

**Name:** Isha Sangpal

**Email:** sangpalisha@gmail.com

**Batch:** March B1

**Level Completed:** Advance

# Table of Content

S.NO	TITLE	Page No
1	Introduction	3
2	Information About the Machine	5
3	Attack Vector Plan	6
	a. 4.1 Port Scanning b. Directory Brute Force c. Login Traffic Interception d. Input Validation & Vulnerability Testing	
4	Conclusion	18
5	References	19
6	Resources Used	20

# 1. Introduction

The purpose of this penetration test was to assess the security posture of a deliberately vulnerable web application designed to simulate real-world attack scenarios. This particular instance represented a more advanced, hard-difficulty environment, requiring a deeper level of reconnaissance, exploitation, and critical thinking to identify and leverage its weaknesses.

The testing environment mimicked conditions commonly found in misconfigured or poorly secured production systems—where legacy services, weak input validation, and insecure communication channels coexist. The goal was to approach the application from an attacker’s perspective, uncover potential vulnerabilities, and document both the path taken and the logic behind each decision.

Throughout the engagement, the focus was not just on finding “any” vulnerability, but on understanding how different components of the system interact—and how flaws in one area can ripple into others. From initial reconnaissance to targeted exploitation, each stage of the assessment aimed to reveal tangible security gaps that could be abused by a real attacker.

This wasn’t a point-and-click CTF. It required chaining multiple discoveries together: from identifying open ports and directory structures to intercepting unencrypted traffic and crafting payloads that bypass weak input sanitization. The nature of the vulnerabilities—such as SQL injection, reflected XSS, and exposed credentials via plaintext traffic—highlighted the risks of insecure development practices and a lack of layered defense.

By the end of this assessment, a clear attack narrative was established—from initial access to full compromise—and this report documents every critical step. The intention is not just to point fingers at what's broken, but to offer meaningful insights and recommendations that could help improve the security of similar systems in real-world deployments.

Ultimately, this report serves as both a record of the engagement and a learning resource—showcasing how thoughtful, methodical pentesting can uncover weaknesses that automated tools alone might miss.

## **2. Information About the Machine**

Primary Host (Main Environment):

Operating System: Ubuntu 24.04 LTS

Role: Main system used for executing all ethical hacking tasks.

Handled payload hosting, reconnaissance, traffic interception, and wireless attacks using tools like nmap, dirb, wireshark, aircrack-ng, and the Metasploit Framework.

### 3. All the attack vector plans and all the initiated attacks

Given the advanced difficulty of the target environment, a structured, multi-phase attack plan was essential. The goal wasn't just to find isolated vulnerabilities, but to follow a logical progression—from identifying exposed services to uncovering flaws deep within the application's logic. This wasn't about running tools blindly; it was about understanding the system's architecture and pressure-testing it step by step.

Each phase of the plan informed the next, with findings at one layer unlocking opportunities further down the stack. Below is a breakdown of the attack strategy and the rationale behind each step.

#### a. Port Scanning – Establishing the Surface:

##### **Severity:**

**Low (2.0/10)** – While port scanning itself doesn't exploit a vulnerability, it reveals the attack surface and is a crucial first step in identifying exploitable services.

**Objective:** Identify open ports and services on the target to understand its accessible attack surface.

##### **Steps Performed:**

1. DNS Resolution with nslookup

First, I ran an nslookup query to resolve the domain name testphp.vulnweb.com to its corresponding IP address. The server responded with:

Resolved IP: 44.228.249.3

This confirmed the domain was live and reachable, and gave me the IP for further scanning.

**Command used:** nslookup testphp.vulnweb.com

```
spectre@fsociety:~/Desktop/IshaSangpal$ nslookup testphp.vulnweb.com
Server:          127.0.0.53
Address:         127.0.0.53#53

Non-authoritative answer:
Name:   testphp.vulnweb.com
Address: 44.228.249.3
```

## 2. Basic Nmap Scan for Open Ports

I followed up with a basic Nmap scan to identify open ports. The scan revealed that port 80/tcp was open, indicating that the web server is serving HTTP traffic.

