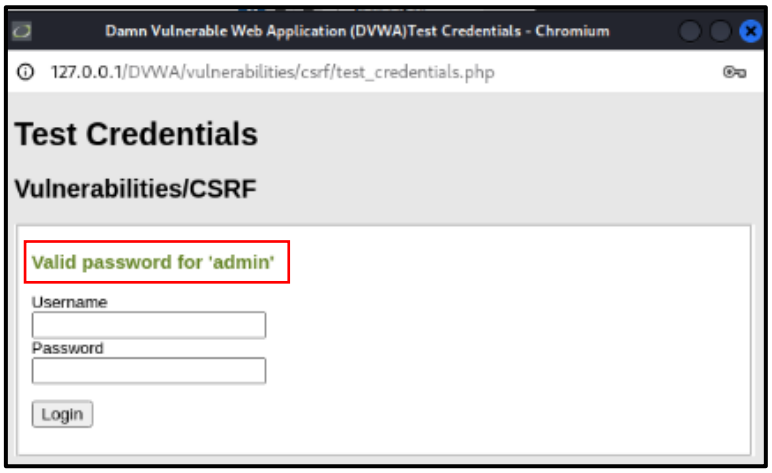


Figure 47 : Exploit CSRF 1.8 – Test Credentials

The updated credentials can be tested in the test credentials feature. The figure above shows the successful login using updated credentials.

7. Additional Exploit (Referrer Header Validation)

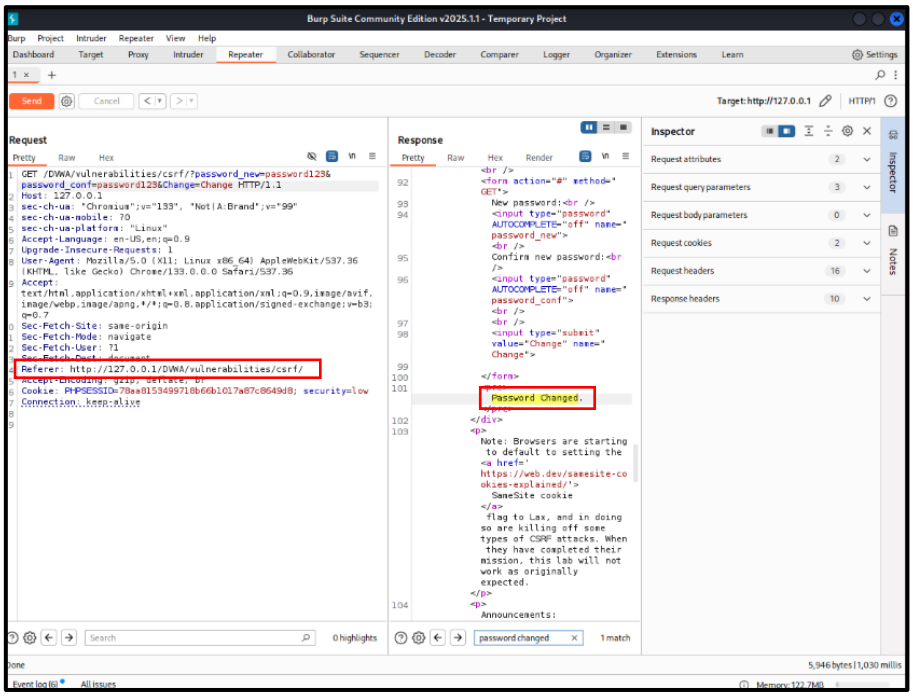


Figure 48 : Exploit CSRF 1.9 – Password Changed

The presence of the referrer header in the previous successful password change suggests that CSRF attack prevention measures may be in place. To test the effectiveness of these protections, the referrer header can be removed or modified to originate from a different source, allowing for an assessment of the CSRF validation mechanism on this endpoint.

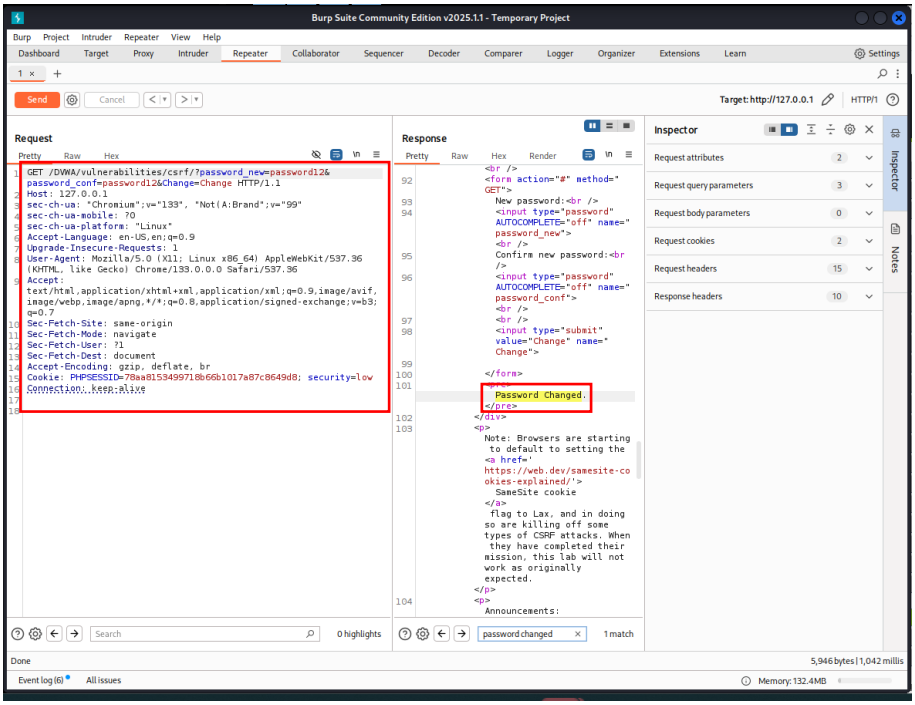


Figure 49 : Exploit CSRF 1.10 – Additional Exploit

Original API (parameter field)	/DVWA/vulnerabilities/csrf/?password_new=password123&password_conf=password123&Change=Change
Modified API (parameter field)	/DVWA/vulnerabilities/csrf/?password_new=password12&password_conf=password12&Change=Change

This attempt changed the password from **password123** to **password12** by modifying the parameter in the HTTP request and removing the referrer header **Referer: http://127.0.0.1/DVWA/vulnerabilities/csrf/**. When the referrer header is removed, the password change message appears, indicating that the request was successfully processed. This confirms that the page does not enforce referrer-based CSRF protection, leaving it vulnerable to such attacks.

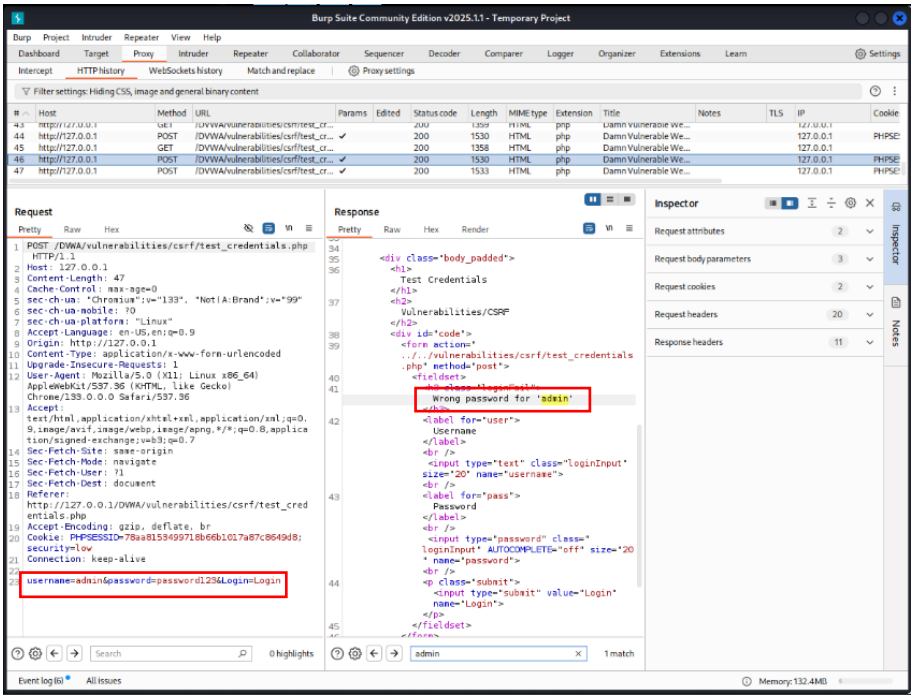


Figure 50 : Exploit CSRF 1.11 - Additional Exploit

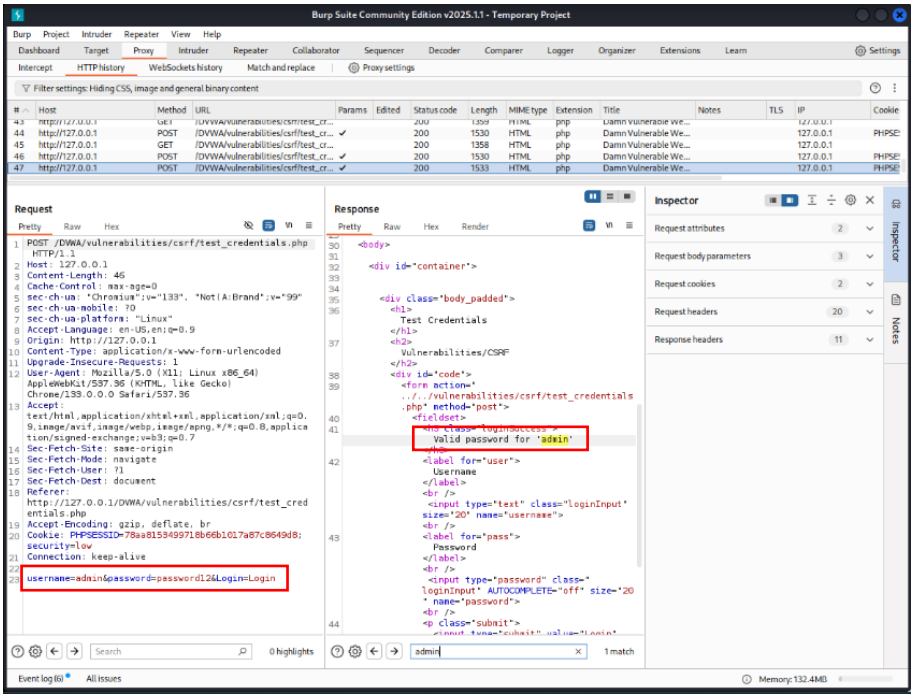


Figure 51 : Exploit CSRF 1.12 - Additional Exploit

The above PoCs show the request for test credentials attempts. The result shows the wrong password for admin when entering **password123** password while the valid password for admin when entering **password12**. This indicates the password has successfully been updated to **password123** for admin.

