Penetration Test Report

ZHIYUN Co., Ltd.

November 2, 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

A targeted penetration test was conducted against the Kioptrix Level 4 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 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. **User Enumeration via Samba**

   The Samba service disclosed valid system usernames, providing attackers with a list of potential targets

2. **SQL Injection (SQLi)**

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

3. **Plaintext Password Storage**

User passwords were stored in plaintext in the database, eliminating any encryption protection.

4. **MySQL Running as root**

   MySQL is running as root, enabling reliable local command execution and straightforward privilege escalation to root when combined with database or web access.

5. **Privilege Escalation**

   Through MySQL User-Defined Functions (UDF), system commands could be executed with root privileges.

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
```

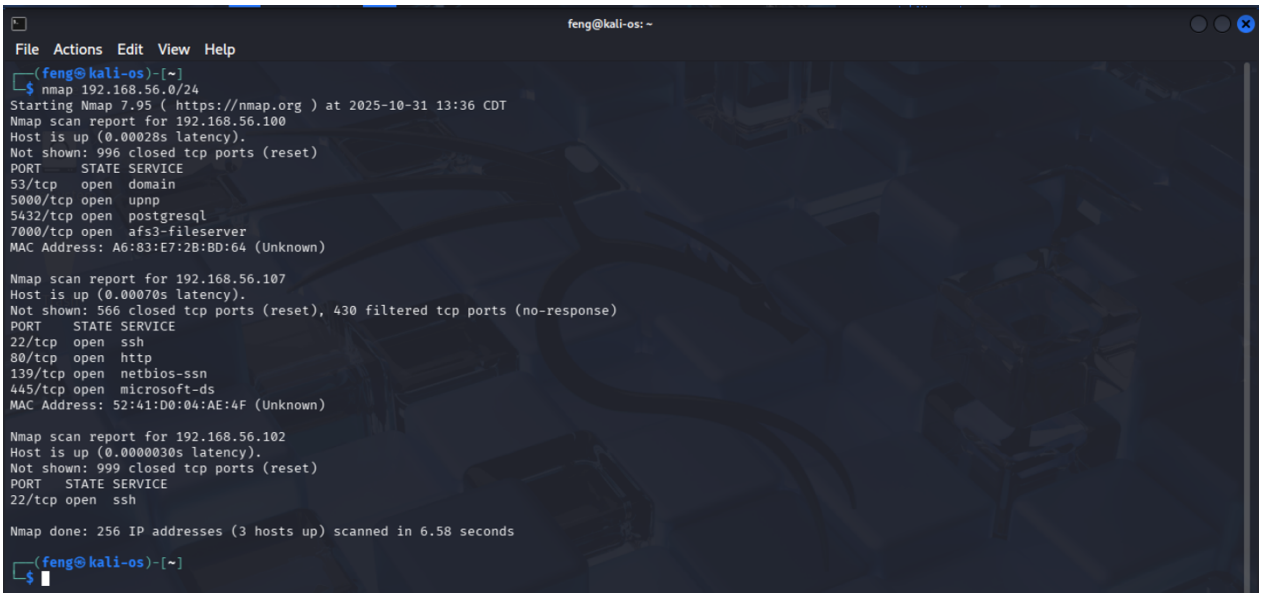

Figure 1 - nmap_scan_hosts

2. Discover open ports and services