**Command used:** nmap testphp.vulnweb.com

```
spectre@fsociety:~/Desktop/IshaSangpal$ nmap testphp.vulnweb.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-03-21 09:20 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.30s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 33.55 seconds
```

## 3. Aggressive Scan for Service and Version Detection

To gather more in-depth information, I ran an aggressive Nmap scan (-A flag), which enabled service detection, OS fingerprinting, and script scanning.

Results:

Port 80 was running nginx version 1.19.0

The HTTP title returned was: "Home of Acunetix Art"

**Command used:** nmap -A testphp.vulnweb.com

```
spectre@fsociety:~/Desktop/IshaSangpal$ nmap -A testphp.vulnweb.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-03-21 09:24 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.29s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
80/tcp    open  http    nginx 1.19.0
|_http-title: Home of Acunetix Art

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 54.92 seconds
```

#### 4. Stealth SYN Scan with Host Discovery Disabled

For stealth and reliability, I ran a SYN scan (-sS) with host discovery turned off (-Pn) and used elevated privileges. This method is commonly used to evade basic detection by not completing TCP handshakes.

**Command used:** sudo nmap -sS -Pn testphp.vulnweb.com

Observed Behavior,

Target IP: 44.228.249.3

Host is live (latency ~0.30s)

Open Port: 80/tcp (HTTP)

Web Server: nginx 1.19.0

Hosted on AWS (ec2-44-228-249-3.us-west-2.compute.amazonaws.com)

Title: Home of Acunetix Art

(Refer to attached screenshots for raw outputs and verification.)



```
spectre@fsociety:~/Desktop/IshaSangpal$ sudo nmap -sS -Pn testphp.vulnweb.com
[sudo] password for spectre:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-03-21 09:26 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.31s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE
80/tcp    open  http
```

### **Impact:**

Identifying open ports and active services allows attackers to:

Enumerate running applications and their versions

Pinpoint outdated or misconfigured services

Begin planning targeted attacks (e.g., known CVEs for nginx)

### **Mitigation Recommendations:**

Restrict exposed ports: Use firewall rules to limit access to essential ports only.

Harden network perimeter: Disable or uninstall unused services.

Conduct routine exposure audits: Regular scans (internally and externally) help detect unintentional exposures.

Enable intrusion detection systems: Alert on unusual or aggressive scanning behavior.

## **b. Directory Brute Forcing**

**Severity:** Medium — CVSS 6.5 (Information Disclosure)

**Objective:** Discover hidden or unlisted directories on the target web application that could expose sensitive information or insecure endpoints.

## **Steps Performed:**

Tool Used: dirb

To perform the brute-force directory enumeration, I used Dirb v2.22, a simple and effective web content scanner. The scan was launched against the target URL:

**Command used:** dirb http://testphp.vulnweb.com/

Wordlist Source:

Dirb used the default wordlist located at:

/usr/share/dirb/wordlists/common.txt

Total of 4,612 words were tested against the base URL.

## **Scan Results:**

Several hidden or unlisted directories and files were discovered, including:

/admin/ — often a sensitive path linked to backend control panels

/cgi-bin/ — classic legacy path, returned 403 Forbidden, which still confirms its existence

/crossdomain.xml — may reveal security policy details for cross-origin access

/CVS/ — indicates a version control directory exposed on the web, including:

/CVS/Entries

/CVS/Repository

/CVS/Root

/favicon.ico — confirms web application branding

/images/ — typically static files; worth deeper review

/index.php — main web entry point

The scan ended with a warning:

FATAL: Too many errors connecting to host — likely due to temporary connection limits or rate limiting by the server.

```
spectre@fsociety:~/Desktop/IshaSangpal$ dirb http://testphp.vulnweb.com/

-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Fri Mar 21 17:51:22 2025
URL_BASE: http://testphp.vulnweb.com/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4612

---- Scanning URL: http://testphp.vulnweb.com/ ----
==> DIRECTORY: http://testphp.vulnweb.com/admin/
+ http://testphp.vulnweb.com/cgi-bin (CODE:403|SIZE:276)
+ http://testphp.vulnweb.com/cgi-bin/ (CODE:403|SIZE:276)
+ http://testphp.vulnweb.com/crossdomain.xml (CODE:200|SIZE:224)
==> DIRECTORY: http://testphp.vulnweb.com/CVS/
+ http://testphp.vulnweb.com/CVS/Entries (CODE:200|SIZE:1)
+ http://testphp.vulnweb.com/CVS/Repository (CODE:200|SIZE:8)
+ http://testphp.vulnweb.com/CVS/Root (CODE:200|SIZE:1)
+ http://testphp.vulnweb.com/favicon.ico (CODE:200|SIZE:894)
==> DIRECTORY: http://testphp.vulnweb.com/images/
+ http://testphp.vulnweb.com/index.php (CODE:200|SIZE:4958)

(!) FATAL: Too many errors connecting to host
(Possible cause: COULDNT CONNECT)

-----

END_TIME: Fri Mar 21 18:02:07 2025
DOWNLOADED: 2079 - FOUND: 8
```

**Impact:**

The discovery of paths like /admin/ and /CVS/ may lead to unauthorized access, configuration disclosure, or even source code leakage.

Forbidden directories (like /cgi-bin/) are still valuable findings since they confirm the presence of a resource, even if direct access is blocked.

**Mitigation Recommendations:**

Disable directory listing in the web server configuration to prevent exposure of unlisted paths.

Restrict access to sensitive directories (e.g., /admin/, /CVS/) using proper authentication and firewall rules.

Hide or rename sensitive paths and avoid using predictable names.

Use .htaccess or server-side rules to return generic error responses or redirect suspicious probing activity.

Monitor for enumeration attempts with rate-limiting and alerting mechanisms.

**c. Login Traffic Interception**

**Severity:** High — CVSS 7.5 (Credential Exposure)

**Objective:** Intercept login credentials transmitted in plaintext over HTTP by capturing network traffic during a login attempt.

**Steps Performed:**

Prepared Wireshark for Packet Capture

Launched Wireshark and selected the active network interface.

**Applied a capture filter to minimize noise:** host testphp.vulnweb.com

Started capturing packets just before attempting to log in.

Performed Login on Target Website

Navigated to the login section of <http://testphp.vulnweb.com/>

(http://testphp.vulnweb.com/login.php).

Entered test credentials in the login form (e.g., test:test) and submitted them.