3.3 CSRF (Referer-Based Validation)

1. Understand Source Code

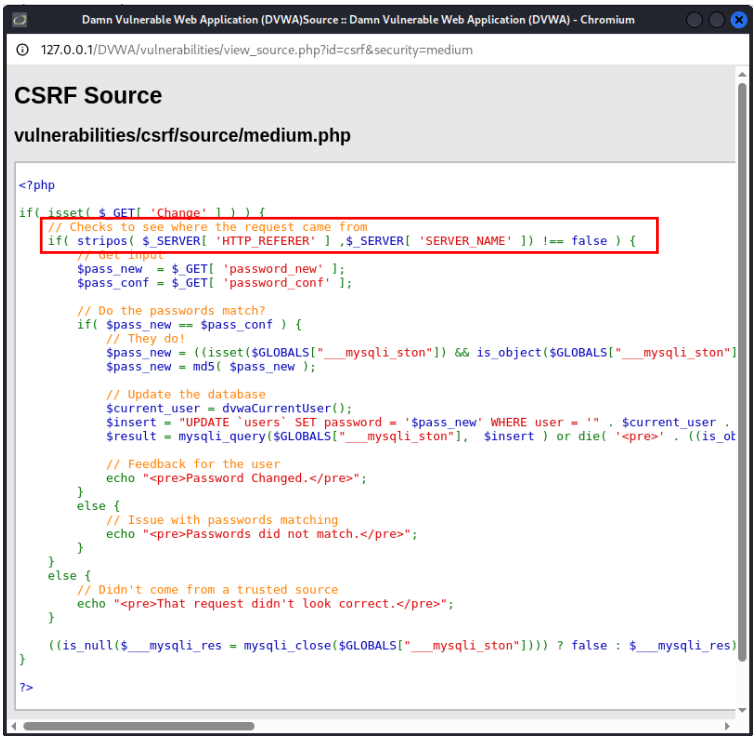


Figure 52 : Exploit CSRF 2 – Source Code

Before attempting the attack, analysing the PHP source code is essential to understand how the application processes CSRF requests. The code implementation shows that the application uses referrer-based validation as a security measure to prevent CSRF attacks. This means the server checks the **Referer** header in incoming requests to verify their legitimacy. Hence, the penetration testing in this section focuses on exploiting the CSRF attack with the misconfiguration of the Referrer header.

2. Change Password to admin

Credentials	
Username	admin
Password	admin

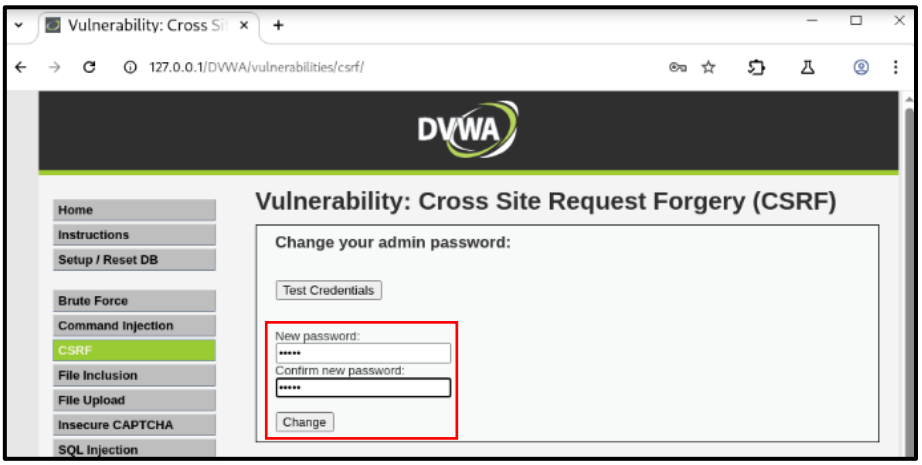


Figure 53 : Exploit CSRF 2.1- Test Credentials

To assess the effectiveness of the CSRF protection, an initial test is performed by changing the password through the web interface. The request is successfully processed, and the password is updated to **admin**.

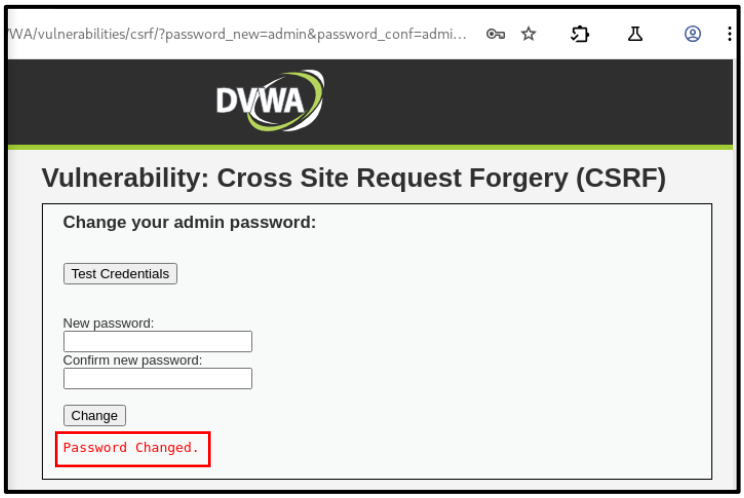


Figure 54 : Exploit CSRF 2.2 – Test Credentials

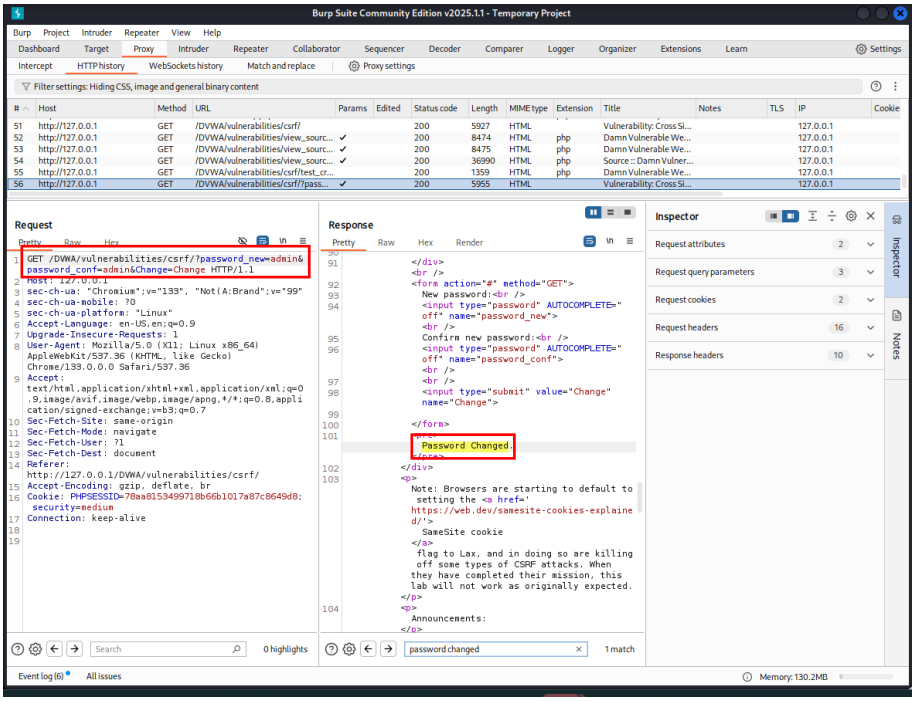


Figure 55 : Exploit CSRF 2.3 – Request Packet

The figures above show the successful changing of password with the request sent to the server and responses received.

3. Change Password on New Window

Credentials	
Username	admin
Updated Password	admin12345

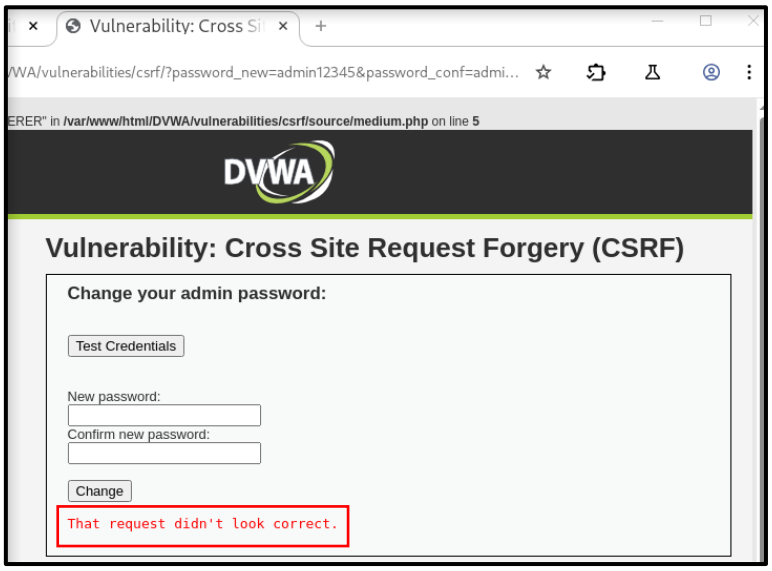


Figure 56 : Exploit CSRF 2.4 – Incorrect Request

Original URL	http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change
Modified URL	http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=admin12345&password_conf=admin12345&Change=Change

The successful **password change request** is then executed on a new browser window, with modifications made to the **password parameter**. However, upon submission, the webpage displays the following error message "**That request didn't look correct.**"

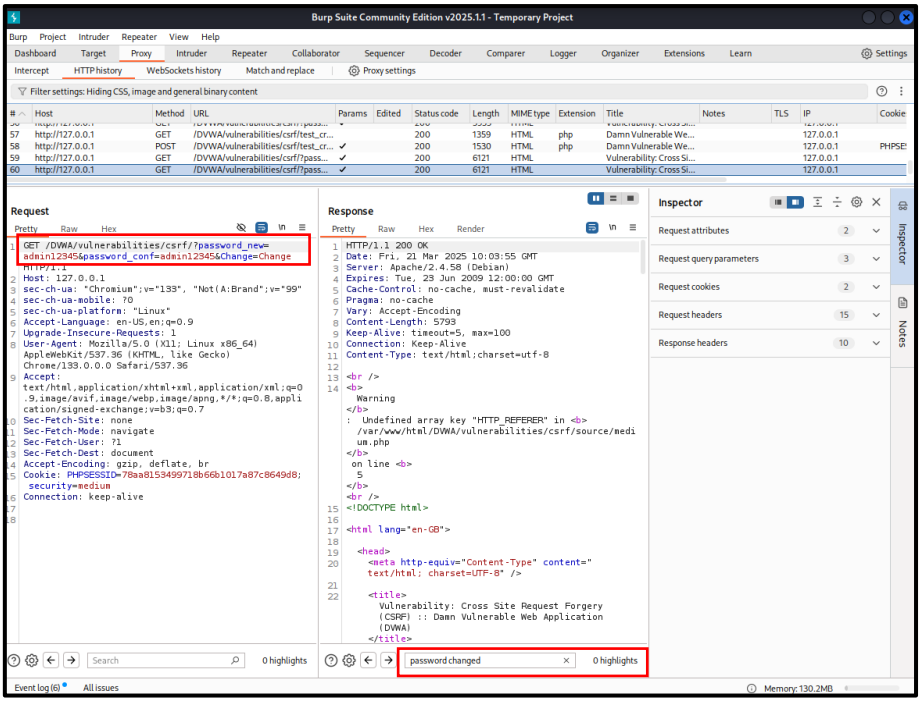


Figure 57 : Exploit CSRF 2.5 – Failed Change Password

Upon further examination of the request, it is observed that the **Referer** header is missing. This suggests that the application relies solely on this header for CSRF validation. The absence of the **Referer** header causes the request to fail, preventing the password from being changed.