```
nmap –sV –O 192.168.56.107 -p1-65535
```

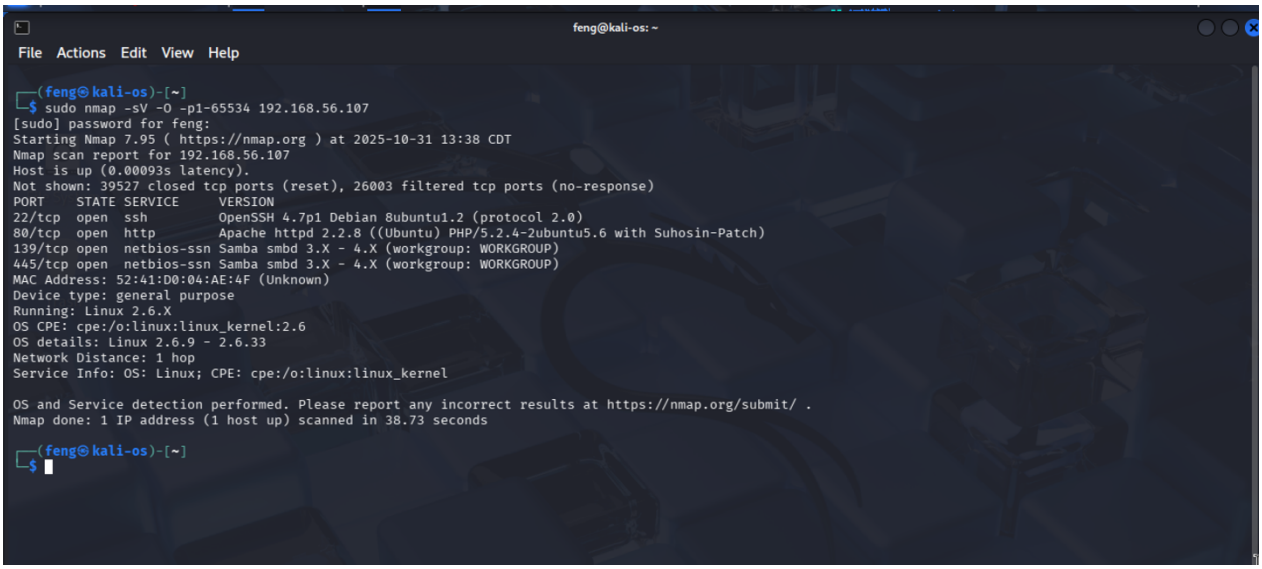

Figure 2 - nmap_full_192.168.56.107

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

```
nikto -h 192.168.56.107
```

The result indicates that an Apache Server is running.

Figure 3 - nikto_scan_192.168.56.107

```
dirb http://192.168.56.107
```

The following output indicates that we are detecting many URLs. So, there should be a website running on 192.168.56.107.

Figure 4 – dirb_scan_192.168.56.107

User Discovery via Enumeration

1. Scan the Samba service with enum4linux