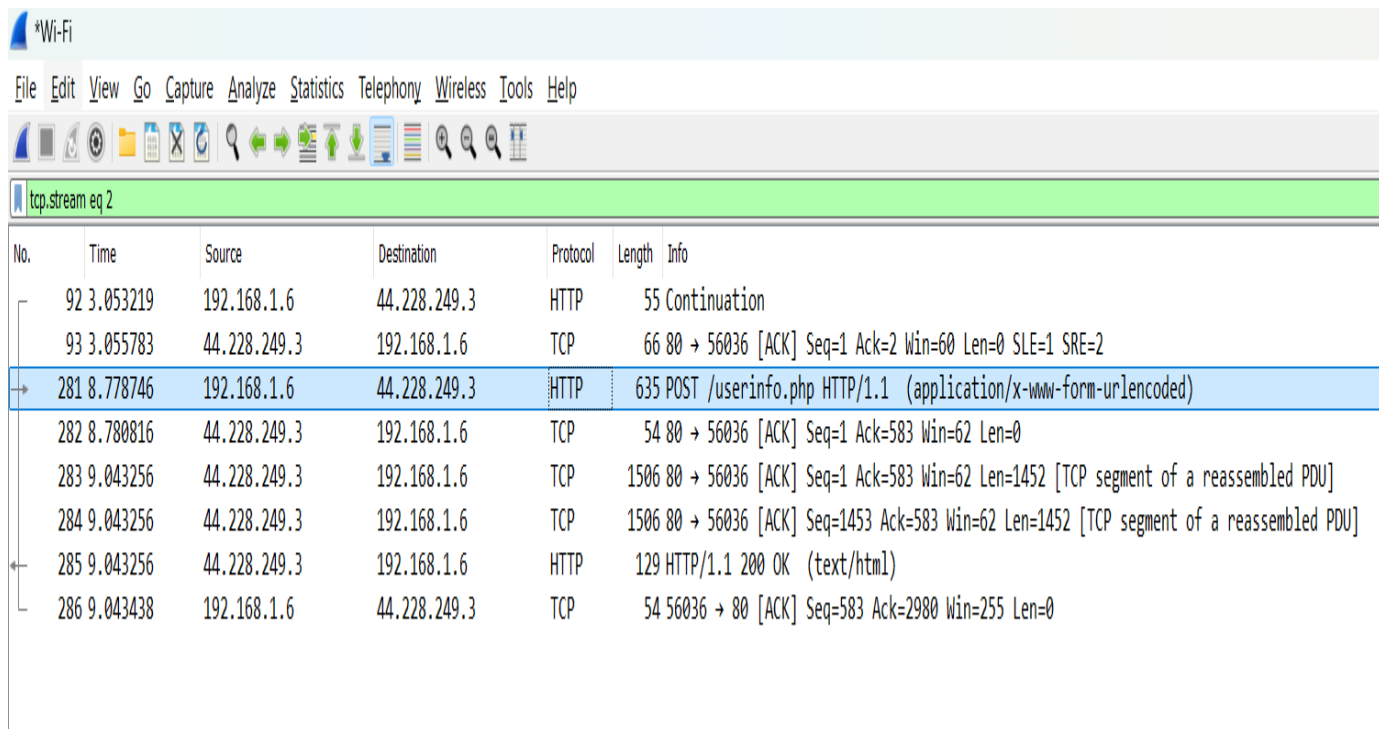
Since the site operates over plain HTTP (not HTTPS), the credentials were transmitted in cleartext.

Filtered and Analyzed Captured Traffic in Wireshark

After the login attempt, stopped the capture to begin analysis.

