### 1. Introduction

The constant-changing world of cybersecurity requires constant alertness and strategies to protect software systems from emerging exploits and attacks. With the rise of artificial intelligence—both as a tool for defense and as a weapon for attackers—the need for robust security measures has become even more critical. AI has amplified both the complexity and frequency of cyber threats, making regular penetration testing more essential than ever in maintaining strong cybersecurity defenses. Penetration testing plays a crucial role in this effort by providing organizations with deep knowledge of how robust and strong their security framework is. Also known as ethical hacking, penetration testing mimics real-world cyberattacks to evaluate how well a system can withstand such intrusions.

The reporter will design a network environment/topology to simulate a real network composed of various devices. Afterwards, the reporter will carry on the task of identifying the target machine (pWnOS) on the system and performing a penetration testing on it. At the end of the security testing and exploration, the aim will be to identity vulnerabilities in the network, explore the risks involved when a real attacker intrudes the system, and suggest potential security measures to prevent a potential attacker performing an attack.

### 1.1 Report Structure and Methodology Used

This report is structured to provide a clear overview of the penetration testing process conducted on the pWnOS system. While there are various recognized methodologies used in penetration testing—such as OWASP Testing Guide, NIST SP 800-115, and PTES (Penetration Testing Execution Standard)—this assessment adopts the **Open-Source Security Testing Methodology Manual (OSSTMM)** as its guiding framework. The following sections will outline the testing approach, present detailed findings from the assessment, and offer practical recommendations aimed at strengthening the overall security posture of the target system.

### 2. Summary and Recommendations

#### 2.1 Summary

The penetration testing assessment of the pWnOS environment uncovered multiple critical vulnerabilities that pose serious security threats to the system. These issues ranged from insufficient authentication mechanisms to the presence of outdated software components, both of which introduce exploitable attack vectors for unauthorized access and privilege escalation.

One of the most pressing concerns identified was the **weak authentication and password policies**. Specifically, the login and register page lacked security reinforcements as Multi-Factor Authentication (MFA), exposing it to brute-force attacks and unauthorized administrative access. This finding underscores a fundamental cybersecurity principle: access to administrative interfaces must be governed by strict authentication protocols, including MFA and the enforcement of complex, regularly rotated passwords. Failure to implement such controls significantly increases the attack surface and compromises system integrity.

A persistent **lack of security patching** and timely system updates further compounded the risk. In the context of modern threat landscapes—where zero-day vulnerabilities and advanced persistent threats (APTs) are increasingly common—delayed patch management represents a critical failure in maintaining a secure infrastructure. Industry best practices, such as those outlined by the Center for Internet Security (CIS) and the NIST Cybersecurity Framework, emphasize continuous vulnerability assessment and prompt remediation as core components of a resilient security program.

Furthermore, the system was found to be running outdated services with known vulnerabilities like **SQL injection and Local File Inclusion**. These unpatched components serve as low-hanging fruit for attackers, who can exploit publicly available exploits to gain footholds within the network or escalate privileges. This aligns with the Common Vulnerability Scoring System (CVSS), which classifies such flaws as high-risk due to their accessibility and potential for widespread impact.

#### 2.2 Recommendations

Following the results of the penetration testing exercise, the following security enhancements are recommended to strengthen the overall security level of PwnOS:

* **Implement Robust Access Control Measures and Multi-Factor Authentication (MFA):**  
   To mitigate the risk of unauthorized access to critical systems such as posing as admin to log into web interfaces, enforce strict role-based access controls (RBAC) and implement Multi-Factor Authentication (MFA) as it reduces the likelihood of successful brute-force or credential-stuffing attacks. Furthermore, all default credentials should be removed or disabled during initial setup, as demonstrated by the successful root login using preconfigured credentials during this pen-testing.
* **Adopt and Enforce a Strong Password Policy:**  
  Establish a comprehensive password policy that mandates passwords to be between 8 and 12 characters in length and include a mix of uppercase and lowercase letters, numbers, and special characters. These must be enforced upon creation, unlike the current state where the reporter could create passwords like “**password123**” and “**admin**” on the register page without special characters and still be accepted. Utilizing password managers and periodically updating credentials can further enhance security.
* **Conduct Routine Security Assessments and Vulnerability Scans:**  
   Schedule regular penetration testing, vulnerability assessments, and automated scans to proactively identify, assess, and remediate security weaknesses. Leveraging tools like Nessus, OpenVAS, or Nexpose can provide visibility into exploitable vulnerabilities before malicious actors can exploit them. Adopting a continuous security testing model aligns with industry best practices such as those advocated by OWASP and NIST.
* **Promote Security Awareness Through Comprehensive User Training:**  
   Conduct regular cybersecurity awareness training for all personnel. This should include best practices such as locking workstations, turning off computers when not in use, avoiding suspicious links or downloads, and recognizing social engineering tactics. A well-informed workforce serves as a crucial line of defense against common attack vectors like phishing and physical intrusion.

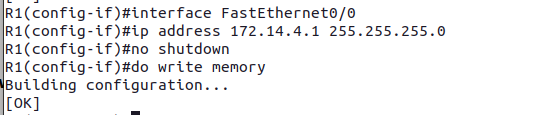
### 4. Network Environment Design

The Apporto environment was used to set up the network. Following the specification guide, a switch, a router (c7200), windows machine, kali machine and pWnOS were combined to form a network. The router was configured to act as the DHCP (Dynamic Host Configuration Protocol) server with the IP block 172.16.4.0/24. The following outlines how this was done:

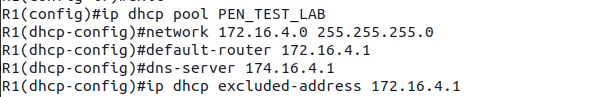
**Configuring the router terminal**



**Assigning the router interface**



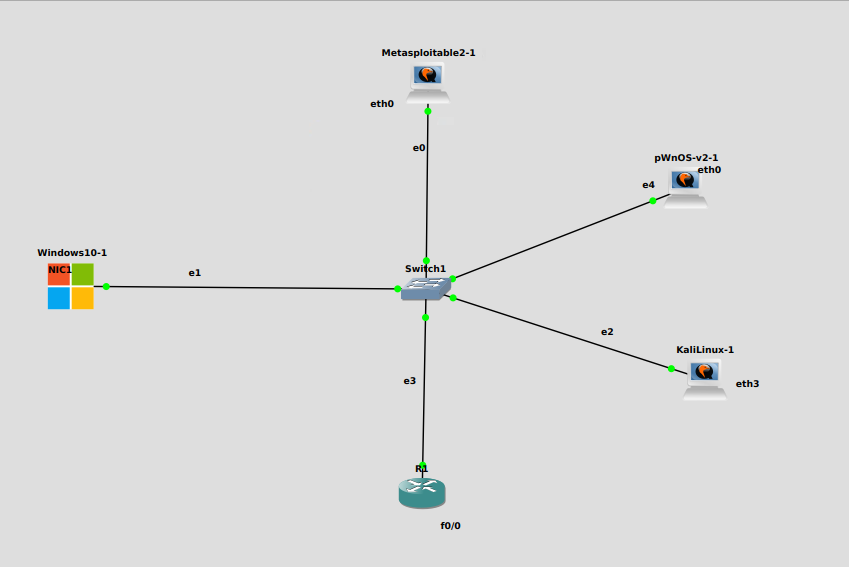
**Configuring the DCHP pool**



**Saving the configuration**



**Final network design and set up**

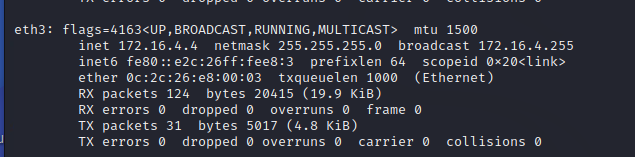


### 5. Information Gathering

After successfully setting up the network, I proceeded to configure the Kali Linux machine to be the machine used for the attack. As part of the setup process, a new user account was created and assigned the username **candice17**, which would be used to carry out further tasks and testing procedures on the system.

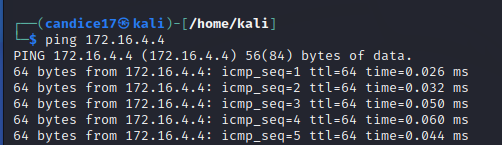
Before initiating any attacks or scans, I needed to confirm that the network infrastructure was properly configured and fully operational. To verify this, the reporter checked the IP address assigned to the Kali machine by executing the **ifconfig** command in the terminal. The output displayed the IP address **172.16.4.3**, which aligned perfectly with the expectations laid out in the network topology diagram above.

According to the diagram, the Kali machine was connected to the switch via the **eth3** network interface, and the IP address observed confirmed that the connection had been successfully established. This step was crucial in ensuring that both the network setup and routing were functioning correctly before moving on to any penetration testing or vulnerability scanning activities.

The results of **ifconfig:**  
   


To verify whether the DHCP server was operational, a simple ping test was conducted. The test involved sending ICMP echo requests to the router, which is typically responsible for handling DHCP services in a network. The router responded successfully to the ping, indicating that it is active and reachable on the network.

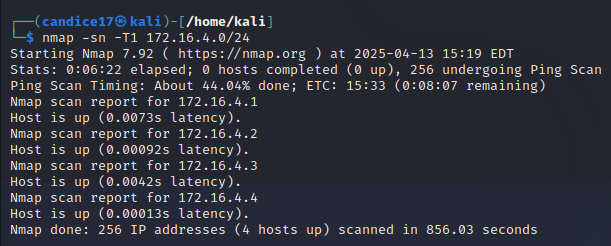
**Ping and test server**:



Having confirmed that the router is up, the next step in the process is to perform network discovery in order to identify all active machines within the environment and locate the intended target. To accomplish this, the **nmap** command in the table belowwas used, utilizing specific flags/options designed to probe the network for live hosts.

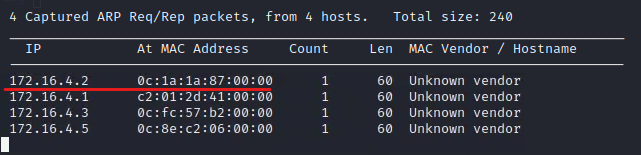
Table 1 (discovering all available IPs in the subnet)

|  |
| --- |
| nmap -sn -T1 172.16.4.0/24 |

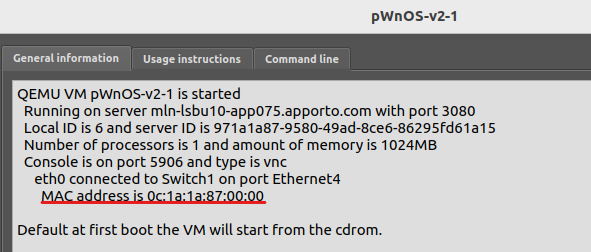
The command uses Nmap - a widely used network scanning utility- to perform a stealthy, low-profile ping scan across all 256 IP addresses in the 172.16.4.0/24 subnet, identifying which hosts are online without scanning for open ports. The flags –**sn** indicates that the scan type should not scan any open ports when finding which hosts are up. The -T1 timing template slows down the scan to reduce detection by intrusion detection systems and firewalls. The figure below shows the result after executing the command above:  
   


The results took about 14 minutes (856.03 seconds) which is relatively slow and prevents detection.

However, to know which of these is the target VM, **sudo netdiscover -r 172.16.4.0/24**, was used which shows the mac address of each IP. This can then be compared to each device's mac in the network.



Comparing each of the mac addresses to the node information of pWnOS in the network, the reporter discovered that the target is VM with IP 172.16.4.2 as its mac address matches that of the pWnOS seen below:



With the initial discovery phase complete, the next logical step was to begin gathering detailed information about the target machine – pNwOS , while aiming to perform this in a stealthy and discreet manner, minimizing the chances of detection by firewalls, intrusion detection systems (IDS), or any other network security mechanisms.

To accomplish this, nmap command with -**sS** (scan and be stealthy) flag in the table below was used. This initiates a **TCP SYN scan**—commonly referred to as a "half-open" scan. This type of scan is designed to be less detectable, as it does not complete the full TCP handshake. Instead, it sends a SYN packet to the target and waits for a response. If a SYN-ACK is received, the port is considered open, but the connection is never fully established, reducing the likelihood of the scan being logged or raising alarms.

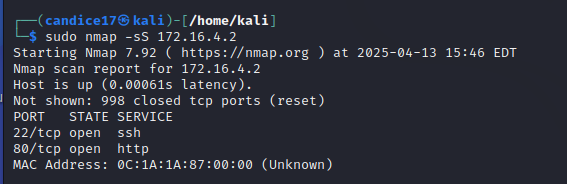
This approach allows the reporter to map open ports and services on the target host while significantly lowering the risk of triggering defensive security systems. It’s a subtle but highly effective method for initial information gathering without alerting the target.

Table 2 (maintaining low profile in information gathering)

|  |
| --- |
| sudo nmap -sS 172.16.4.2 |

This command revealed the following opened ports:

1. Port 80
2. Port 22

  
  
Additionally, to get more information than just opened ports, an aggressive scan was carried out using the command below:

|  |
| --- |
| sudo nmap -A -D RND:70 172.16.4.2 |

**-A**: This flag/option enables aggressive scanning. It tells Nmap to perform multiple types of scans in one go. Specifically, it includes:

* + OS detection
  + Version detection
  + Script scanning
  + Traceroute

**-D RND:70**: This is the decoy option. It tells Nmap to generate and use 70 random decoy IP addresses during the scan. The purpose of this is to **obfuscate the source** of the scan. When the target or a firewall inspects the incoming traffic, it will see requests coming from multiple IP addresses (including fake ones), making it more difficult to identify the real scanning machine.