```
enum4linux 192.168.56.107
```

From the image below, we can see some local users on the host, such as john, Robert and loneferret

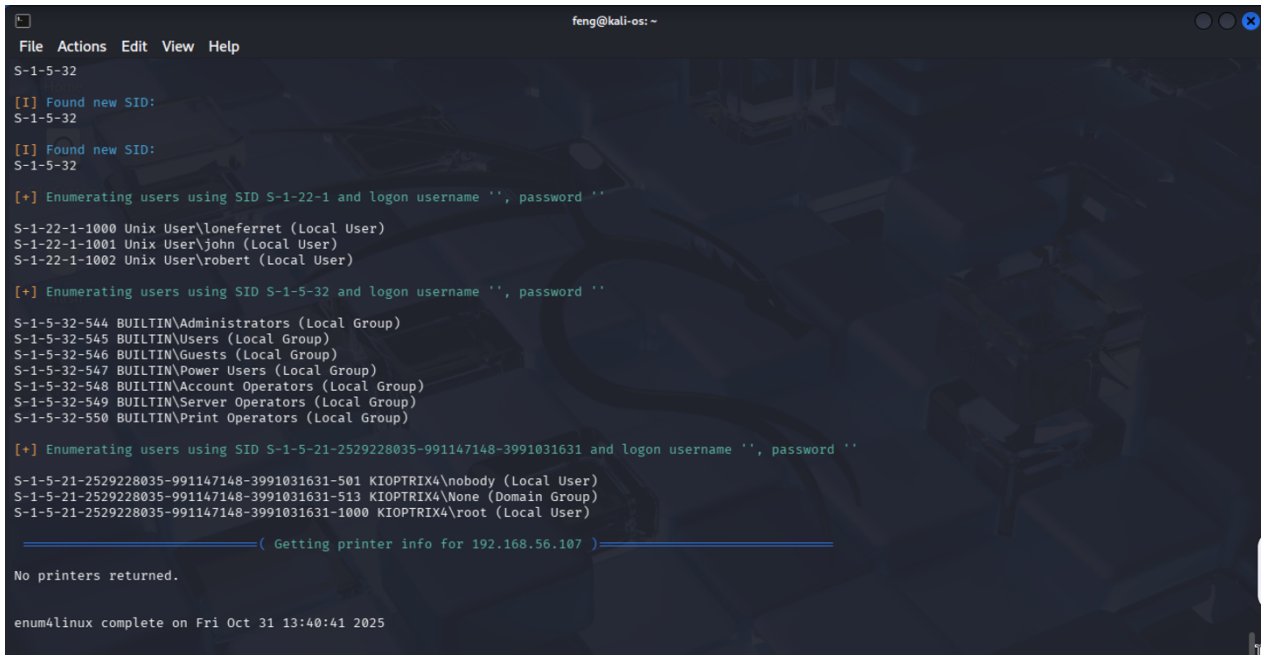

Figure 12 - scan_local_users

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://192.168.56.107/

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

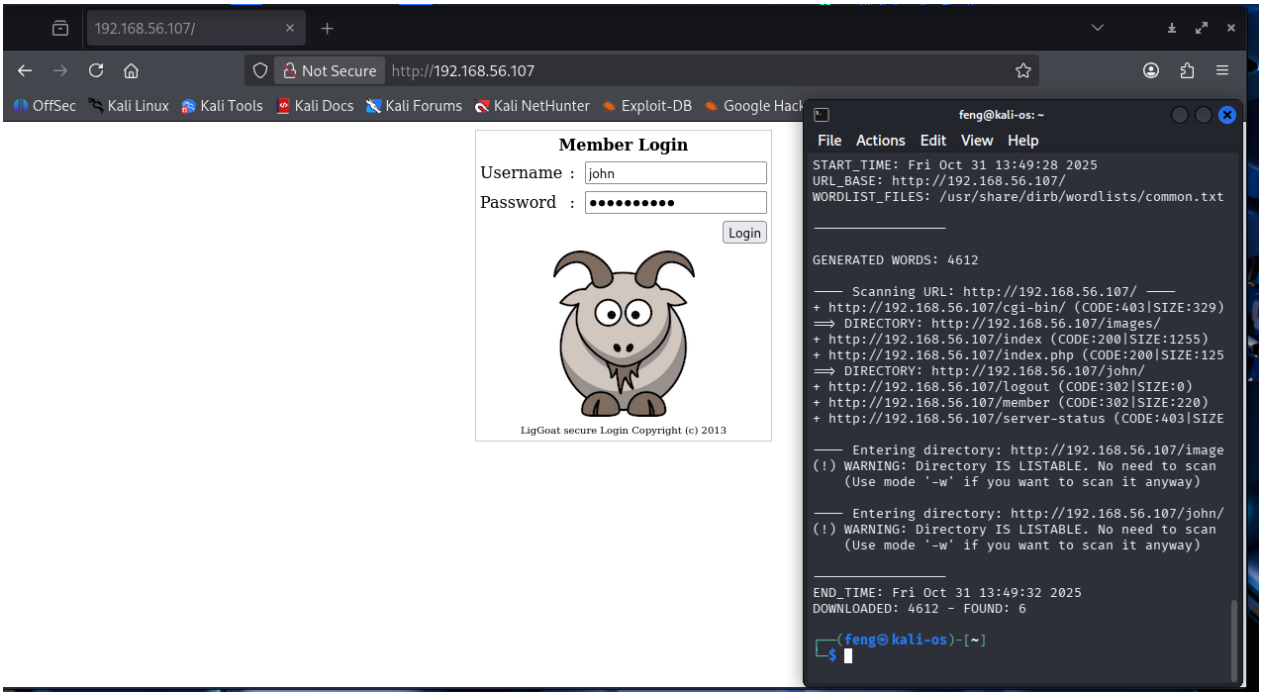

Figure 5 – firefox_open_website

Unfortunately, the following screenshot shows that we have logged in and gotten the John's password.

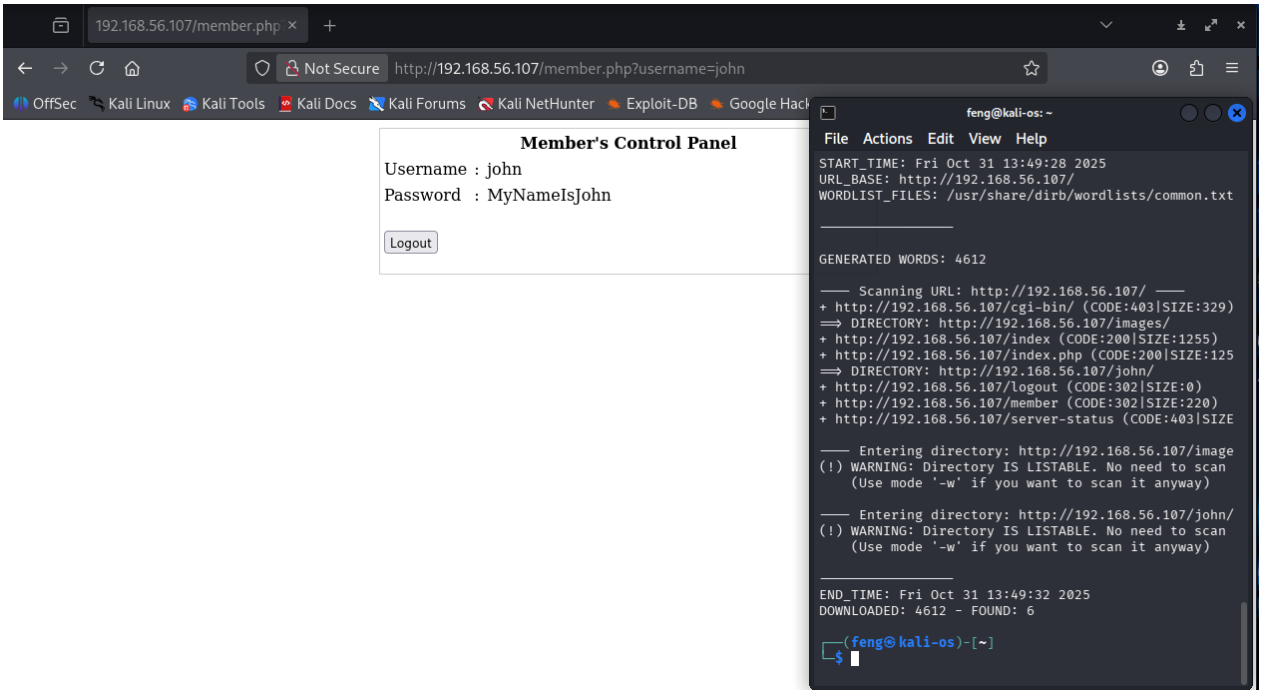

Figure 6 – website_logged_in

MySQL Running as root

1. Since we have John's password, let's attempt to log in via SSH.