4. Insert Referrer Header in Request Packet

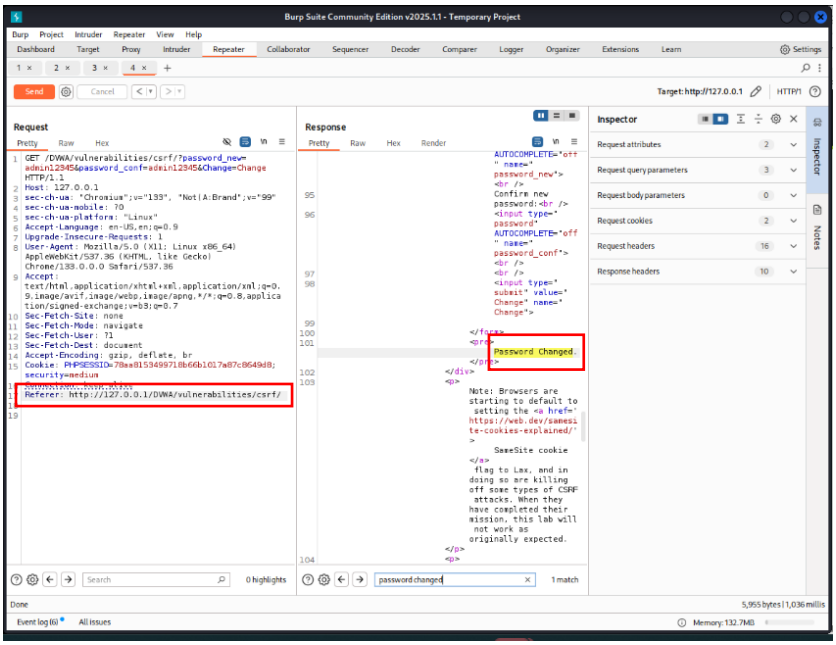


Figure 58 : Exploit CSRF 2.6 – Password Changed

Referer Header	Referer: http://127.0.0.1/DVWA/vulnerabilities/csrf/

To bypass the CSRF protection, the **Referer** header is manually inserted into the request, specifying the expected origin. Once the modified request is submitted, the password change is successfully processed. This confirms that the application's CSRF protection is entirely dependent on the **Referer** header, making it vulnerable to manipulation.

5. Password Changed Successfully

Tested Credential	
Username	admin
Password	admin12345

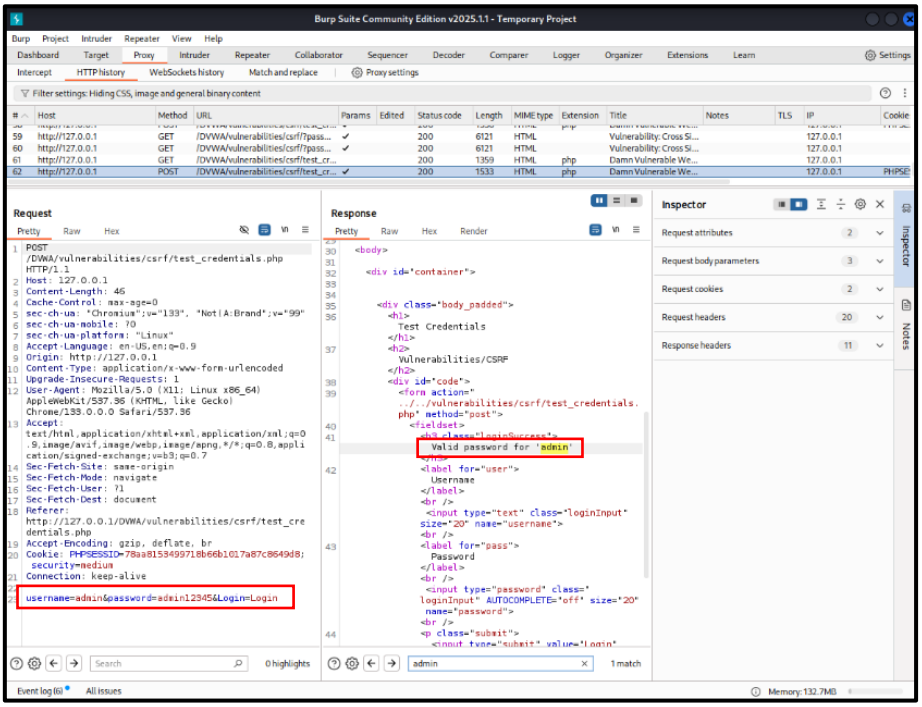


Figure 59 : Exploit CSRF 2.7 – Test Credentials

The figure above shows the testing of credentials using the password (**admin12345**). The message "**Valid password for "admin"**" indicates the successful changing of the password.

3.4 CSRF (CSRF Token Implemented)

1. Understand Source Code

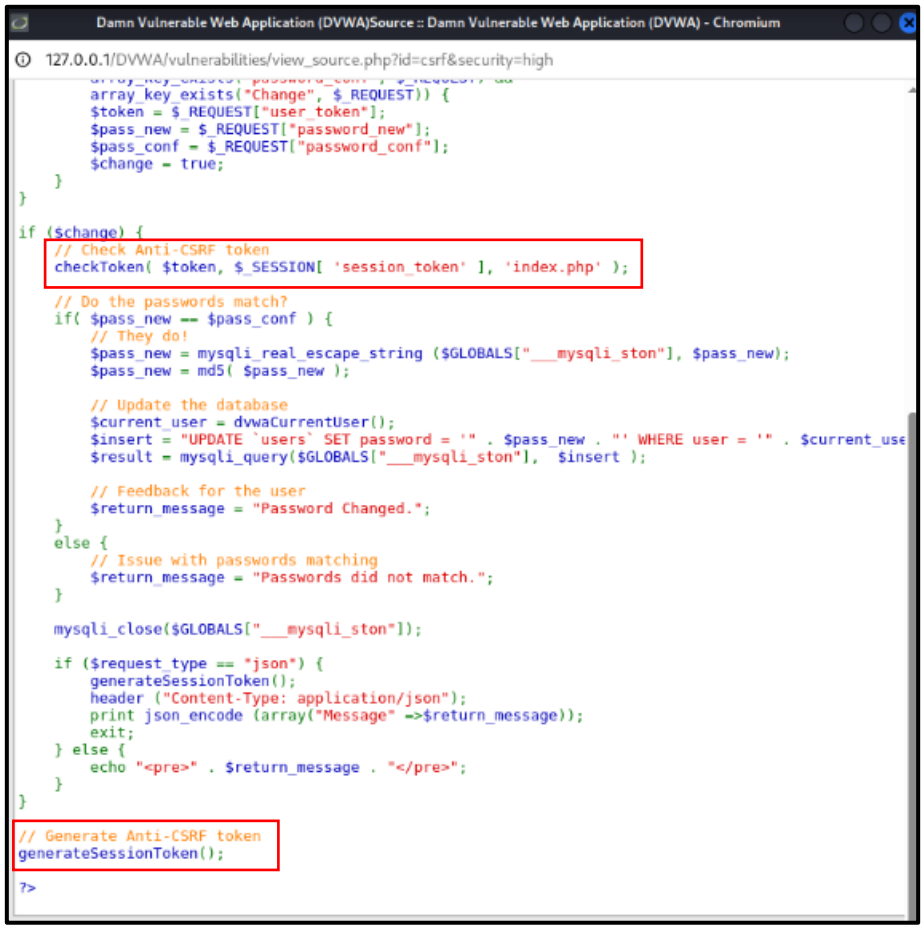


Figure 60 : Exploit CSRF 3 – Source Code

Before attempting to exploit the vulnerability, it is crucial to analyse the PHP source code of this section to understand the architecture of the webpage. According to the PHP source code shown, this section seems to be implemented with password change functionality and is implemented with a CSRF token (**user_token**). Upon each new session or request, a new CSRF token will be generated. This indicates every request requires user authentication using token validation. The application will only process if it verifies the submitted **user_token**.

Hence, this attack focuses on finding a way to bypass CSRF token verification such as obtaining a valid CSRF token before making a request.

2. Change Password to admin

Credential	
Username	admin
Password	admin

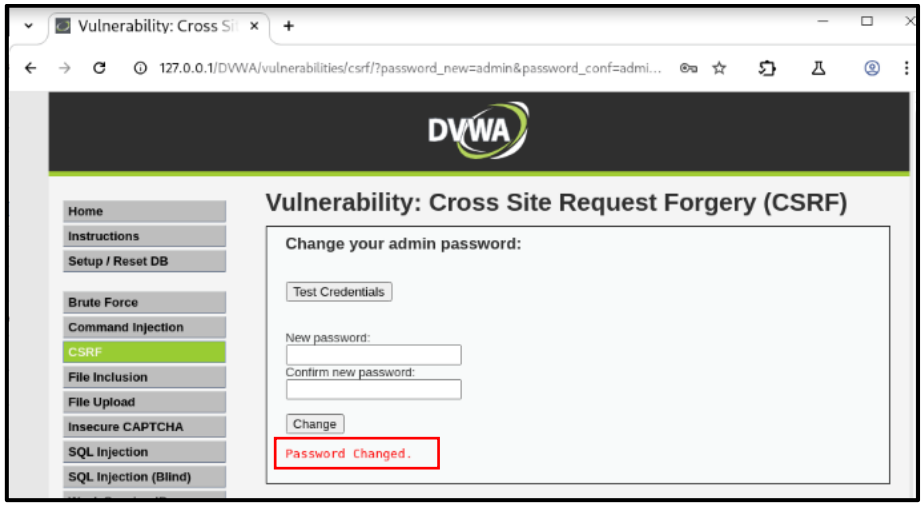


Figure 61: Exploit CSRF 3.1- Password Changed

The exploit starts with generating a successful password-changing request that can be used to check for further vulnerabilities. Firstly, changing the password to admin. The figure above shows password has been changed successfully.

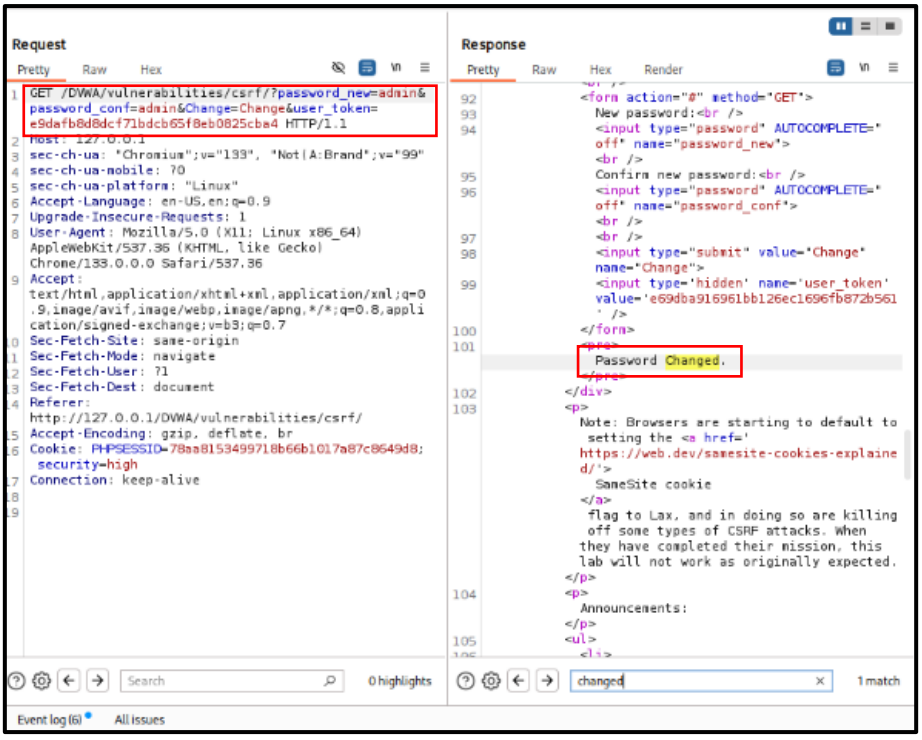


Figure 62: Exploit CSRF 3.2 – Request Packet

Upon successful changing of the password, a request packet will be generated. According to the figure shown above, the request packet is generated with parameters (**password_new**, **password_conf**, **user_token** and **Change**). The presence of **user_token** verifies that a valid CSRF token is required to perform the password change function.