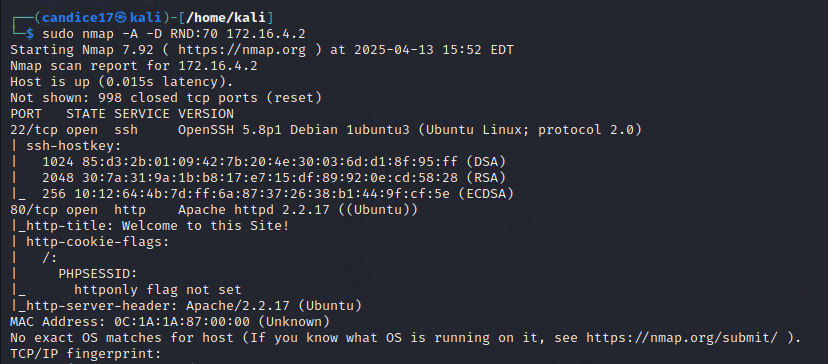
The result of this is seen in the figure below and summarized using the table :  
  
 

Table 3 (aggresive information)

|  |  |
| --- | --- |
| Target VM Details : pWnOS | |
| IP Address | 172.16.4.2 |
| SSH Service | OpenSSH 5.8p1 Debian 1ubuntu3 (Ubuntu Linux; protocol 2.0) |
| SSH Host Keys | 1. DSA: 85:d3:2b:01:09:42:7b:20:4e:30:03:6d:d1:8f:95:ff 2. RSA: 30:7a:31:98:17:e7:15:df:89:92:0e:cd:58:28 3. ECDSA: 10:12:64:4b:7d:ff:6a:87:37:26:38:b1:44:9f:cf:5e |
| HTTP Service | Apache httpd 2.2.17 ((Ubuntu)) |
| Host status | |  | | --- | |  |  |  | | --- | | Host is up (0.015s latency) | |
| MAC Address | 0C:1A:1A:87:00:00 (Unknown) |
| Opened Ports | 22 (SSH), 80 (HTTP) |

### 6. Scanning and Mapping

Ports 80 and 22 were discovered to be open on the target device during the information gathering process. It is now time to explore these ports.

**Port 22 scan**

Port 22 is associated with the Secure Shell (SSH) protocol, which is commonly used for secure remote access to systems over a network. This indicates that the user may have remotely connected to their organization’s infrastructure to perform administrative tasks, such as managing servers or performing system maintenance.

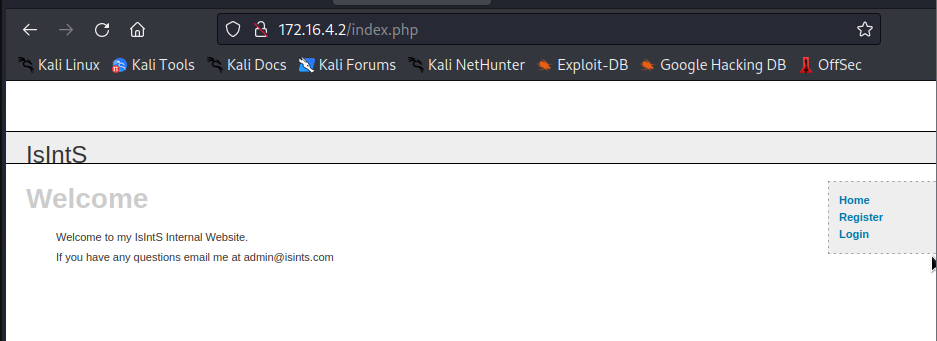
Additionally, SSH can be utilized for network resource sharing, including file transfers, printer access, or even fax communication, all while maintaining encrypted communication. To the reporter, the presence of an open SSH port could present a valuable entry point, especially if weak authentication methods or outdated SSH configurations are in place. Understanding that SSH is active on this port helps the reporter identify possible attack vectors, such as brute-force login attempts or exploiting known vulnerabilities in the SSH service.

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

During the reconnaissance (information gathering) stage, a scan of the target system revealed that **port 80 (HTTP)** was open, indicating the presence of a web server. To further investigate, I accessed the service using **Firefox on Kali Linux** and observed the following:

* **Homepage Content**:  
  The web application presented a simple interface with navigation options:
  + **Home**
  + **Register**
  + **Login**

No additional details about the service or its purpose were immediately visible, suggesting it might be a custom or internal application.

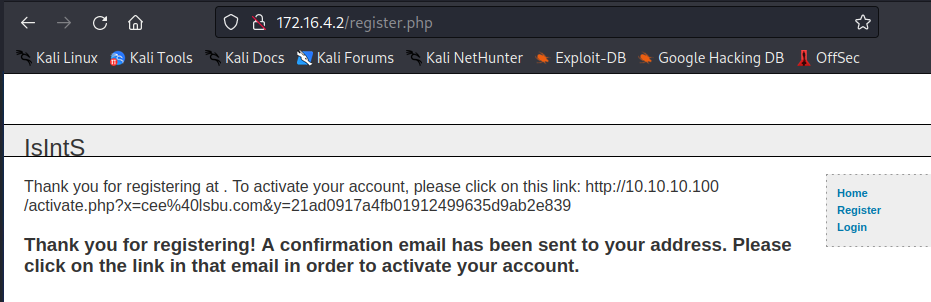


**2. User Registration Attempt**

To assess the functionality of the web application, I attempted to register a new account with the following credentials:

* **Email:** [cee@lsbu.com](mailto:redin@yahoo.com)
* **Password:** password123

The response is seen below:



**Observations:**

1. **Registration Response**:  
   After submitting the registration form, the server responded with a confirmation message:

*"Thank you for registering! A confirmation email has been sent to your address. Please click on the link in that email in order to activate your account."*

* 1. **Implication**: The application follows a standard email-based account activation process.

1. **Activation Link Structure**:  
   Despite no actual email being received (likely due to the target being an internal system), the next screen displayed an **activation link** in the following format:

[http://10.10.10.100/activate.php?x=cee%40lsbu.com&y=](http://10.10.10.100/activate.php?x=cee%40lsbu.com&y=21ad091a4fb0101230062ab23839) 21Ad091a4fB010f230062eb2b3839

* 1. **Breakdown of the URL**:
     1. x=cee%40lsbu.com: The encoded registered email (cee@lsbu.com).
     2. y=21Ad091a4fB010f230062eb2b3839: A 32-character hexadecimal string, likely an **MD5 hash** of a token or activation key.