**Applied a display filter to locate the relevant HTTP POST request:**

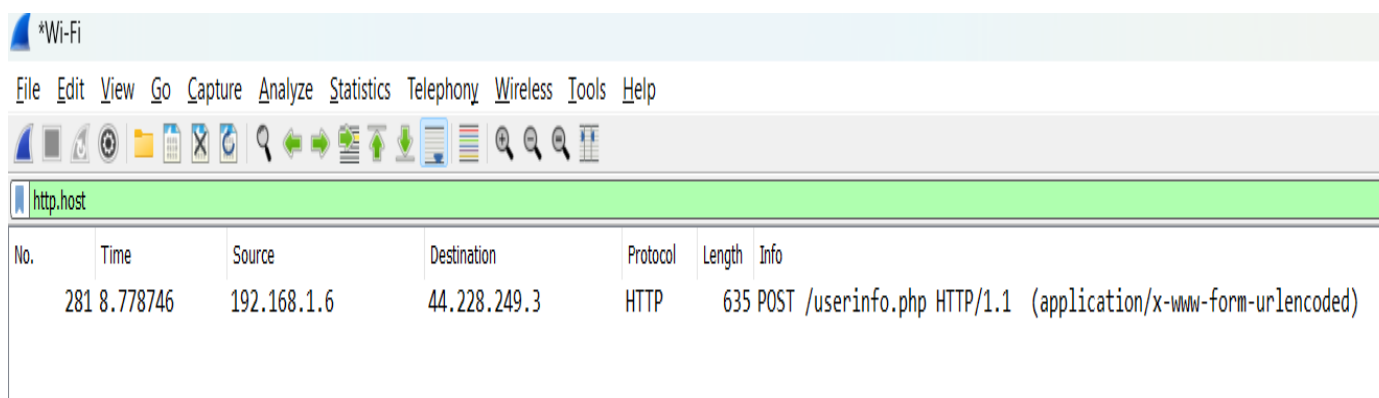
tcp.stream eq 2



The screenshot shows the Wireshark interface with the display filter 'tcp.stream eq 2' applied. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
92	3.053219	192.168.1.6	44.228.249.3	HTTP	55	Continuation
93	3.055783	44.228.249.3	192.168.1.6	TCP	66	80 → 56036 [ACK] Seq=1 Ack=2 Win=60 Len=0 SLE=1 SRE=2
281	8.778746	192.168.1.6	44.228.249.3	HTTP	635	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
282	8.780816	44.228.249.3	192.168.1.6	TCP	54	80 → 56036 [ACK] Seq=1 Ack=583 Win=62 Len=0
283	9.043256	44.228.249.3	192.168.1.6	TCP	1506	80 → 56036 [ACK] Seq=1 Ack=583 Win=62 Len=1452 [TCP segment of a reassembled PDU]
284	9.043256	44.228.249.3	192.168.1.6	TCP	1506	80 → 56036 [ACK] Seq=1453 Ack=583 Win=62 Len=1452 [TCP segment of a reassembled PDU]
285	9.043256	44.228.249.3	192.168.1.6	HTTP	129	HTTP/1.1 200 OK (text/html)
286	9.043438	192.168.1.6	44.228.249.3	TCP	54	56036 → 80 [ACK] Seq=583 Ack=2980 Win=255 Len=0

**Also applied this for quick HTTP POST identification: http.host**



The screenshot shows the Wireshark interface with the display filter 'http.host' applied. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
281	8.778746	192.168.1.6	44.228.249.3	HTTP	635	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)

Identified the POST request to: /userinfo.php

Protocol: HTTP/1.1

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

Host: testphp.vulnweb.com

Followed the TCP stream to inspect the raw HTTP request/response:

Found the credentials clearly embedded in the POST body:

**uname=test&pass=test**



Wireshark · Follow TCP Stream (tcp.stream eq 2) · Wi-Fi

```
.POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 20
Origin: http://testphp.vulnweb.com
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://testphp.vulnweb.com/login.php
Cookie: login=test%2Ftest
Upgrade-Insecure-Requests: 1
Priority: u=0, i

uname=test&pass=testHTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Fri, 21 Mar 2025 14:28:31 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Set-Cookie: login=test%2Ftest
Content-Encoding: gzip
```

## **d. Input Validation and Vulnerability Testing**

**Severity:** High — CVSS 7.5 (Credential Exposure)

**Objective:** Intercept login credentials transmitted in plaintext over HTTP by capturing network traffic during a login attempt.

### **Steps Performed:**

#### **1. Identified User Input Vectors**

Explored the authenticated area post-login to find input forms and dynamic parameters.

Located the login form (/login.php) and search functionality that reflected user input in the response.

Confirmed that user input was being rendered directly into HTML responses without apparent sanitization.

#### **2. SQL Injection Testing on Userinfo Form**

Used Burp Suite to intercept the login POST request.

Injected a basic payload to test for SQL injection:

Name: ' OR '1'='1

The payload successfully bypassed authentication, and was reflected on the web page.

This confirmed that the backend failed to properly sanitize SQL queries, making it susceptible to injection.

← → ↻ Not Secure testphp.vulnweb.com/userinfo.php

codecrafters-io/build-... TryHackMe | Red Team... TryHackMe | Advance... What is a Penetration ... Wayback Ma

**acunetix** **acuart**

TEST and Demonstration site for **Acunetix Web Vulnerability Scanner**

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#) [Logout test](#)

**search art**

[Browse categories](#)

[Browse artists](#)

[Your cart](#)

[Signup](#)

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

[Logout](#)

**Links**

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)

**1 (test)**

On this page you can visualize or edit you user information.