3. Change Password on New Window

Credential	
Username	admin
Password	admin123

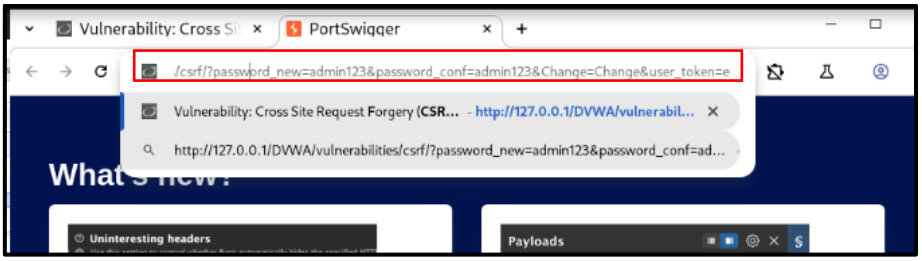


Figure 63: Exploit CSRF 3.3 – Change Password

Original URL	http://DVWA/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change&user_token=e9dafb8d8dcf71bdcb65f8eb0825cba4
Modified URL	http://DVWA/vulnerabilities/csrf/?password_new=admin123&password_conf=admin123&Change=Change&user_token=e9dafb8d8dcf71bdcb65f8eb0825cba4

The exploit continued by opening the successful change of password request in a new window browser that attempted to test the ability to change the password again using the same URL. This step is tested by submitting **admin123** as the new password by modifying the password parameter field.

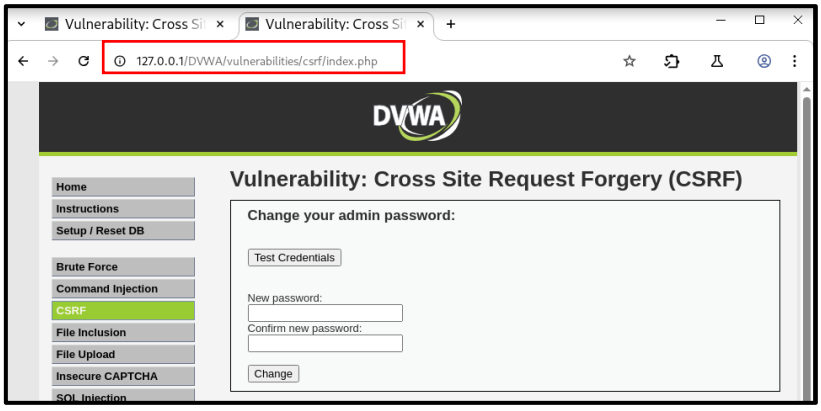


Figure 64: Exploit CSRF 3.4 – Request Not Processed

Upon request submission, the window browser is then redirected to an unrelatable page, which is the CSRF page without the 'password changed' message shown.

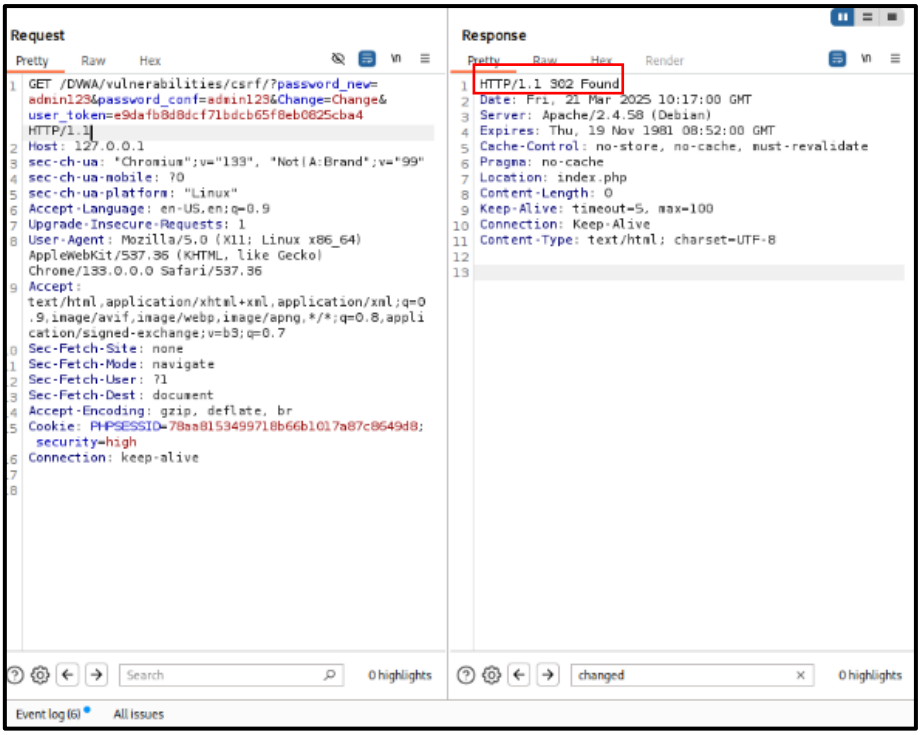


Figure 65: Exploit CSRF 3.5 – Request Packet

Upon checking on Burp Suite, the request packet shows **HTTP status 302 Found** indicating the request was **not successfully processed** instead of status 200 which indicates the successful submission. This reflects that the CSRF token is session-based and cannot be reused in a different browser session.

4. Modify Request Parameter to Change Password

Credential	
Username	admin
Password	admin12

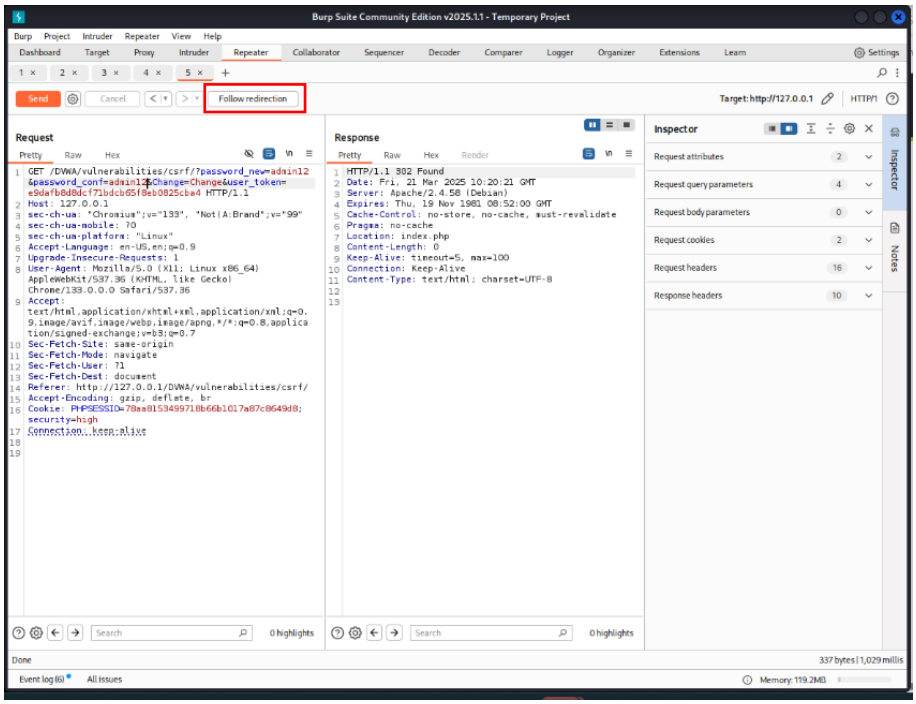


Figure 66: Exploit CSRF 3.6 – Change Password

Original API (parameter field)	/DVWA/vulnerabilities/csrf/?password_new=admin123&password_conf=admin123&Change=Change&user_token=e9dafb8d8dcf71bdcb65f8eb0825cba4
Modified API (parameter field)	DVWA/vulnerabilities/csrf/?password_new=admin12&password_conf=admin12&Change=Change&user_token=e9dafb8d8dcf71bdcb65f8eb0825cba4

In order to test the detailed redirection upon change password request submission, Burp Repeater will be utilized to intercept the password change request to modify the password parameters (**password_new** and **password_conf**) to **admin12**. Upon submitting the request, the server responds with the status **HTTP 302 Found** indicates the failure to process the request and it will be redirected to another page.

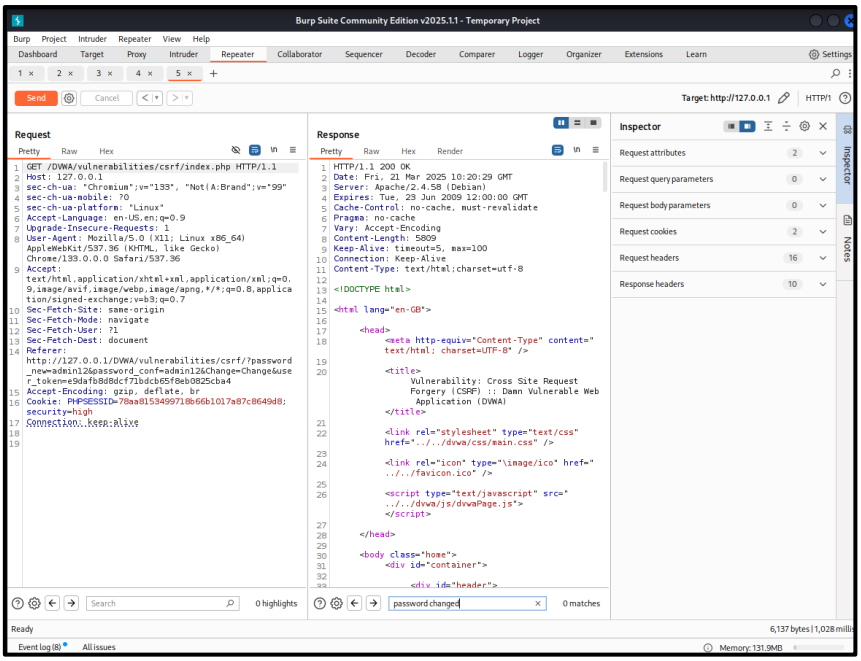


Figure 67: Exploit CSRF 3.7 – Unrelated Page Redirection

Upon following redirection, the page is redirected to **/csrf/index.php**, which is the main page of CSRF without displaying the "Password Changed" message. This indicates that the **request was rejected** due to an invalid or missing CSRF token.

Note:

This section of DVWA is designed with **anti-CSRF protection** which means each request requiring user authentication must include a **valid CSRF token** (**user_token**). This token is dynamically generated for each user session and helps prevent unauthorized actions from being performed on behalf of an authenticated user.

The role of the CSRF token is to verify that the request originates from the legitimate user and not from an attacker trying to forge a request. Each time a user submits a request, a new **unique token** is required, which means previously used tokens become invalid. If an attacker attempts to replay an old request without updating the token, the request will be rejected by the server.

Hence, to successfully exploit CSRF in this scenario, pen-testers are required to:

1. **Identify potential exposure of valid CSRF tokens** such as in GET requests, hidden form fields, JavaScript variables, or the HTML source code.
2. **Obtain a valid CSRF token** that has not been used yet.
3. **Inject the new token into the request** before sending it to the server.

Since the application uses a token-based verification method, any attempt to change user settings such as changing passwords, will be unsuccessful without a valid CSRF token. To find out if the CSRF token can be recovered and used, the next step is to examine the request flows and source code of the application.