1. **Potential Security Concerns**:
   1. **No HTTPS**: The link uses **HTTP**, meaning credentials and tokens are transmitted in **plaintext**, making them susceptible to interception.
   2. **Predictable Token?** If the activation token is generated predictably (e.g., based on email or timestamp), it could be brute forced.
   3. **Lack of Rate Limiting**: The application does not restrict repeated activation attempts; hence, the reporter could exploit this to enumerate valid accounts.

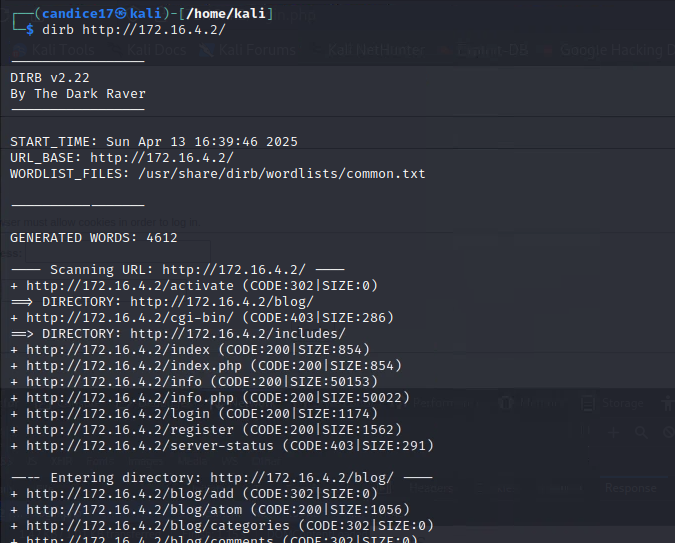
Moving on, to explore other web pages, the command below was used:

|  |
| --- |
| dirb http://172.16.4.2/. |

dirb is used to **brute-force directories and files** on a web server using a built-in or custom wordlist.

The command will:

1. **Connect to** the IP 172.16.4.2 via HTTP.
2. **Try to discover hidden files/folders** (like /admin, /login, /config.php, etc.).
3. **Use a wordlist** to try different URL paths (unless a custom one is specified, it uses its default list).

The result of executing the command is seen below:  


This DIRB scan of <http://172.16.4.2/> showed several important findings about the web server's structure and potential vulnerabilities. The scan uncovered accessible directories like /blog/ and /includes/, as well as key pages such as **index.php, login.php,** and **register.php**, which are common entry points for web applications.

Notably, the **/activate** endpoint returned a **302 redirect**, suggesting it may be part of an account activation flow, while **/cgi-bin/** and **/server-status** returned **403 Forbidden**, indicating restricted access that could still be probed for misconfigurations.

The presence of **info.php** (returning a **200 OK**) is revealing, as it likely exposes PHP configuration details via **phpinfo()**, leaking sensitive server information that I could use to refine exploits.

The **/blog/** subdirectory contains dynamic endpoints like **/blog/add** (which redirects) and **/blog/comments** (returning a **302**), hinting at a content management system (CMS) or custom blog platform. These endpoints could be tested for **injection flaws** (SQLi, XSS) or **authentication bypasses**.

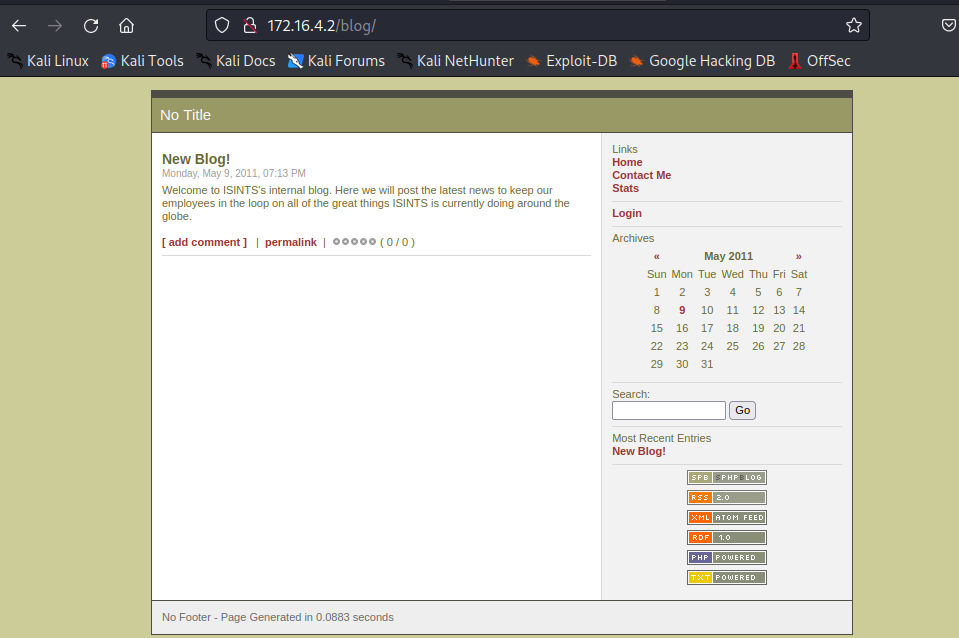
The combination of exposed admin interfaces (/login, /register), debug pages (info.php), and unsecured directories suggests the server may lack hardening.

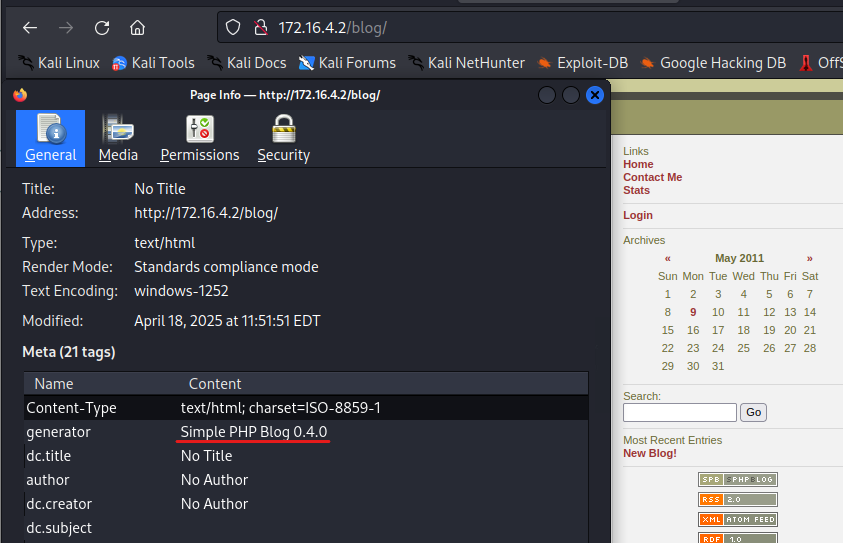
The next phase, enumeration would include manual testing of these endpoints and discover which ones are likely to grant access.