```
ssh -oHostKevAlgorithms=+ssh-rsa john@192.168.56.107
```

As the image below shows, we are logged in as the user John, not as the root user.

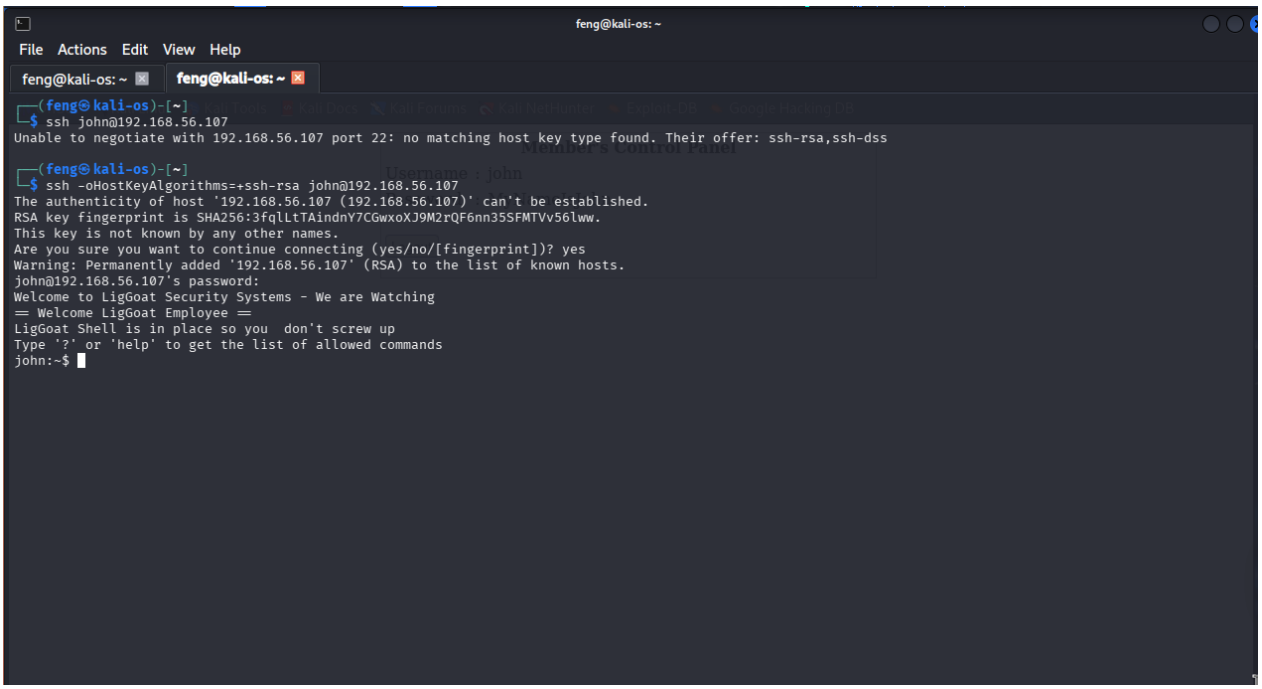

Figure 7 - ssh_logged_in

2. Our goal is to get root privileges, so we will look for processes running as root:

```
ps –eaf | grep root
```

In the list of processes run by root, we will find the MySQL process. If we can log into MySQL, we can escape to the shell.

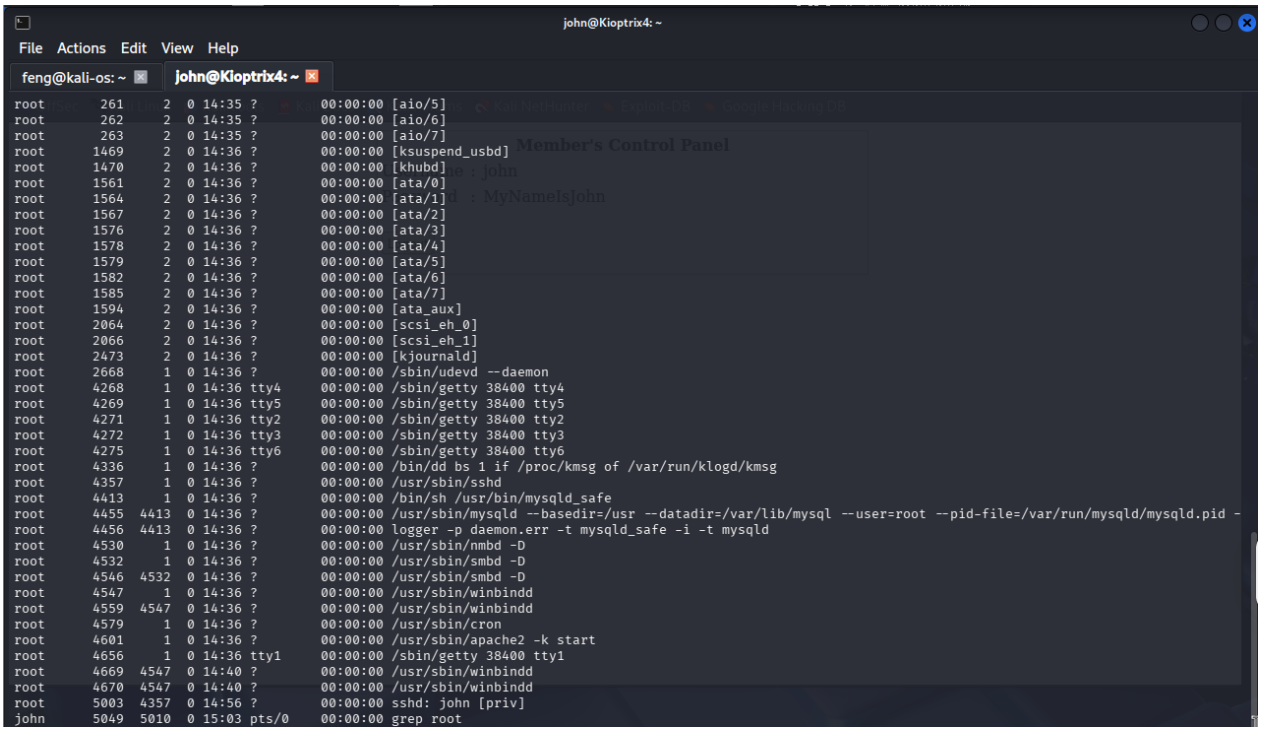

Figure 8 - ps_process_list

3. Locate the connection settings of the web application (located in /var/www) and get the connection information on the checklogin page.