5. Explore New CSRF Token

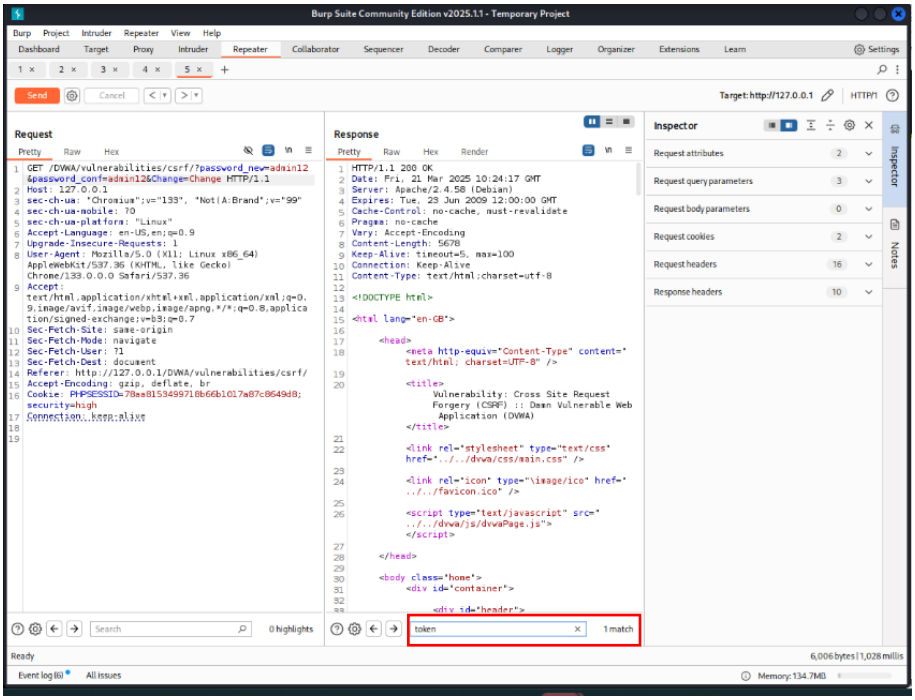


Figure 68: Exploit CSRF 3.8 – Token Found

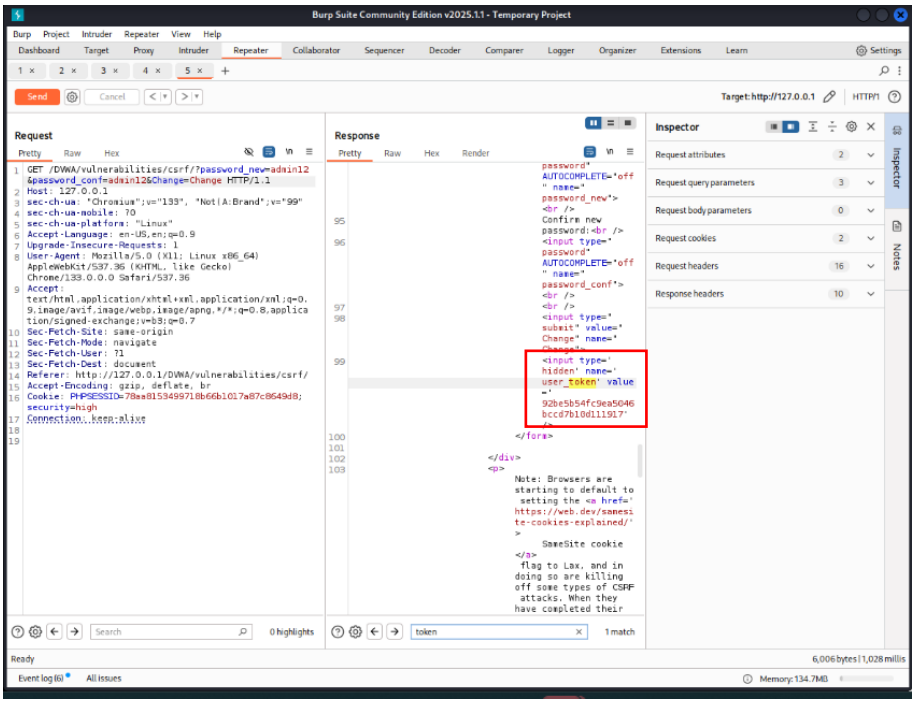


Figure 69: Exploit CSRF 3.9 – Token Found

Hidden Input Field	<input type='hidden' name='user_token' value='92be5b54fc9ea5046bccd7b10d111917'>
Valid Token	92be5b54fc9ea5046bccd7b10d111917

The exploit continues to explore possible fields that could include valid tokens such as HTML sources. According to the figure shown above, the latest CSRF token is stored in the HTML source code upon failure to change the password page within a hidden input field. This valid token can be retrieved and tested on the next steps by replacing the previous token with the latest valid token.

Credential	
Username	admin
Password	admin12

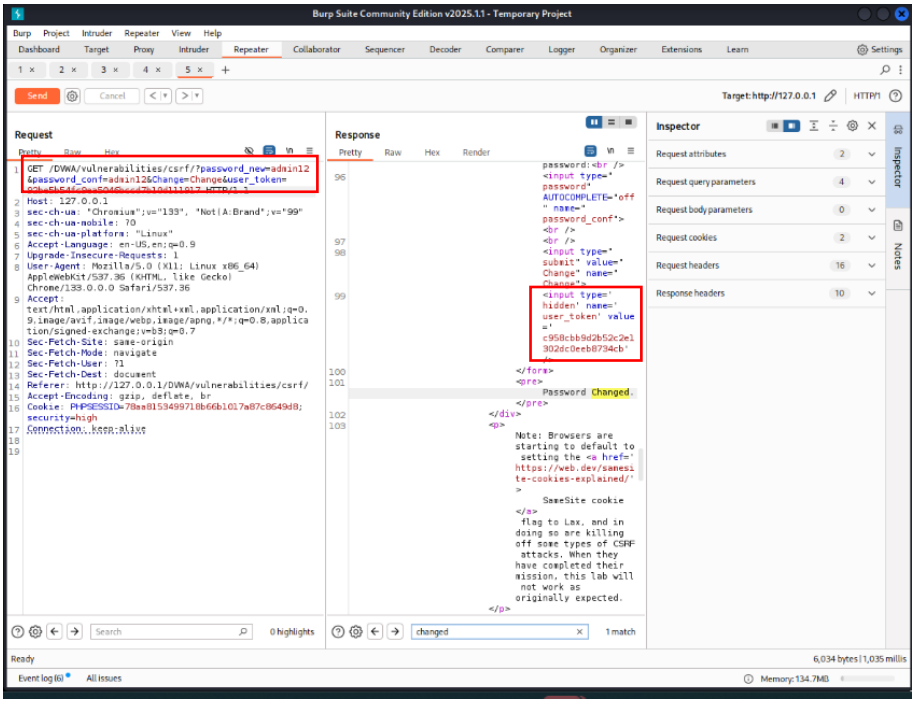


Figure 70: Exploit CSRF 3.10 – Token Found

Valid user_token	92be5b54fc9ea5046bccd7b10d111917

The latest valid token is used to craft a successful password change request by replacing the previous **user_token**. After modifying the **user_token** parameter in the **Change Password** request with the latest valid token, the request is successfully processed by the server. The response confirms that the password has been changed.

6. Password Changed Successfully

Credential	
Username	admin
Password	admin12

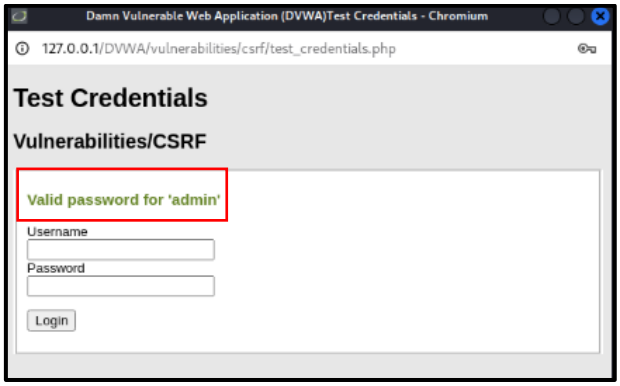


Figure 71: Exploit CSRF 3.11 – Test Credentials

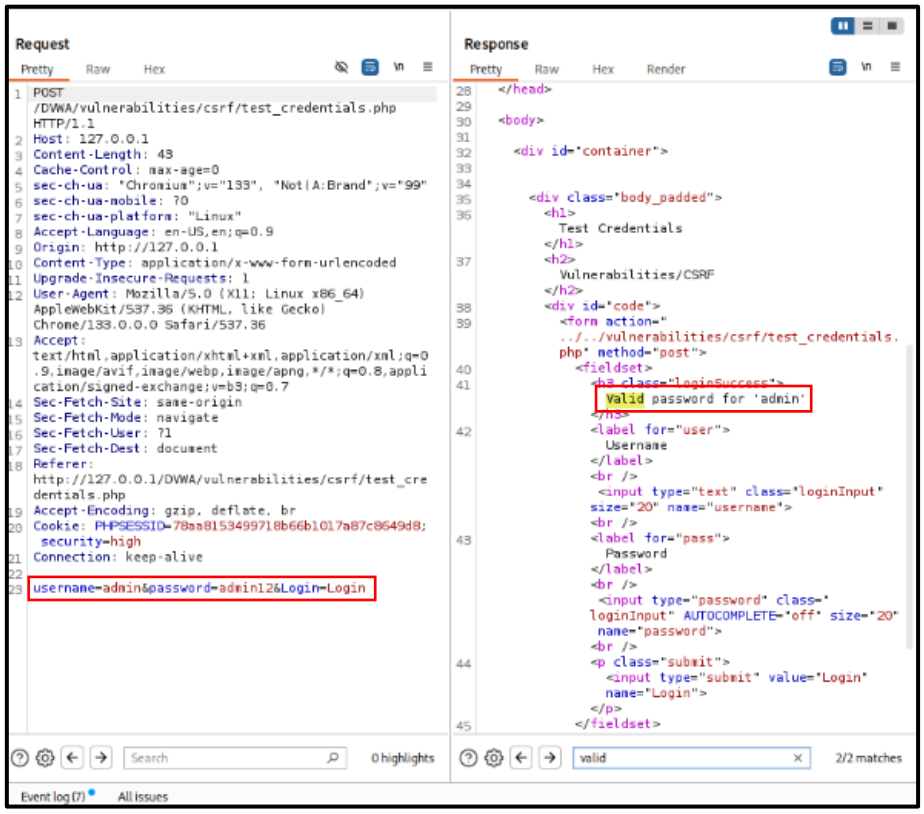


Figure 72: Exploit CSRF 3.12 – Valid Password

The above figures show that "**Valid password for admin**" uses credentials **admin** and **admin12** as username and password respectively. This result proves that the CSRF attack was effective in modifying the credentials without requiring direct user interaction.

3.5 SQL Injection (Extracting Users Password)

SQL injection is a critical web application vulnerability that allows an attacker to manipulate backend SQL databases by injecting malicious queries. In this assessment, **SQLMap**, a powerful penetration testing tool, is used to exploit **SQL injection vulnerabilities** present in the **Damn Vulnerable Web Application (DVWA)**. The following steps outline how an attacker can extract sensitive data, such as user credentials, from the DVWA database using SQLMap.

1. Analyse Source Code

SQL Injection Source

vulnerabilities/sqli/source/low.php

```php
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["_

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Get values
                $first = $row["first_name"];
                $last  = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
            }

            mysqli_close($GLOBALS["___mysqli_ston"]);
            break;

        case SQLITE:
            global $sqlite_db_connection;

            #$sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
            #$sqlite_db_connection->enableExceptions(true);

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
                while ($row = $results->fetchArray()) {
                    // Get values
                    $first = $row["first_name"];
                    $last  = $row["last_name"];

                    // Feedback for end user
                    echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
                }
            } else {
                echo "Error in fetch ".$sqlite_db->lastErrorMsg();
            }
            break;
    }
}

?>
```

Figure 73: Exploit SQL Injection 1 – Source Code

According to the provided source code, the PHP script supports two database management systems (DBMS): **MySQL and SQLite**. In both cases, the script constructs an SQL query where the **id** parameter is directly concatenated into the statement **without any input sanitization**. The PHP code retrieves user input through the **id** parameter, which is passed via an HTTP request and inserted directly into the SQL query without validation or filtering. This causes the script becomes vulnerable to **SQL injection attacks**, as it processes user input without proper security measures. This allows an attacker to manipulate the query, potentially retrieving, modifying, or deleting sensitive database records.