### 7. Enumeration

This phase of penetration testing process focuses on actively interacting with discovered services and systems in order to extract detailed information. After completing the initial scanning and mapping stages, the goal here is to go beyond identifying open ports and services by diving deeper into what those services reveal. This includes gathering intelligence about user accounts, shared resources, file system structures, system configurations, and other potentially sensitive or useful data. By actively querying these systems, the reporter can build a more comprehensive understanding of the environment, which is essential for attempting to gain access exploitation.

From the previous phase (scanning and mapping), it was discovered that the webapp had some paths that could be explored. This will be the focus to find any vulnerabilities.

Exploring the blog page (http://172.16.4.2/blog):  
  


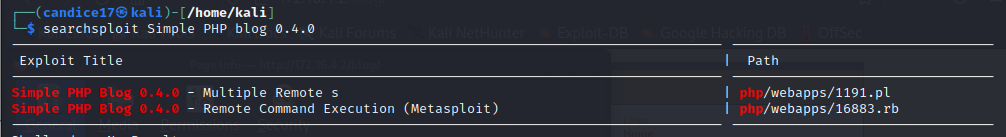
The homepage indicates it was an internal blog for a company called ISINTS’s. To search for vulnerabilities, the reporter checked the page source using the inspect tab but that did not provide any information. The next approach was to inspect the website’s metadata. To achieve this, the reporter clicked on the padlock icon next to the address bar and navigated to “more information” which showed the window below:  


After investigating all the tabs, the General tab has a generator showing that the blog was built using **Simple PHP blog 0.4.0.** This is useful as it enabled the reporter to check for any known vulnerabilities associated with the version of PHP. To check this, the command below was used:

|  |
| --- |
| Searchsploit Simple PHP blog 0.4.0 |

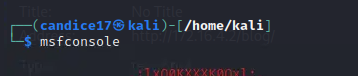
This command searches the Exploit Database (Exploit-DB) for known vulnerabilities in Simple PHP Blog version 0.4.0 by scanning the database for publicly disclosed exploits, PoCs (Proof of Concepts), or vulnerabilities affecting this specific blog software version.

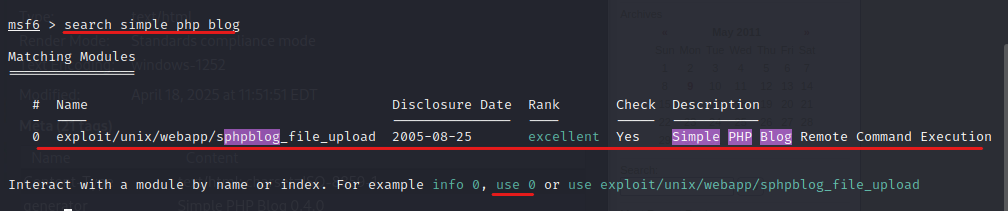
The results after the execution is seen below:



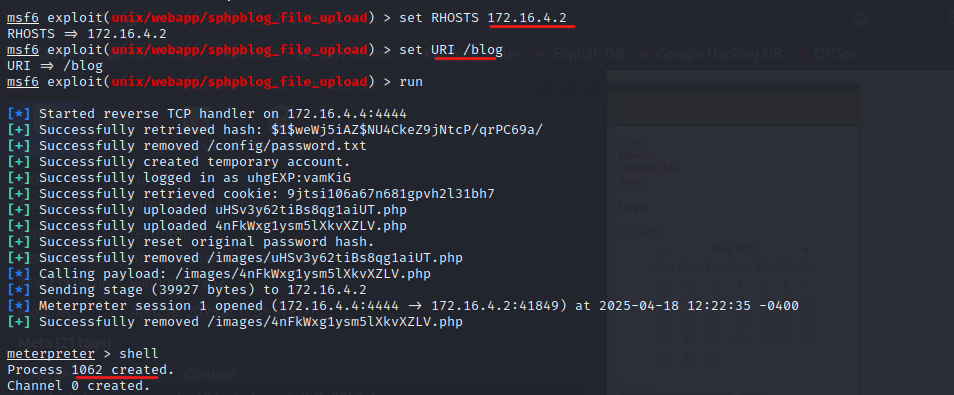
From the image above, it shows that the blog is exploitable by the Remote Command Execution (RCE) using metasploit.

The reporter launched the msfconsole in kali as seen below:



The available modules related to Simple PHP blog 0.4.0 was searched and it revealed the module seen below:  


The default payload was selected for the module using the command (use 0). Afterwards, the parameters were configured:

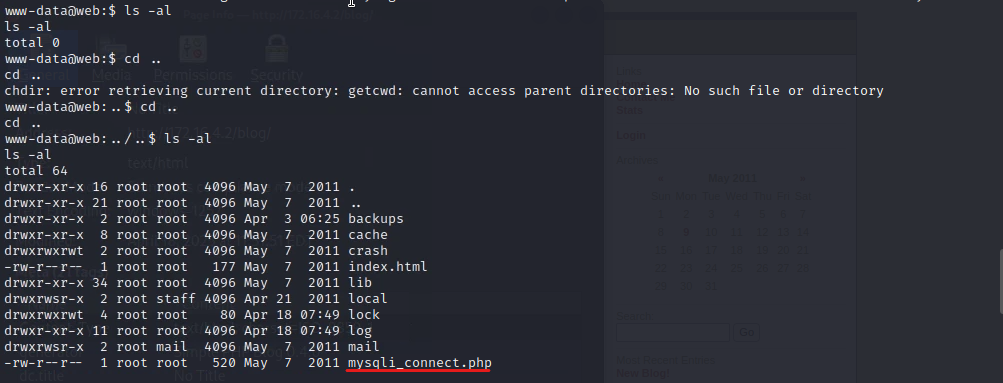
  
RHOSTS **(Remote Hosts)**

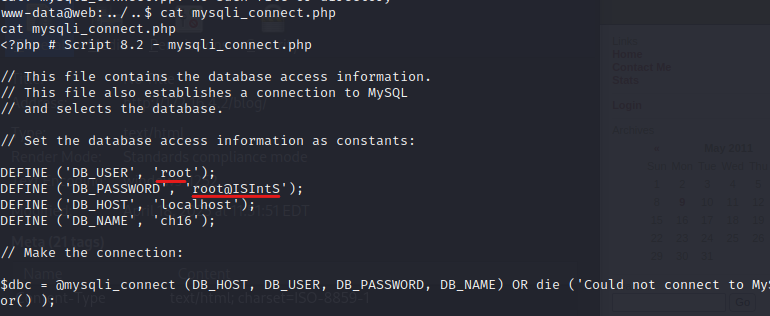
* **Purpose**: Specifies the **target IP address** (pWnOS machine).
* **Why it matters**: Directs the exploit to attack 172.16.4.2 (the vulnerable server)