```
head /var/www/checklogin.php
whereis lib_mysqludf_sys.so
```

We will find the username and password that are used to connect to MySQL. Additionally, it confirms that the **mysqludf** module is installed.

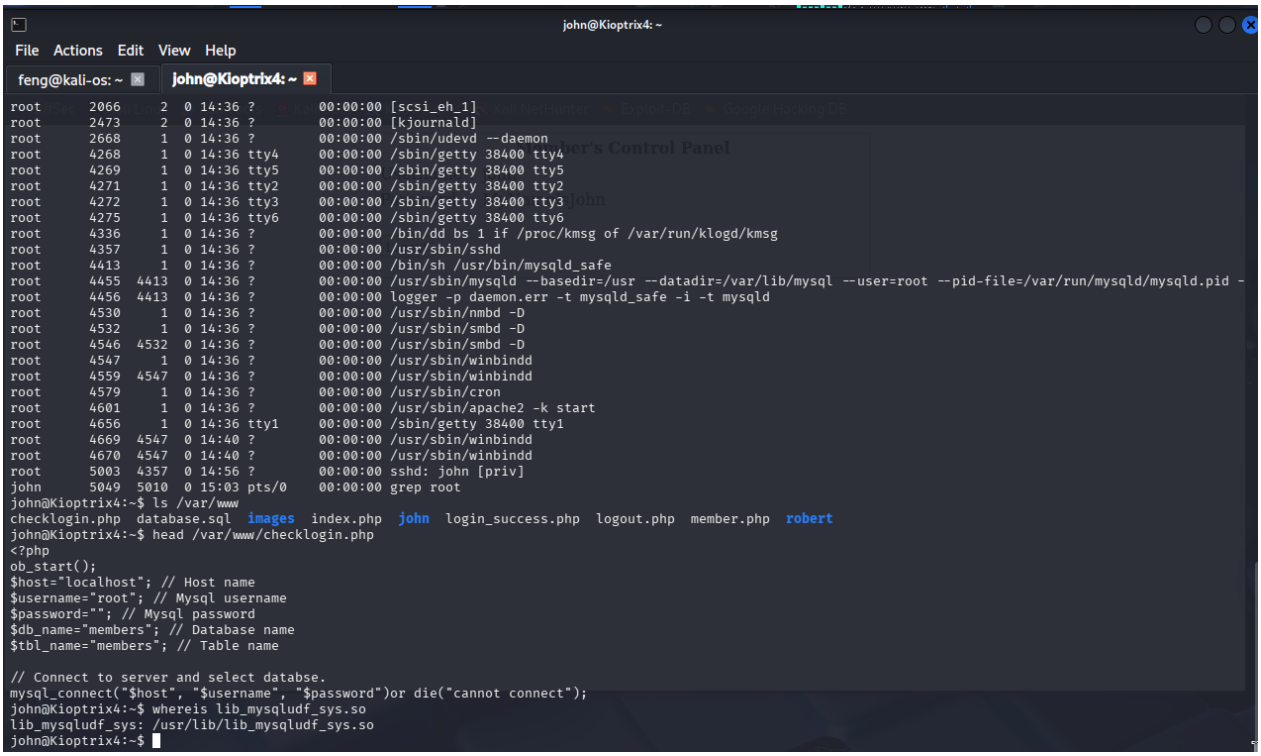

Figure 9 - mysql_user_passwd

Privilege Escalation

1. Log into MySQL and execute the command to add John to the admin group