Hence, this exploitation conducts **penetration testing** to evaluate the security risks associated with this vulnerability and demonstrate how an attacker can exploit the **SQL injection** flaw to gain unauthorized access to sensitive information.

2. Submit a Request and Extract Cookies

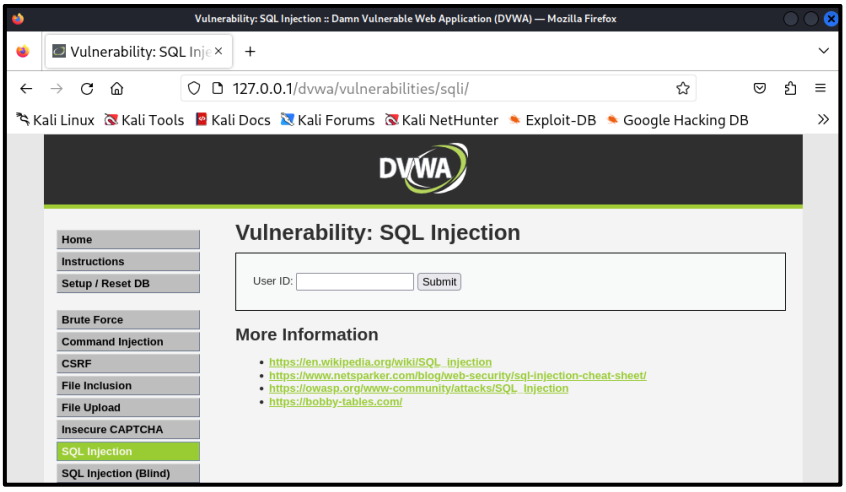


Figure 74 : Exploit SQL Injection 1.1 - Page

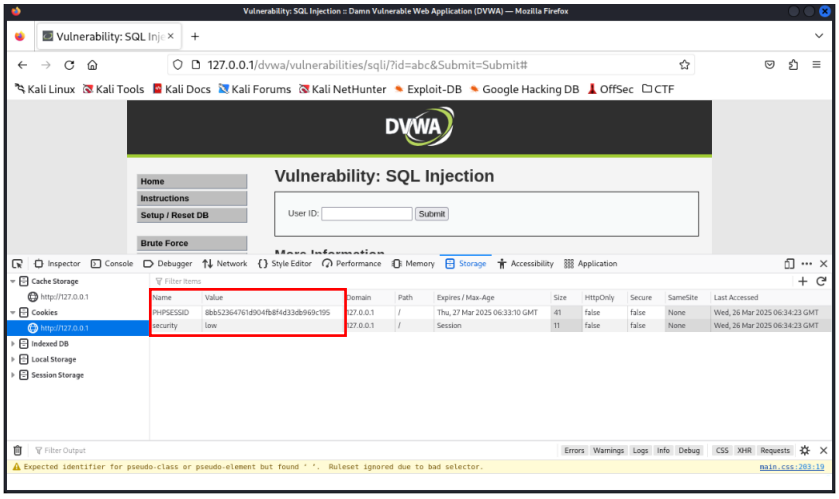


Figure 75 : Exploit SQL Injection 1.2 - Cookies

To initiate the exploitation process, the pentester first interacts with the vulnerable input field within DVWA, such as a search bar, and submits a User ID **abc**. Upon successful submission of the user ID, no changes or messages are shown on the page that mentions the unavailable user ID entered. This unexpected behaviour helps confirm the possible presence of an injection point.

PHPSESSID	8bb52364761d904fb8f4d33db969c195
security	low

Hence, to continue the exploits by using SQLMap to extract key insights, session cookies are needed to maintain access while interacting with the database.

3. Run SQLMap Commands to Exploit SQL Injection Vulnerabilities

Once the vulnerable endpoint is identified and the necessary session cookies are obtained, SQLMap is used to exploit the SQL injection vulnerability. This includes the retrieval of database information such as basic injection test to identify the vulnerability of target URL towards SQL Injection, database tables enumeration to analyse available databases, column names extraction to identify the possible information that can be extracted from the database, and data dumping to print detailed information of the database.

Basic Injection Test

sqlmap Command
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=abc&Submit=Submit#" --cookie="PHPSESSID=8bb52364761d904fb8f4d33db969c195;security=low"

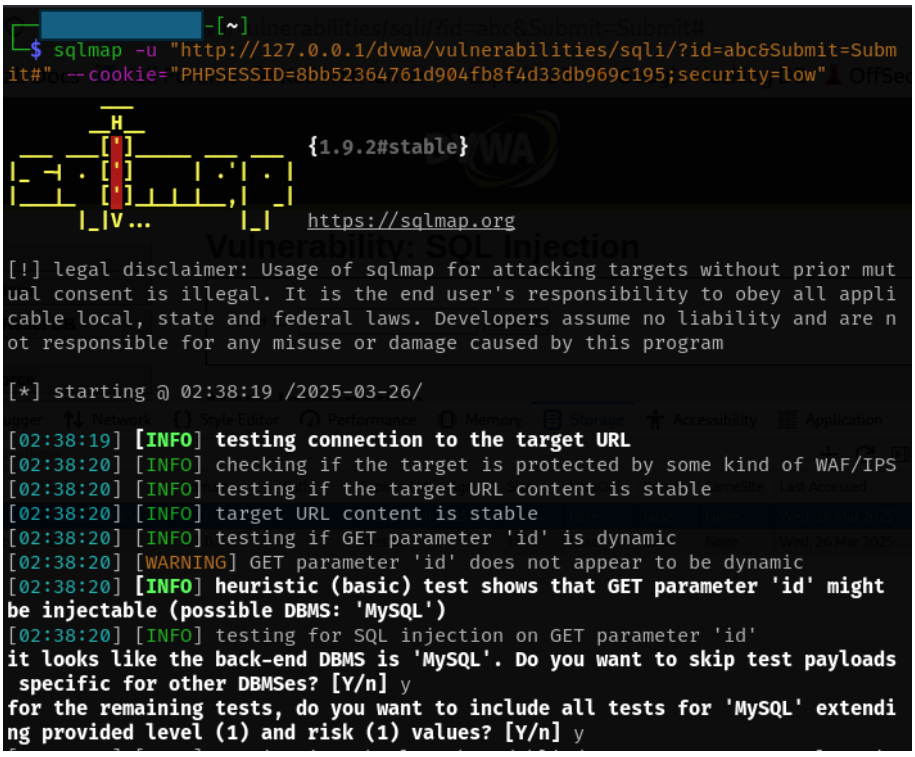


Figure 76 : Exploit SQL Injection 1.3 – SQLMap

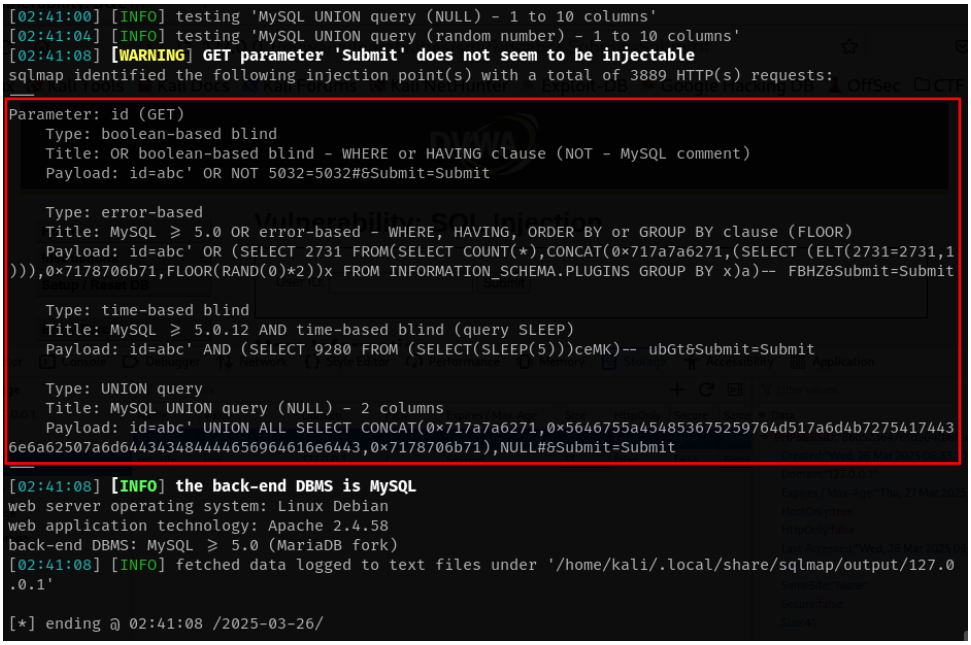


Figure 77 : Exploit SQL Injection 1.4 – SQLMap

According to the result of the command execution to test the vulnerability of the target URL towards SQL Injection, the target URL is vulnerable to SQL injection in which the **id** parameter can be detected as an injection point using multiple types of payloads.

Type SQL Injection	SQL Query / Payload
Boolean-Based Blind	id=abc' OR NOT 5032=5032#&Submit=Submit
Error-Based	id=abc' OR (SELECT 2731 FROM(SELECT COUNT(*),CONCAT(0x717a7a6271,(SELECT (ELT(2731=2731,1))),0x7178706b71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- FBHZ&Submit=Submit
Time-Based Blind	id=abc' AND (SELECT 9280 FROM (SELECT(SLEEP(5)))ceMK)-- ubGt&Submit=Submit
UNION Query	id=abc' UNION ALL SELECT CONCAT(0x717a7a6271,0x5646755a454853675259764d517a6d4b72754174436e6a62507a6d6443434844446569646 16e6443,0x7178706b71),NULL#&Submit=Submit

Hence, the existence of SQL Injection vulnerability in this context supports the ongoing exploitation of database to carry out more detailed exploitation focusing on analysing the degree of impact on the target web application.

Database Tables Enumeration

sqlmap Command
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=abc&Submit=Submit#" --cookie="PHPSESSID=8bb52364761d904fb8f4d33db969c195;security=low" --tables

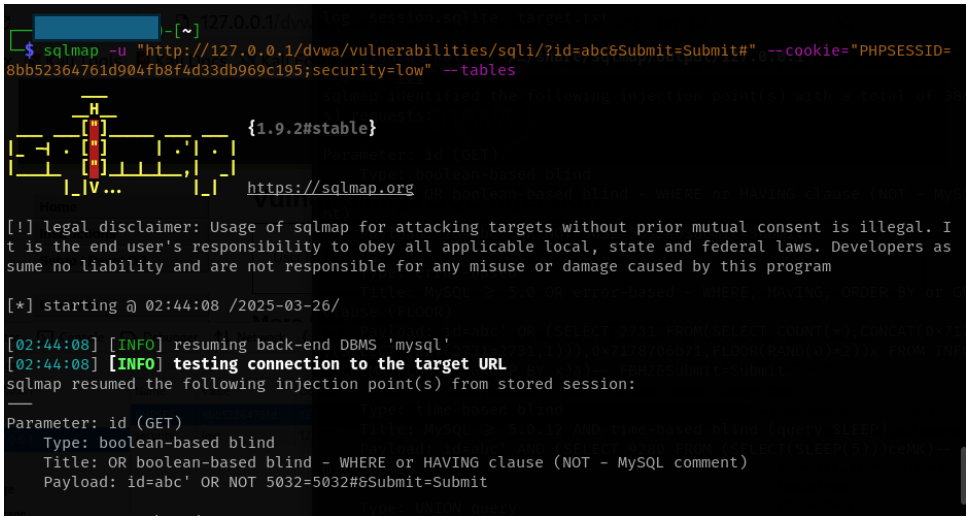


Figure 78 : Exploit SQL Injection 1.5 – SQLMap

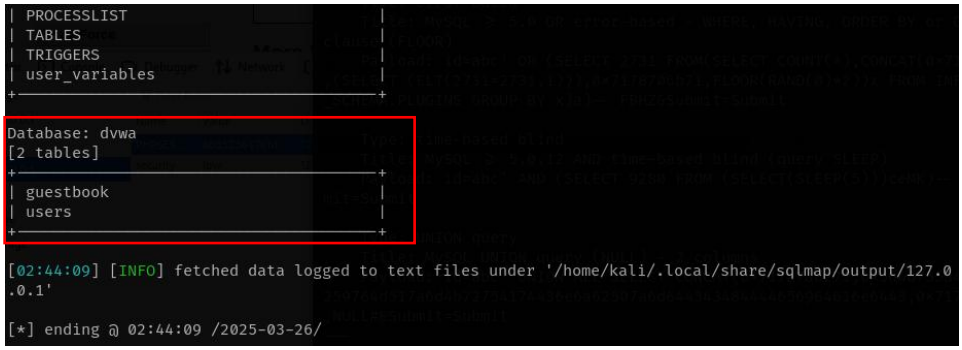


Figure 79 : Exploit SQL Injection 1.6 – SQLMap

According to the result shown above based on the command executed (**--table**) to explore the available database table, there are 2 tables that exist in **dvwa** database: **guestbook and users**.