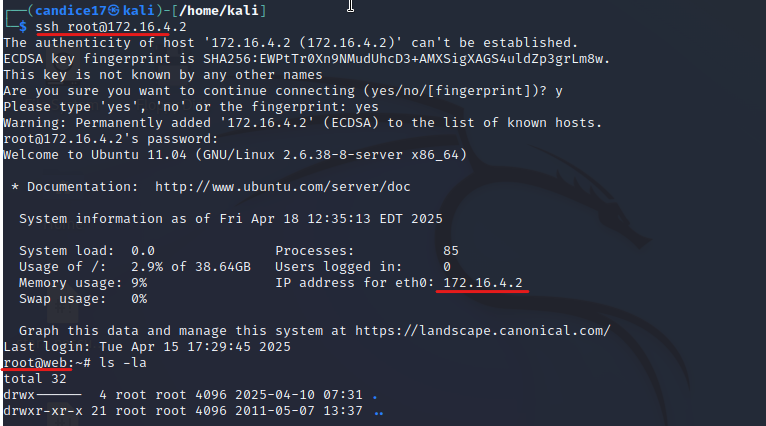
URI **(Uniform Resource Identifier)**

* **Purpose**: Defines the **path to the vulnerable web application** ( /blog).
* **Why it matters**: Ensures the exploit reaches the correct endpoint (e.g., <http://172.16.4.2/blog>).

After the shell started, the reporter started to explore the contents of the current user’s directory (www-data@web) and eventually saw a file named mysqli\_connect.php which looked suspicious.



The reporter used the “cat” command to use the contents of the file which instantly revealed the root user credentials as seen below:  


To verify this, a new terminal was launched, and the reporter initiated an SSH (Secure Shell) connection to login as the root use with the discovered credentials which worked successfully.  
  
The reporter observed that root-level access to the target system (172.16.4.2) could be obtained with unexpected ease, even without exploiting the publicly exposed login or registration forms on the web application. This finding suggests significant misconfigurations or unpatched vulnerabilities in the system’s authentication mechanisms. The ease of exploitation underscores critical weaknesses in the system’s access controls, which could be leveraged by malicious actors to gain unauthorized administrative privileges with minimal effort.

To validate the initial results, the tester pursued an alternative attack vector to confirm the system’s susceptibility.

### 8. Gaining Access (Alternative)

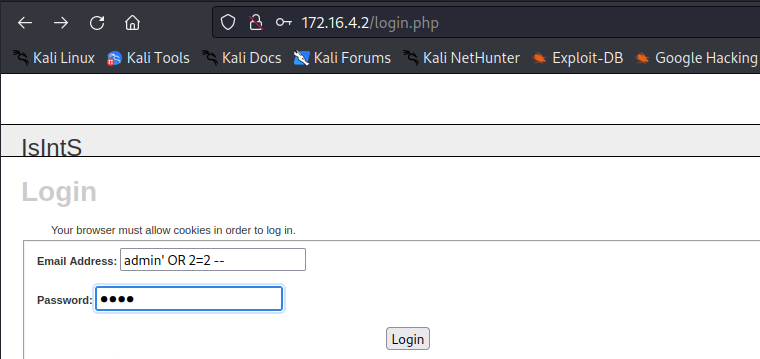
Following the enumeration phase, the reporter revisited the login page to test for SQL Injection (SQLi) vulnerabilities — a widespread security flaw where unsanitized input allows attackers to inject malicious SQL queries into the backend database. Rather than extracting data, the initial goal here was to **bypass authentication** and gain unauthorized access to the application.

A common method for this is the **Boolean-based SQL injection**, which manipulates the logic of the SQL query using always-true conditions. For example, a standard login query might look like:

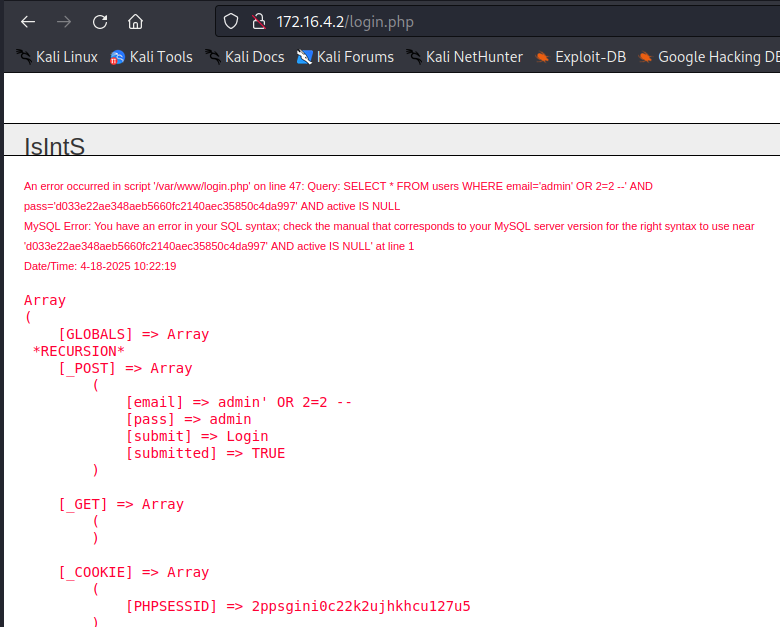
**SELECT \* FROM users WHERE username = '[input]' AND password = '[input]';**

By injecting a payload such as ' OR 1=1 -- into the **username** field, the SQL statement transforms into:

**SELECT \* FROM users WHERE username = '' OR 1=1 --' AND password = '[ignored]';**

This effectively bypasses the password check and causes the query to always return a valid result. With this, the reporter entered the username field of the login page as **admin’ OR 2=2 –** and an arbitrary password “**admin**”, as shown below:  
  


After clicking on the login button, the result affirmed the suspicion of presence of the SQL injection vulnerability. This is seen below:



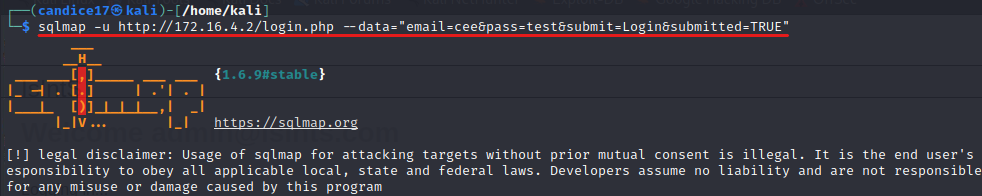
This **SQL** error page displayed after submitting the input admin' OR 2=2 -- in the username field reconfirms the presence of a **SQL Injection (SQLi) vulnerability**. Specifically, it shows that user input is being directly concatenated into the SQL query without proper sanitization or parameterization. The error message discloses the raw SQL query being executed, which includes the injected payload, and this indicates poor handling of user input and lack of input validation. This vulnerability not only confirms that the backend is susceptible to SQLi, but also potentially exposes sensitive database structure and logic, which could be further exploited for unauthorized access, data extraction, or even full system compromise.