```
mysql -u root
select sys_exec('usermod -a -G admin john');
```

Once the command executes successfully, John can run 'sudo su' to become root because the admin group can then run 'sudo'.

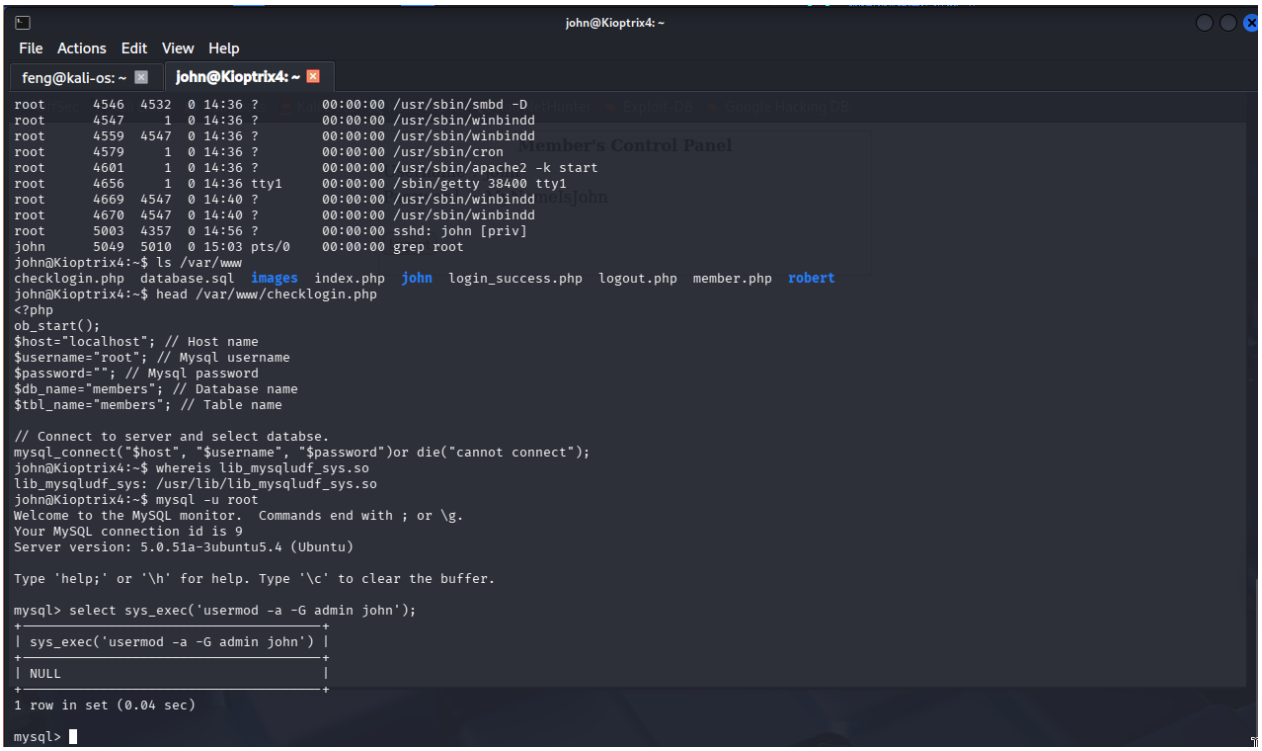

Figure 10 - Mysql_logged_in

2. Obtain the root privileges

