Privilege Escalation and the Root of Chaos

T-401-ICYB: Lab 5

Alexander Joseph Emilsson   Alfa Reynisdóttir   Gísli Hrafn Halldórsson   Kim Anna Hudson

Executive Summary

In this lab we analysed a Linux virtual machine from the perspective of a normal unprivileged user and identified several paths to escalate our privileges to root. Using LinPEAS we found multiple weaknesses, including weak file permissions, a misconfigured `sudo` rule, a world-writeable `systemd` service unit, a Python interpreter with dangerous Linux capabilities, and a very weak root password. Each of these issues could be exploited to achieve full system compromise.

The goal of the assignment was to gain root access to the system and understand why these vulnerabilities exist and how to fix them. By exploring these different attack paths we learned how privilege escalation works in practice, how small configuration mistakes can be chained into full control of a system, and what issues system administrators should look out for to prevent similar problems.

I. Introduction

In this report, we will go over everything we did to discover the vulnerabilities of the system. What commands, what tools, and what resources we used. We found several glaring and inexcusable vulnerabilities in the system. Later in the report, we will also go over suggested fixes for every problem,

II. Methodology & Reconnaissance

First we set up the virtual machine and logged in as the normal user `master`. We then read through the documentation for LinPEAS to understand what kinds of misconfigurations it looks for.

We ran the command `./linpeas.sh -a > linpeas_output.txt`, which analyses the system and collects information about potential privilege-escalation vectors. To review the results in a readable way (with colours and control characters interpreted correctly), we used `less -r linpeas_output.txt`. Initially we tried viewing the file without the `-r` flag, but that broke the formatting.

From the LinPEAS output we identified several high-impact findings:

- A `sudo` rule that allows `/usr/bin/find` to be run as root without requiring a password. Because `find` supports the `-exec` option, this is immediately suspicious.
- The interpreter `/usr/bin/python3.11` has the capability `cap_setuid=ep`, meaning a Python process started by an unprivileged user can change its own UID (for example to UID 0).
- The backup script `/usr/local/bin/backup.sh` is world-writable, even though it is intended to be run by root.
- The `systemd` unit file `/etc/systemd/system/vuln.service` is also writable by the normal user, so the `ExecStart` command can be changed.
- The world-writable directory `/tmp` appears in the `$PATH`, which is dangerous if it is used by privileged shells.

In addition, it seems that LinPEAS was able to brute-force the root password and discovered that it was set to `toor`, an extremely weak password (short, all lowercase, no numbers or symbols).

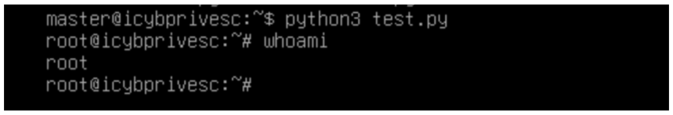
III. Exploitation Strategy

One of the simplest issues to abuse was the weak root password. Because the root password was set to `toor`, it could be brute-forced very quickly, giving us direct interactive root access.

We also exploited the Python capability issue. Since `/usr/bin/python3.11` has `cap_setuid=ep`, any user can start Python and change the process UID to 0 before launching a shell. For example:

```python
import os
os.setuid(0)
os.system("/bin/bash")
```

Running this as the normal user drops us into a root shell, as shown below:



```
master@icybprivesc:~$ python3 test.py
root@icybprivesc:~# whoami
root
root@icybprivesc:~#
```

The misconfigured `sudo find` rule is also straightforward to exploit. Because we can run `/usr/bin/find` as root without a password, and `find` supports the `-exec` action, we can execute `/bin/bash` as root with:
`sudo find / -exec /bin/bash \; -quit`
This starts a root shell by asking `find` (running as root via `sudo`) to execute `/bin/bash` the first time it encounters a file.

The `/tmp` issue can also be abused if `/tmp` appears before standard system directories in the PATH used by a privileged user. An attacker can create a fake `apt` binary in `/tmp`:

```
    echo '#!/bin/bash
    /bin/bash' > /tmp/apt
    chmod +x /tmp/apt
```

If root then runs a command such as `apt update && apt upgrade` and their `$PATH` resolves `apt` from `/tmp` first, the malicious script is executed instead of the real `/usr/bin/apt`, again giving a root shell.