To perform a thorough assessment of the login page for potential SQL injection vulnerabilities, the reporter employed the automated tool **sqlmap**, using the following command to test the input fields for injection points:

|  |
| --- |
| sqlmap -u http://172.16.4.2/login.php -- data="email=cee&pass=test&submit=Login&submitted=TRUE" |

The -u flag specifies the target URL (<http://172.16.4.2/login.php>), and the --data flag includes the **POST parameters** sent to the server:

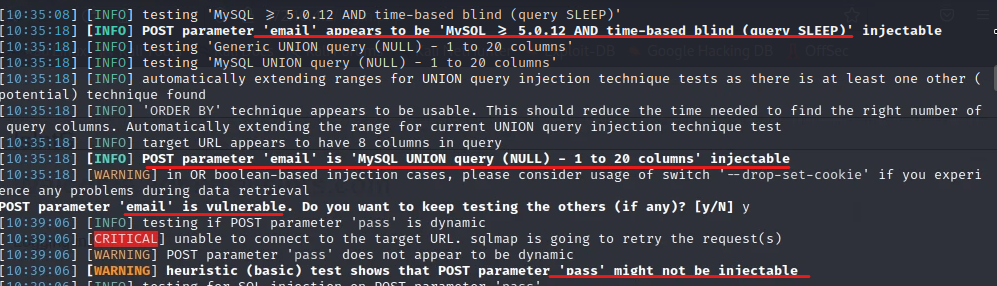
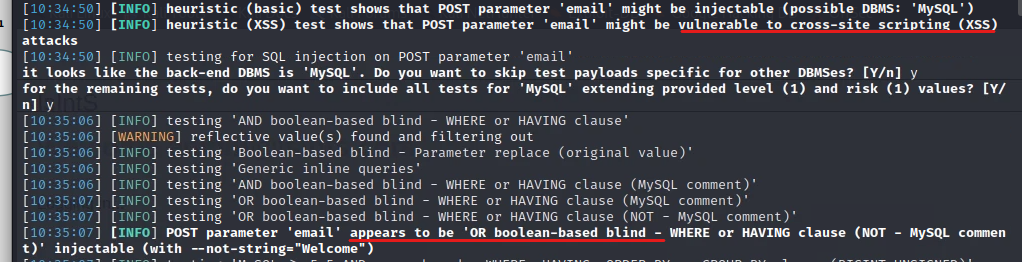
* email=cee (username field)
* pass=test (password field)
* submit=Login&submitted=TRUE (hidden/form submission flags).



This tells sqlmap to test the login form where users input credentials which automatically:

* **Probes the parameters** (email, pass) to detect if they’re vulnerable to SQL injection.
* **Sends malicious payloads** (e.g., ' OR 3=3 --) to trick the database into returning unauthorized data (e.g., bypassing login).
* Identifies database type (MySQL, PostgreSQL, etc.) and exploits misconfigured queries.

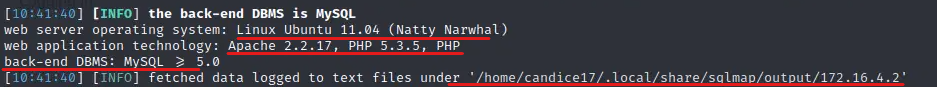
Upon execution, sqlmap reported several significant findings indicating critical vulnerabilities in the target application as seen below:



The **email parameter** in the login form was identified to be **vulnerable to SQL injection** through multiple techniques:

* + **Boolean-based blind SQL injection** as already explained.
  + **Cross site scripting (XSS).**
  + **UNION-based injection**, allowing the attacker to combine results from different SELECT queries into a single response.
  + **Time based** blind injection.

This information conclusively demonstrates that the login form is critically vulnerable to SQL injection, especially through the email parameter. At the end, it revealed the versions of technology being used by the server as seen below:



After the reporter did some request, it was revealed that there were some vulnerabilities, and the table below lists them:

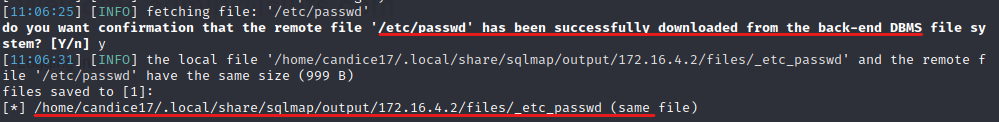
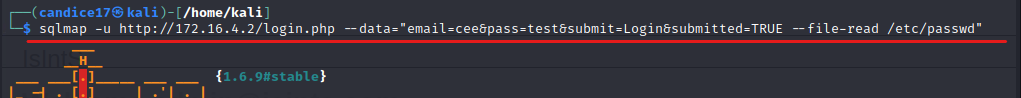
|  |  |  |
| --- | --- | --- |
| Technology | Version revealed | Vulnerabilities |
| **MySQL** | ≥ 5.0 | CVE-2012-2122 (Authentication Bypass) |
|  |  |  |
| **Apache** | 2.2.17 | - CVE-2011-3192 (DoS via Range Headers) - CVE-2017-7679 (HTTP Smuggling) |
| **Ubuntu** | 11.04 | End-of-Life (No security patches since 2012) |
| **PHP** | 5.3.5 | CVE-2011-2719 (LFI – local file Inclusion)  - CVE-2012-1823 (RCE – Remote Execution Command via CGI) |

Following this, the reporter proceeded to assess the full extent of the vulnerability by utilizing sqlmap. The objective at this stage was to verify whether the web application was susceptible to Local File Inclusion (LFI) through the database engine. Local File Inclusion (LFI) is a type of vulnerability commonly found in PHP-based websites that occurs when an application includes files based on user input without proper validation. This allows an attacker to manipulate the input to include arbitrary files from the server, potentially exposing sensitive information ([ref](https://www.geeksforgeeks.org/local-file-inclusion-lfi/)).