Name:	<input type="text" value="1"/>
Credit card number:	<input type="text" value="987555662"/>
E-Mail:	<input type="text" value="Lavender@gmail.com"/>
Phone number:	<input type="text" value="340964333"/>
Address:	<input type="text" value="St. Laurent"/>

You have 0 items in your cart. You visualize you cart [here](#).

[About Us](#) | [Privacy Policy](#) | [Contact Us](#) | ©2019 Acunetix Ltd

### 3. Reflected XSS Testing in Search Field

Navigated to a search form where user input was echoed in the page.

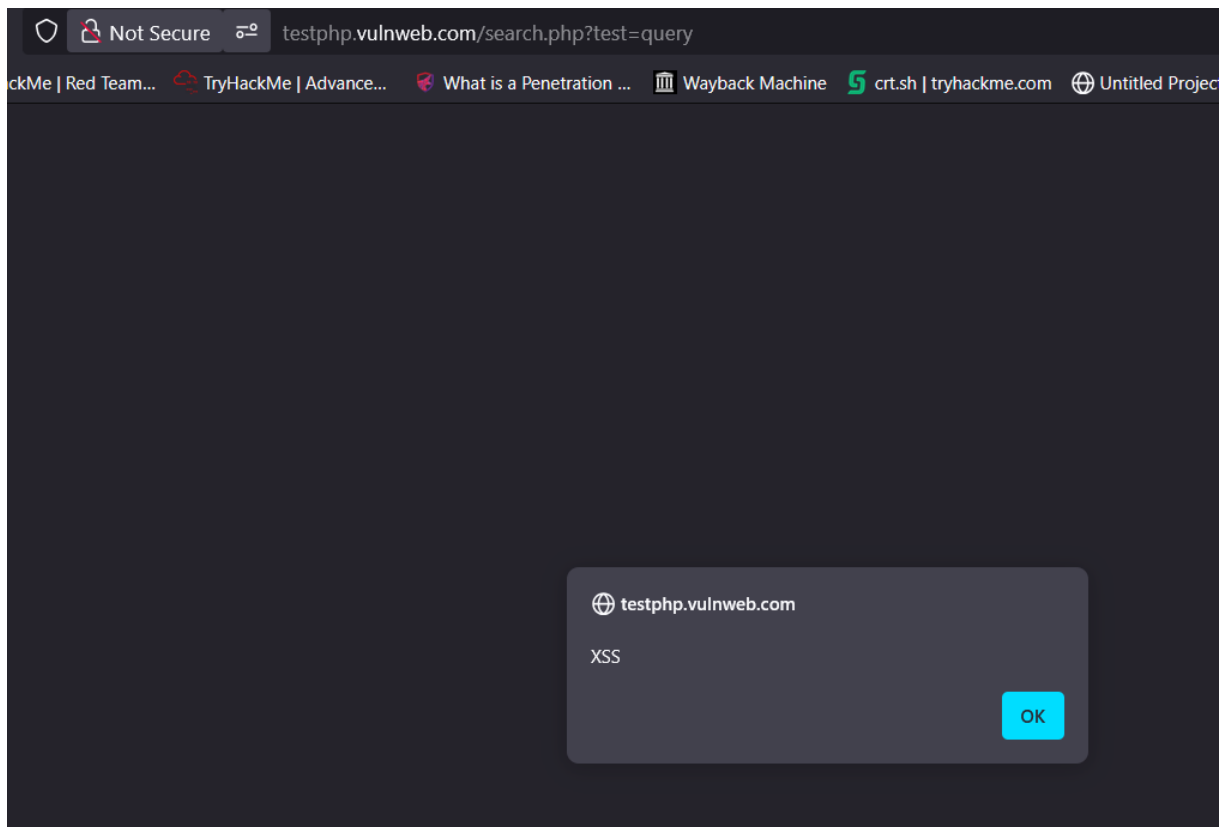
Injected a reflected XSS payload:

```
<script>alert('XSS')</script>
```

Upon submission, the script was executed in the browser, indicating no output encoding or input sanitization on the server side.