In order to exploit the login credentials of DVWA users, the assessment will be focused on exploiting users' sensitive information.

Column Names Extraction

sqlmap Command
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=abc&Submit=Submit#" --cookie="PHPSESSID=8bb52364761d904fb8f4d33db969c195;security=low" --columns -T users --batch

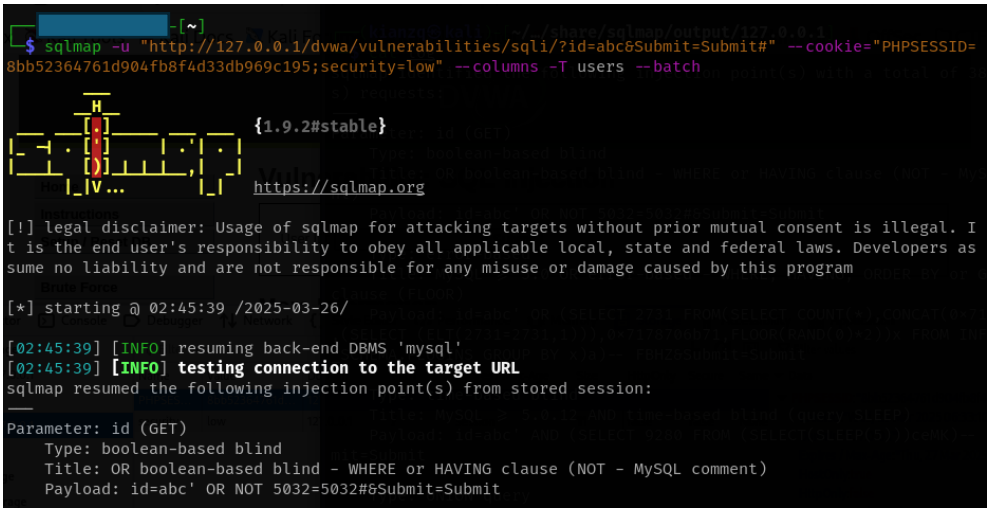


Figure 80 : Exploit SQL Injection 1.7 – SQLMap

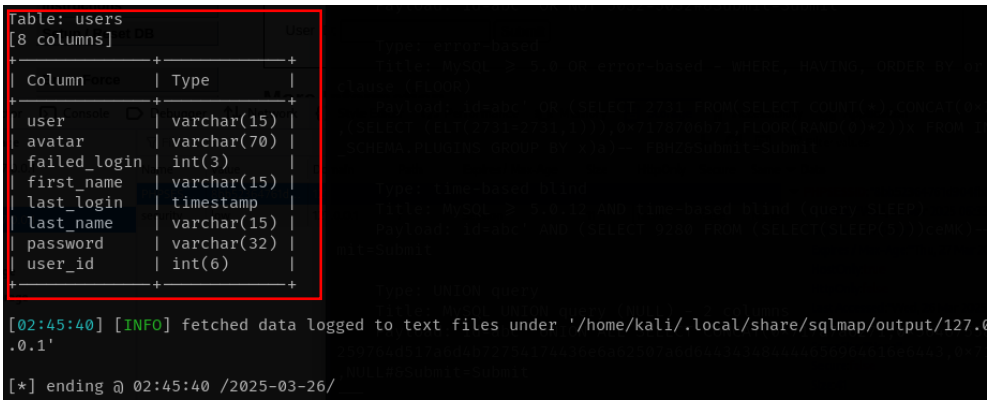


Figure 81 : Exploit SQL Injection 1.8 – SQLMap

To further exploit information included in **dvwa** users table, the assessment will first extract column names to identify the field of information contained within the specified database table.

The command "**--columns -T users**" extracts the column name of the user's table while the command "**--batch**" executes SQLMap in an automated mode which does not require the manual script confirmation. Based on the result shown, the password column exists in the user table and serves as a possible path to extract the user's password.

4. Extracting Users Sensitive Information

sqlmap Command
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=abc&Submit=Submit#" --cookie="PHPSESSID=8bb52364761d904fb8f4d33db969c195;security=low" --dump -T users --batch

Figure 82 : Exploit SQL Injection 1.9 – SQLMap

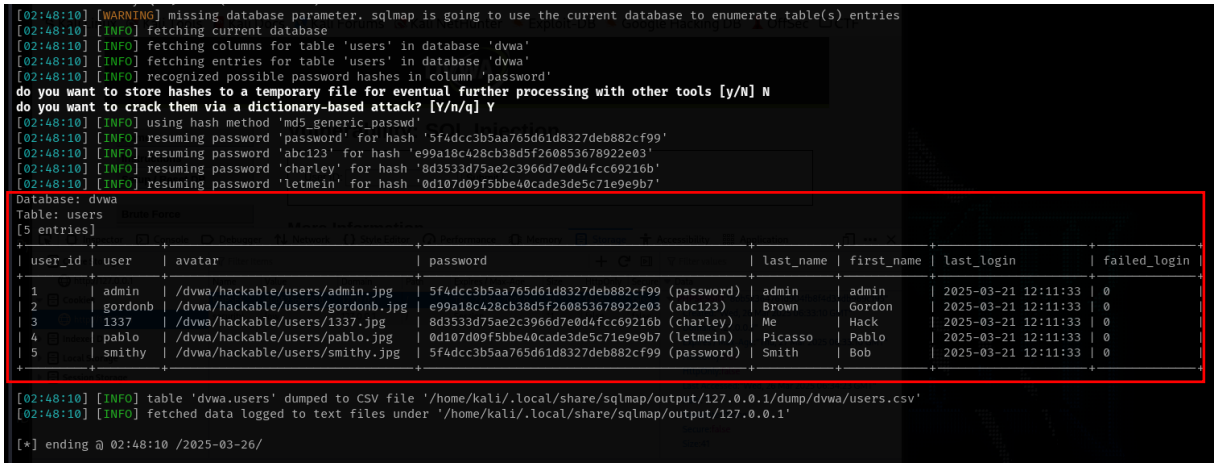


Figure 83 : Exploit SQL Injection 1.10 – SQLMap

To extract user passwords from **dvwa**'s user's table, the command "**--dump**" is executed to retrieve all data. Based on the result shown in the figure above, all user details have been exploited, including sensitive passwords that are encrypted and in plaintext form.

5. Correct Login Credentials Extracted

Credential	
Username	gordonb
Password	abc123

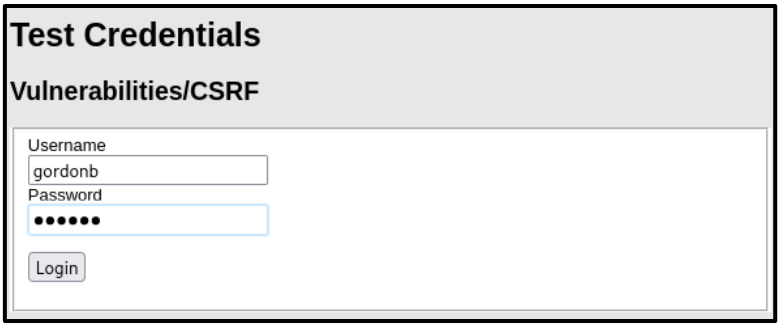


Figure 84 : Exploit SQL Injection 1.11 – Test Credentials

Figure 85 : Exploit SQL Injection 1.12 – Valid Password

Extracted credentials are then tested such as **gordonb** with password (**abc123**) and the message "**Valid password for "gordonb"**" appeared indicating the available sensitive data can be extracted through SQL injection.

3.6 Potential Impact of Vulnerabilities

3.6.1 CSRF

No CSRF Prevention Measures Implemented

Attackers can easily carry out illegal actions on behalf of authenticated users if a web application lacks CSRF prevention features. This implies that private operations, including altering user preferences, changing passwords, or carrying out transactions, can be carried out without the victim's awareness. Since attackers can create malicious links or forms that, when clicked by a logged-in user, immediately execute illegal requests, the application is extremely vulnerable due to the lack of CSRF security.

Referer-Based Validation Implemented but Easily Bypassed

Referer-based validation is used by some online applications to guard against Cross-Site Request Forgeries (CSRF). Before completing sensitive operations, this method verifies that the request is coming from a trustworthy domain. This security measure is weak, though, and is frequently circumvented. In order to insert a genuine referer into the request, attackers can alter the referer header using programs like Burp Suite or specially written scripts. Furthermore, the referer header may be stripped by certain browsers and privacy settings, rendering this protection mechanism unreliable.

CSRF Token Implemented but Exposed in HTML Code

The CSRF token is a unique session-based value which is a more secure method by validating every request sent to the server. However, an attacker with access to the website source can still obtain the CSRF token if it is embedded in the HTML code, for example, inside a hidden input field. This reduces the security since, in the event that there are XSS vulnerabilities, attackers can use JavaScript to extract the token or other methods like GET requests or leaked answers. An attacker may still carry out illegal activities even if they are able to acquire and reuse a legitimate token.

3.6.2 SQL Injection

Unauthorized Database Access

Attackers can directly interact with the backend database without the necessary authorization if a web application is susceptible to SQL injection. They can access private user data, get around authentication procedures, and maybe take over the system as administrators thanks to this. A serious security compromise could result from an attacker using this vulnerability to query, alter, or remove important data.

Sensitive Data Extraction

Threat actors can retrieve financial information, private company information, and personally identifiable information (PII) from databases through SQL injection attacks. Legal repercussions, monetary damages, and harm to one's reputation could arise from GDPR, PCI-DSS, or other compliance infractions. Businesses that hold client information, including trade secrets, credit card numbers, and medical records, are especially vulnerable.

User Credential Theft

Attackers can retrieve password hashes, usernames, and other private information from databases by using SQL injection. An attacker can quickly crack the passwords and obtain unauthorized access to user accounts if the program does not employ robust encryption or hashing mechanisms. Identity theft, account takeovers, and additional user data exploitation may result from this.

Data Manipulation and Integrity Issues

Database records can be changed or removed by an attacker taking advantage of a SQL injection vulnerability. This implies that they might tamper with system records, change money transactions, or remove crucial data—possibly leading to serious operational interruptions. Loss of data integrity, particularly when it affects consumer or business-critical data, can harm an organization's reputation and have legal repercussions.