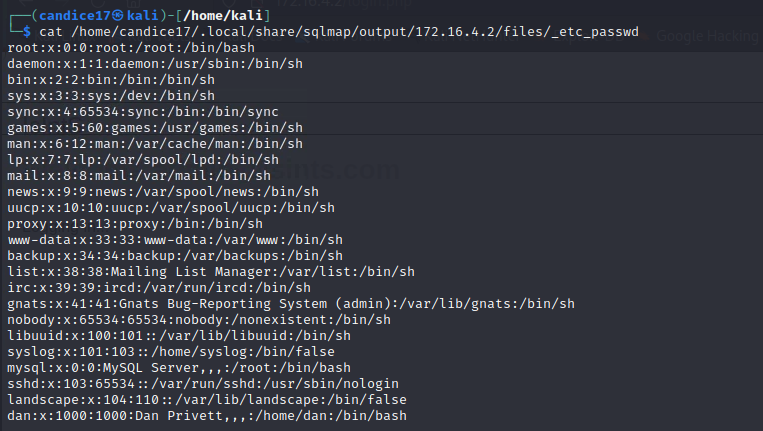
This involved testing MySQL's ability to read files from the local file system, a capability that can be exploited under certain misconfigurations. To do this, the reporter used the following command to attempt reading sensitive system files:

|  |
| --- |
| sqlmap -u 172.16.4.2/login.php - data="email=cee&pass=test&submit=Login&submitted=TRUE" --file-read /etc/passwd |

* **--file-read="/etc/passwd"**: Attempts to read the server’s /etc/passwd file through the database. The /etc/passwd file is a **critical system file** on Unix/Linux operating systems that stores essential user account information like username, user ID (UID), home directory, group ID (GID) and more ([ref](https://www.ibm.com/docs/en/aix/7.1?topic=passwords-using-etcpasswd-file)).

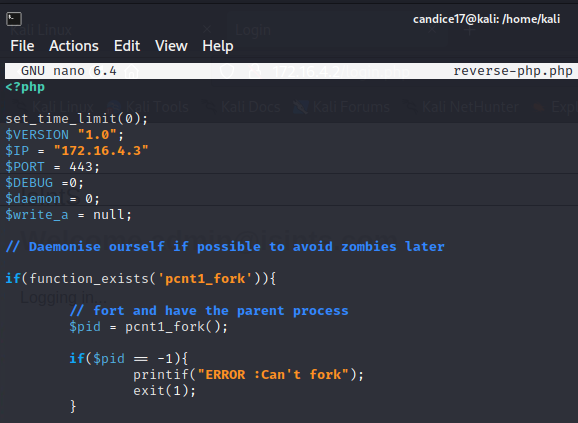


After successfully grabbing this file, its contents were displayed as seen below:

  
  
After successfully retrieving the /etc/passwd file through SQL Injection using the **--file-read** technique, the reporter proceeded to analyze the list of user accounts present on the target system. The discovery of typical Linux user accounts — such as **root**, and **admin** — indicated that privilege escalation might be achievable with deeper access.

To advance toward gaining an interactive shell on the compromised machine, the reporter uploaded a PHP reverse shell payload. For this, a widely recognized and reliable script from Pentestmonkey was used. The Pentestmonkey PHP reverse shell is a trusted tool in penetration testing, designed to provide a simple and effective way to establish a reverse connection from a vulnerable web server back to the attacker's machine ([ref](https://www.acunetix.com/blog/web-security-zone/what-is-reverse-shell/)).

The reporter created a file named **reverse-php.php** and used the nano text editor to modify the script by pasting the content from the pentestmonkey and changing the IP to specify the reporter’s IP address and listening port to receive the incoming shell connection. The full script will be attached to the submission file.



Following the earlier success in reading the /etc/passwd file, the reporter proceeded to take things a step further by writing a file to the target system. This time, instead of reading data from the target, the goal was to upload a PHP reverse shell script from the local machine to the target's web server located at 172.16.4.3.

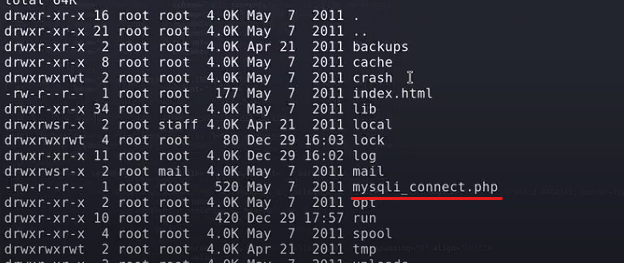
This was accomplished using the following sqlmap command:

|  |
| --- |
| sqlmap -u 172.16.4.2/login.php - data="email=cee&pass=test&submit=Login&submitted=TRUE" --file-write . /reverse-php.php –file-dest 172.16.4.2/var/www/rev.php |

The command leveraged a writable injection point to upload the reverse shell (reverse-php.php) to the web root directory of the target server. Once the script was successfully placed at /var/www/rev.php and triggered, it established a reverse connection back to the reporter's machine, effectively granting remote shell access to the system.

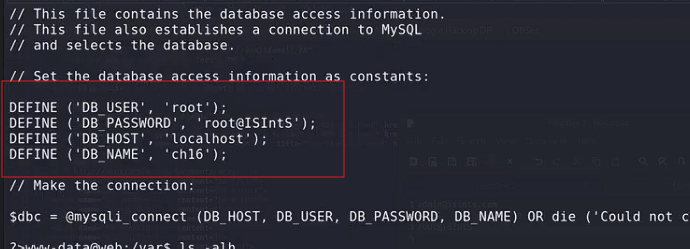
The shell session was running under the privileges of the web server user, which allowed access to typical directories and files used by the server. With an initial foothold gained, the reporter moved on to enumerate files, specifically scanning through the web root, configuration files, and other accessible areas in search of misconfigurations, hardcoded credentials, or sensitive information that could aid in privilege escalation or further compromise.

During the process of examining the structure of the server's directory, the reporter navigated to the root directory of the web application, commonly located at /var/www/html. Once inside, the ls command was executed to list all visible files and directories. Among the listed files, one in particular stood out — mysqli\_connect.php.



This file is typically used in PHP-based applications to handle database connections using the mysqli (MySQL Improved) extension, which allows interaction with MySQL or MariaDB databases (reff). Files like this often contain sensitive information, such as hardcoded database credentials.

To verify its contents, the reporter used the less command within the active reverse shell session to read the file. The output exposed the database connection details, including the username and the root password: root@ISintS.



To confirm these credentials, the reporter accessed the pWnOS machine on the network and attempted to log in using the credentials:  
 **username:** root and **password:** root@ISintS.

The login was successful, indicating that the default or known credentials were valid, as shown in the screenshot below.

### 8. Conclusion

The penetration testing exercise successfully identified several critical security weaknesses within the pWnOS environment. Notably, the presence of exposed services such as SSH and HTTP, outdated software versions (e.g., Apache 2.2.17 and legacy Linux kernels), and vulnerable login mechanisms were confirmed through enumeration and scanning tools like Nmap.

The use of SQL injection techniques and automated tools like sqlmap, further revealed the system's susceptibility to injection-based attacks — allowing unauthorized data extraction and potential file access (/etc/passwd). Additionally, the presence of default credentials (root:root@ISintS) on the pWnOS system highlighted poor credential hygiene, significantly increasing the risk of unauthorized access