```
sudo su
```

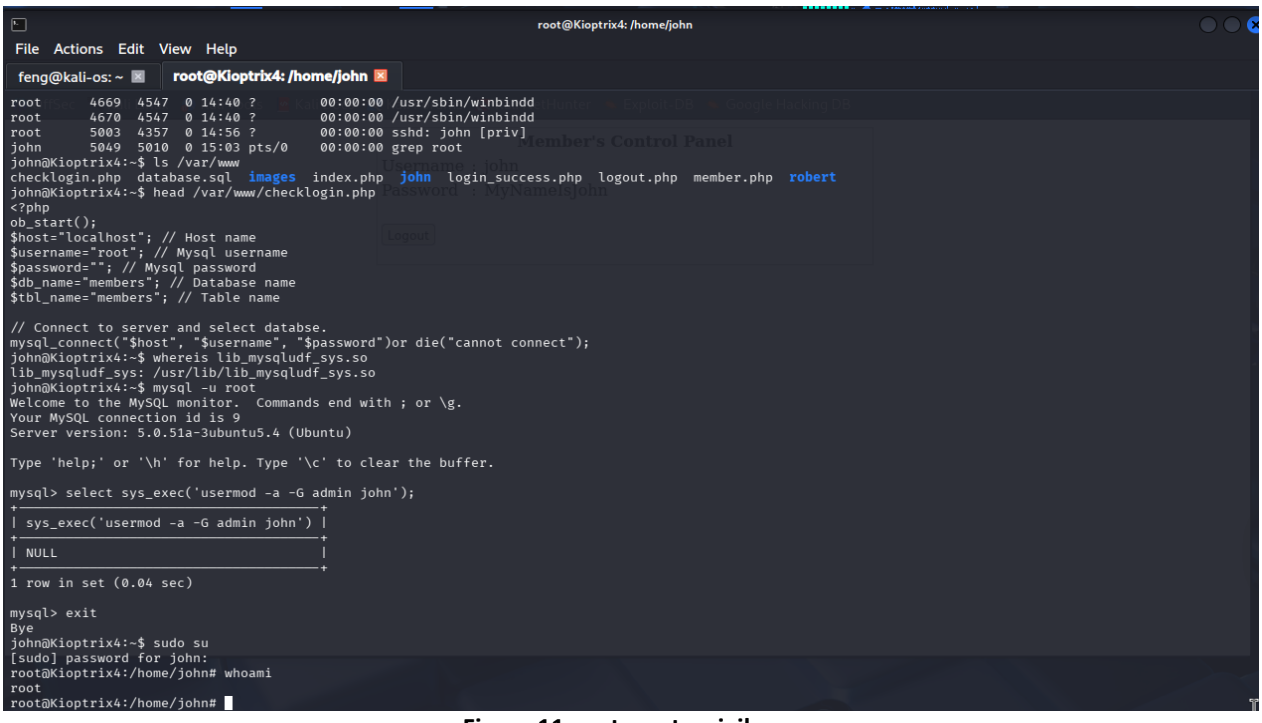

Figure 11 - get_root_privileges

Conclusion

The Kioptrix Level 4 system demonstrated multiple severe security vulnerabilities that enabled a complete compromise from external network access to root-level system control. The attack chain progressed through the following stages:

- Information disclosure via Samba enumeration
- SQL injection bypass of authentication controls
- Restricted shell escape
- Privilege escalation through misconfigured MySQL service

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

- Reconfigure MySQL to run under a dedicated, non-privileged service account instead of root.
- Remove or disable MySQL User-Defined Functions (UDF) to prevent abuse for arbitrary command execution.
- Implement parameterized queries throughout the web application to eliminate SQL injection vulnerabilities.
- Ensure that all web and application services operate under minimally privileged OS accounts with restricted file system access.

- Disable SMBv1/legacy Samba protocols and limit access to only trusted hosts.
- Enforce secure password storage (bcrypt/Argon2) and eliminate plaintext credentials across the environment.

Risk Rating

The overall risk identified in this assessment is evaluated as **High**, driven by the presence of multiple critical misconfigurations and application-layer vulnerabilities that provide a clear and reliable path to full system compromise.

A complete attack chain exists from the web application interface through credential exposure, restricted-shell escape, and ultimately root-level privilege escalation via MySQL running as the root user. Each vulnerability demonstrates high exploitability and high potential impact, and chaining these issues requires only moderate skill from the attacker.

As a result, it is reasonable to conclude that a malicious actor with similar capabilities could successfully execute an attack leading to:

- Unauthorized access to internal systems, databases, and sensitive information
- Compromise of valid user credentials, enabling lateral movement and persistent access
- Execution of arbitrary commands resulting in data exfiltration, service disruption, or full host takeover

Given the severity and exploitability of the identified weaknesses, immediate remediation and strengthened ongoing security governance practices are strongly recommended to reduce the organization’s exposure and prevent recurrence.

Appendix A: Vulnerability Detail and Mitigation

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*

Plaintext Password Storage

Rating: High

Description: User passwords are stored in plaintext without cryptographic hashing or encryption, violating security standards and compliance requirements.

Impact: Immediate compromise of all user credentials upon database access. Enables credential stuffing attacks across multiple services. Creates regulatory compliance violations (GDPR, HIPAA, PCI-DSS) and severe reputational damage.

Remediation: Immediately implement bcrypt, Argon2, or PBKDF2 password hashing with unique salts. Force password reset for all users. Never log or transmit passwords in plaintext. Consider implementing multi-factor authentication.

- OWASP Password Storage Cheat Sheet:*https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html*
- CWE-256: Plaintext Storage of a Password: *https://cwe.mitre.org/data/definitions/256.html*

MySQL Running as root

Rating: High

Description: MySQL database runs with root operating system privileges. When MySQL runs as root, any database compromise provides complete system control through User-Defined Functions (UDF).

Impact: Arbitrary command execution as root user. Complete system compromise including rootkit installation, backdoor creation, credential theft, and lateral movement to other network systems.

Remediation: Create dedicated mysql service account with minimal privileges. Update configuration to run as non-privileged user. Remove UDF capabilities unless required. Implement AppArmor/SELinux mandatory access controls. Restrict network access to MySQL.

- CIS MySQL Benchmark: *https://www.cisecurity.org/benchmark/mysql*
- *CWE-250: Execution with Unnecessary Privileges: https://cwe.mitre.org/data/definitions/250.html*

Samba User Enumeration

Rating: High

Description: Samba service allows anonymous enumeration of system users without authentication, providing attackers with valid usernames for targeted attacks.

Impact: Reduces complexity of brute-force attacks by providing valid username lists. Enables effective password spraying and credential stuffing. Provides intelligence for social engineering campaigns.

Remediation: Set restrict anonymous = 2 in smb.conf. Disable SMB v1 protocol. Implement network segmentation for file sharing. Enable authentication for all Samba operations. Monitor for enumeration attempts.

- *CWE-200: Information Exposure: https://cwe.mitre.org/data/definitions/200.html*

Appendix B: About the Team