4.0 Countermeasures & Recommendations

In the aftermath of the Vulnerability Assessment and Penetration Testing (VAPT) conducted on **Damn Vulnerable Web Application (DVWA)**, it is crucial to implement effective **countermeasures** and **security recommendations** to mitigate the identified vulnerabilities. The assessment revealed critical security flaws, including **Cross-Site Request Forgery (CSRF) and SQL Injection**, which can lead to unauthorized actions and data breaches if left unaddressed. Implementing **robust preventive measures**, such as **anti-CSRF tokens, referer-based validation, and parameterized queries**, can significantly reduce the risk of exploitation.

Beyond these specific threats, the evaluation also highlighted **wider security concerns related to the OWASP Top 10**, including **Security Misconfiguration, Broken Access Control, Insecure Design, and Software & Data Integrity Failures**. Addressing these issues requires a **comprehensive security strategy** that incorporates **secure coding practices, proper access control mechanisms, and adherence to industry security standards**. By strengthening these areas, organizations can **enhance the overall security posture of their applications, prevent future exploitation, and ensure compliance with best security practices**. The following sections will outline targeted countermeasures for **CSRF and SQL Injection**, along with strategic recommendations to mitigate the broader security risks identified in the assessment.

4.1 Countermeasures on CSRF and SQL Injection

Vulnerability	Issue	Countermeasures
CSRF: No Prevention Measures	The application lacks CSRF protection, allowing attackers to execute unauthorized actions on behalf of authenticated users.	• **Put CSRF Tokens into Practice:** Create and verify distinct, erratic CSRF tokens for every session. In all state-changing requests (such as POST, DELETE, and PUT), the token ought to be included. • **Use the SameSite cookie attribute:** To limit cross-origin requests, set cookies with SameSite=strict or SameSite=lax.
CSRF: Referer-Based Validation Implemented but Easily Bypassed	The system relies on the HTTP Referer header for CSRF protection, which is unreliable and can be spoofed or removed by attackers.	• **Employ Secure CSRF Tokens:** Rather than depending solely on referer-based validation, use CSRF tokens. • **Use Origin Header Validation:** For secure HTTPS queries, validate the Origin header rather than the Referer header. • **Put Content Security Policy (CSP) into effect:** Limit the sources from which requests can be made.
CSRF: CSRF Token Implemented but Exposed in HTML Code	The CSRF token is visible in the page source, allowing attackers to easily retrieve and use it for attacks.	• **Store Tokens in HTTP-Only Cookies:** Use HTTP-Only cookies to stop client-side access to tokens. • **Use Token Regeneration:** To avoid reuse, regenerate and invalidate tokens following each request. • **Employ Secure Headers:** To stop data leaks, use CSP and other secure headers.
SQL Injection: Extracting Sensitive User Information	The system allows SQL injection, enabling attackers to extract sensitive data such as login	• **Employ Prepared Statements and Parameterized Queries:** To guard against injection attacks, use parameterized queries instead of straight SQL queries.

	credentials from the database.	• **Implement Least Privilege Principle:** Limit database user privileges to read-only access for queries that don't alter data. • **Clean User Inputs:** To prevent harmful SQL payloads, use input validation. • **Install a Web Application Firewall (WAF):** To filter and prevent SQL injection attempts, use a WAF.

4.2 Other Recommendations

In addition to enhance the overall security posture of the target system, it is essential to implement robust countermeasures against common vulnerabilities identified in the **OWASP Top 10**. Addressing these vulnerabilities helps prevent potential exploitation, protects sensitive data, and ensures compliance with security best practices.

4.2.1 Injection Attacks

One of the most critical vulnerabilities in web applications is **injection attacks**, such as SQL injection, command injection, and LDAP injection. By inserting malicious code into instructions or queries, attackers take advantage of these flaws, resulting in data breaches and illegal system access.

In order to prevent Injection Attacks, developers should avoid direct database query manipulation by using prepared statements and parameterized queries to reduce injection concerns. Furthermore, prior to processing, potentially hazardous information can be filtered out by implementing input validation and sanitization. Implementing least privilege access controls limits database rights, reducing possible damage, and using Object-Relational Mapping (ORM) frameworks to provide built-in defenses against injection attacks. (*A03 Injection - OWASP Top 10:2021*, 2021)

4.2.2 Security Misconfigurations

Security misconfiguration is another common weakness that exposes web applications to cyber threats. This includes improper default settings, unnecessary services, and the lack of security headers.

To prevent security misconfigurations, organizations should eliminate default credentials, disable unnecessary features and services, and make sure secure configuration baselines adhere to CIS benchmarks. Layers of defense against different types of attacks are added by implementing security headers like X-Frame-Options, HTTP Strict Transport Security (HSTS), and Content Security Policy (CSP). Organizations can also identify and address misconfigurations instantly by turning on logging and monitoring. (*A05 Security Misconfiguration - OWASP Top 10:2021*, 2021)

4.2.3 Broken Access Control

Broken access control is a major risk that allows unauthorized users to access restricted areas of a system. Attackers can exploit poorly implemented access control mechanisms to manipulate permissions, escalate privileges, or access sensitive information.

To address this, Role-Based Access Control (RBAC) should be implemented by businesses in order to limit user access according to predetermined roles. By strengthening authentication procedures, Multi-Factor Authentication (MFA) lowers the possibility of unwanted access. User sessions are shielded from attempts at hijacking by employing secure session management techniques, such as the usage of HttpOnly and Secure cookies. Permissions are in line with security policies and business needs when frequent access control audits are carried out. (*OWASP Top 10 Broken Access Control Explained*., 2023)

4.2.4 Insecure Design

Insecure design poses a significant threat when security is not considered during the software development lifecycle. Without proper planning, applications may lack essential security controls, making them vulnerable to attacks.

To mitigate this risk, a secure Software Development Lifecycle (SDLC) that incorporates security measures from the very beginning of design is something that enterprises should implement. Proactive security measures can be implemented by identifying possible attack vectors early on through threat modelling and risk assessments. By ensuring that developers follow standard practices, OWASP Secure Coding Guidelines lower the possibility of vulnerabilities being introduced. Additionally, protecting sensitive data from unwanted access is facilitated by implementing encryption standards like RSA and AES-256. (*A04 Insecure Design - OWASP Top 10:2021*, 2021)

4.2.5 Software and Data Integrity Failure

Lastly, **software and data integrity failures** can occur when attackers exploit vulnerabilities in software updates, third-party dependencies, and supply chain processes. Ensuring the integrity of software components is crucial in preventing unauthorized modifications.

In this case, cryptographic signatures and checksums should be used by companies to ensure the integrity of software and libraries when they download or update third-party components. Software update code signing guarantees that only authentic updates are installed and guards against tampering. Strict repository access controls and ongoing dependency vulnerability monitoring with tools like OWASP Dependency-Check and Snyk can help find out-of-date or hacked software components, which will further secure the software supply chain. Additionally, to prevent the loss or corruption of important data, safe backup and recovery procedures must be in place. (*A08 Software and Data Integrity Failures - OWASP Top 10:2021*, 2021)

By adopting these security measures, organizations can **significantly enhance their security posture** and protect their web applications against evolving cyber threats. Conducting regular **Vulnerability Assessment and Penetration Testing (VAPT)** using tools like **OWASP ZAP, Burp Suite, and SQLMap** ensures continuous monitoring and improvement of security defenses. Aligning with security frameworks such as **OWASP, NIST, and ISO 27001** further strengthens the security strategy, providing a structured approach to risk mitigation and compliance.

5.0 Conclusion

In conclusion, the Vulnerability Assessment and Penetration Testing (VAPT) conducted on Damn Vulnerable Web Application (DVWA) using OWASP ZAP for Vulnerability Assessment (VA) and Burp Suite and SQLMap for Penetration Testing (PT) has provided valuable insights into the security weaknesses of the target system. By leveraging these industry-standard tools, the assessment successfully identified and exploited various vulnerabilities, such as injection attacks, broken access control, security misconfigurations, and other critical weaknesses outlined in the OWASP Top 10.

OWASP ZAP played a crucial role in detecting vulnerabilities by performing automated scanning, identifying security flaws, and generating comprehensive reports. This proactive approach to vulnerability assessment ensured that known weaknesses, such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF), were effectively analysed before exploitation. The use of Burp Suite for manual testing, interception, and request manipulation allowed for a deeper understanding of how attackers could exploit authentication flaws, session management weaknesses, and privilege escalation vulnerabilities. Additionally, SQLMap was instrumental in automating SQL injection attacks, demonstrating the ability to extract sensitive data and compromise the database, emphasizing the importance of securing backend systems.

The findings from this assessment highlight the need for stronger security controls, improved authentication mechanisms, and better access control policies to mitigate potential threats. Implementing parameterized queries and prepared statements can effectively prevent SQL injection while enforcing Role-Based Access Control (RBAC) and Multi-Factor Authentication (MFA) strengthens user authentication and authorization. Addressing security misconfigurations by properly configuring servers, disabling unnecessary features, and ensuring secure session management can further reduce attack surfaces.

Regular security assessments, continuous monitoring, and adherence to industry standards such as OWASP, NIST, ISO 27001, and PCI-DSS are critical to maintaining a robust security posture. By integrating secure coding practices, periodic vulnerability scanning, and proactive penetration testing, organizations can prevent future exploitation, minimize security risks, and enhance the overall resilience of their web applications. This assessment underscores the importance of VAPT as a continuous process rather than a one-time evaluation, ensuring that security defenses remain effective against evolving cyber threats.

6.0 References

Ashwani K. (2023, September 15). What is OWASP ZAP and use cases of OWASP ZAP? - DevOpsSchool.com. DevOpsSchool.com. https://www.devopsschool.com/blog/what-is-owasp-zap-and-use-cases-of-owasp-zap/

A03 Injection - OWASP Top 10:2021. (2021). Owasp.org. https://owasp.org/Top10/A03_2021-Injection/

OWASP Top 10 Injection Attacks Explained. (2023, August 17). Securityjourney.com; Security Journey. https://www.securityjourney.com/post/owasp-top-10-injection-attacks-explained

A05 Security Misconfiguration - OWASP Top 10:2021. (2021). Owasp.org. https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

A01 Broken Access Control - OWASP Top 10:2021. (2021). Owasp.org. https://owasp.org/Top10/A01_2021-Broken_Access_Control/

A04 Insecure Design - OWASP Top 10:2021. (2021). Owasp.org. https://owasp.org/Top10/A04_2021-Insecure_Design/

A08 Software and Data Integrity Failures - OWASP Top 10:2021. (2021). Owasp.org. https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/

Burp Suite Tutorial (Part 1): Introduction to The Burp Suite Proxy | Cybrary. (2021). Cybrary.it. https://www.cybrary.it/blog/burp-suite-tutorial-part-1-introduction-to-the-burp-suite-proxy

OWASP Top 10 Broken Access Control Explained. (2023, August 10). Securityjourney.com; Security Journey. https://www.securityjourney.com/post/owasp-top-10-broken-access-control-explained

OWASP Top 10 Insecure Design Explained. (2023, August 24). Securityjourney.com; Security Journey. https://www.securityjourney.com/post/owasp-top-10-insecure-design-explained

Part 3: Using Burp Suite Repeater More Efficiently | Cybrary. (2021). Cybrary.it. https://www.cybrary.it/blog/part-3-using-burp-suite-repeater-more-

efficiently#:~:text=Burp%20Repeater%20essentially%20allows%20repeating,to%20Forward%20or%20Drop%20requests.

SQLMAP | Bugcrowd. (2022). Bugcrowd. https://www.bugcrowd.com/glossary/sqlmap/#:~:text=SQLMAP%20is%20an%20open%2Dsource,features%20to%20support%20penetration%20testing.