We exploited the writable `systemd` unit `/etc/systemd/system/vuln.service` by editing its `ExecStart` directive to run a shell or arbitrary payload as root. Once the service is restarted (for example, on reboot or by an administrator), our modified command runs with full root privileges.

Finally, the writable backup script `/usr/local/bin/backup.sh` can be modified by a normal user to include malicious commands. When root later runs the backup (manually or via a scheduled job), those commands execute as root, providing yet another path to full system compromise.

IV. Remediation & Mitigation

First, `/tmp` should be removed from the `$PATH` for privileged users. Because `/tmp` is world-writeable, keeping it in PATH allows attackers to inject malicious executables that may be run as root instead of the intended system binaries.

The misconfigured `sudo find` rule should be corrected. `sudo` should not allow arbitrary use of `/usr/bin/find` with `-exec` as root without a password. The NOPASSWD rule for `find` should be removed or restricted so that it can only run specific, safe commands.

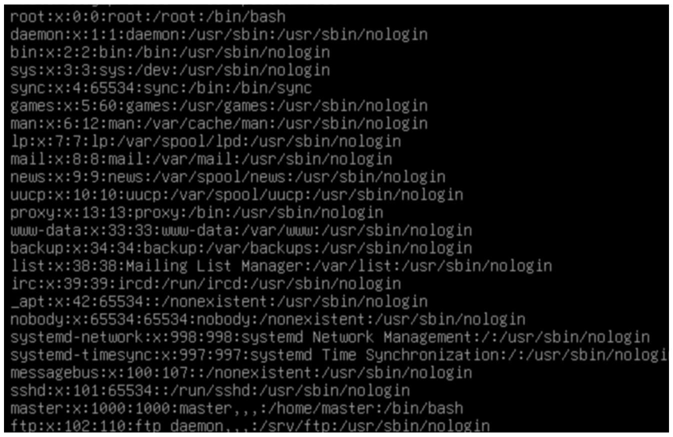
The root password must be replaced with a strong one. In addition, the system should enforce password complexity requirements and implement rate limiting for login attempts, for example with tools like Fail2ban, to make brute-force attacks harder.

The `cap_setuid=ep` capability should be removed from `/usr/bin/python3.11`, since a world-executable interpreter with this capability effectively behaves like a SUID-root binary. This can be done with: `sudo setcap -r /usr/bin/python3.11`.

To fix the `/etc/systemd/system/vuln.service` privilege-escalation issue, we must ensure that only root can modify the service unit file. For example: `sudo chown root:root /etc/systemd/system/vuln.service` and `sudo chmod 644 /etc/systemd/system/vuln.service`. Here, `chmod 644` means the owner (root) can read and write the file, while everyone else can only read it.

Similarly, the permissions on `/usr/local/bin/backup.sh` should be restricted so that only root can modify or execute the script. For instance: `sudo chown root:root /usr/local/bin/backup.sh` and `sudo chmod 700 /usr/local/bin/backup.sh`. With `chmod 700`, only the owner (root) has read, write, and execute access, and all other users have no access to the script.

V. The Sovereign Flag



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologi
messagebus:x:100:107::/nonexistent:/usr/sbin/nologin
sshd:x:101:65534::/run/sshd:/usr/sbin/nologin
master:x:1000:1000:master,,,:/home/master:/bin/bash
ftp:x:102:110:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
```

Contents of `/etc/passwd` file

VI. Reflection

Initial challenges were honestly just viewing the output, but the rest was more straightforward after that.
An interesting part of this lab was just how many different attack vectors there are from so many different angles. Seemingly, any small loophole or crack can be exploited. While many vulnerabilities aren't an issue on their own, when certain ones are present simultaneously, they can suddenly offer a huge attack vector for the attacker.
While learning what to account for is essential, seeing the attack surface from an attacker's viewpoint suddenly makes it crystal clear *why* all these things matter and *why* we have so many rules, regulations, and defences in place.

VII. Conclusion

We infiltrated an isolated virtual environment, applied reconnaissance and identified several critical privilege escalations. Using LinPEAS we started as a low-privileged user and ended up in root. We discovered several different weaknesses such as a weak root password, bad sudo settings, writeable system files and insecure permissions. Each of these issues provided its own path to root.
The lab showed how small oversights in linux permissions and configs can add up to serious security risks as well as reinforcing the importance of checking the whole system. By documenting everything step by step we were able to understand how attackers think and how to prevent these problems.