This proved the application was vulnerable to client-side script injection, potentially allowing for session hijacking or phishing attacks.





#### 4. Validated Vulnerabilities

SQL Injection was confirmed via successful login bypass using crafted payloads.

Reflected XSS was verified through live script execution in the browser.

No Web Application Firewall (WAF) or basic input validation appeared to be in place to block these malicious inputs.

## 6. Conclusion

This engagement against the hard-level vulnerable web application was a clear demonstration of how layered weaknesses—when overlooked—can lead to full system compromise. While each vulnerability on its own might seem manageable, the real danger came from how they could be chained together in a logical, methodical attack path.

Starting with basic **network reconnaissance**, open ports revealed the presence of a web server running over **unencrypted HTTP**, which was already a signal of poor security hygiene. From there, simple yet effective **directory brute-force enumeration** uncovered hidden and potentially sensitive endpoints—like admin logins and backup paths—that should never be publicly accessible.

One of the most critical points in the assessment was the **interception of login credentials** through plaintext traffic. The fact that valid usernames and passwords were being transmitted without encryption represents a high-severity flaw in any environment. It provided the attacker with immediate access to authenticated areas of the application—without the need for brute force, social engineering, or guessing.

Once inside, the vulnerabilities deepened. The application suffered from a lack of **input validation**, enabling both **SQL Injection** and **Reflected Cross-Site Scripting**. These issues weren't just hypothetical—they were exploited successfully to bypass authentication mechanisms and execute scripts directly within the browser context. These could easily be weaponized to extract sensitive data, hijack sessions, or pivot into further parts of a larger network if this system were deployed in production.

Importantly, no protective controls—such as Web Application Firewalls, rate-limiting, input sanitization, or even basic HTTPS—were observed throughout the assessment. This suggests a complete absence of secure coding practices or security architecture planning.

Overall, the test highlighted how real-world attackers think: not by looking for silver bullets, but by combining small misconfigurations and overlooked flaws into a functional exploit chain. From port 80 all the way to post-auth script injection, each step exploited a predictable gap—gaps that are preventable with proper security awareness, modern frameworks, and proactive defense-in-depth strategies.

If this were a live production system, the risk of compromise would be extremely high.

## **7. References**

Below are the references consulted throughout the course of identifying, exploiting, and understanding the vulnerabilities demonstrated in this lab:

OWASP Top 10 – Web Application Security Risks

<https://owasp.org/www-project-top-ten/>

Nmap Reference Guide

<https://nmap.org/book/man.html>

Wireshark User Guide

[https://www.wireshark.org/docs/wsug\\_html\\_chunked/](https://www.wireshark.org/docs/wsug_html_chunked/)

Exploit-DB & CVE Details – For checking service vulnerabilities.

<https://www.exploit-db.com/>

<https://www.cvedetails.com/>

Common Vulnerability Scoring System (CVSS) Calculator v3.1

<https://www.first.org/cvss/calculator/3.1>

HTTP Protocol – RFC 7230

<https://tools.ietf.org/html/rfc7230>

CVE Details – Vulnerability Database

<https://www.cvedetails.com/>

## **8.Resources Used**

Below is a summary of the tools, platforms, and learning resources that supported the completion of all beginner and intermediate level tasks:

### **Tools & Utilities**

Nmap – Network reconnaissance and port scanning

Dirb – Directory brute-forcing on web servers

Wireshark – Packet capturing and traffic analysis

### **Learning Platforms / Labs**

ShadowFox Cybersecurity Labs – Hands-on practice environment for each task

TryHackMe –Methodology reference

YouTube / Blogs – For walkthroughs on setting up tools like Nmap in practice

Portswigger Labs