**Evitando SEHOP por SYSDREAM IT Security Services traducido por Ivinson/CLS**

SYSDREAM
IT Security Services

# Evitando SEHOP

Stéfan Le Berre
s.leberre@sysdream.com
Damien Cauquil
d.cauquil@sysdream.com

Tabla de contenidos

0. Introducción…….…….4
1. Especificaciones del SEHOP (versión corta)…….……..……..4
2. Tratar con SEHOP al explotar un desbordamiento de pila…….……..7
   2.1 Romper con el esquema de explotación clásica……...…..…….…..7
   2.2 La parte difícil…….…….…8
3. Prueba de Concepto (PoC)……..…..9
   3.1.Programa objetivo y las limitaciones…….……..9
   3.2.Error y Explotación……..……9
4. Conclusión…….…...…11
5. Créditos…….…….……..11
6. Bibliografía…….……..12

# 0. Introducción

Microsoft ha implementado recientemente en muchas versiones de Windows una nueva característica de seguridad llamada:

« Protección de sobreescritura del Manejo estructurado de excepciones» [1 y 2].

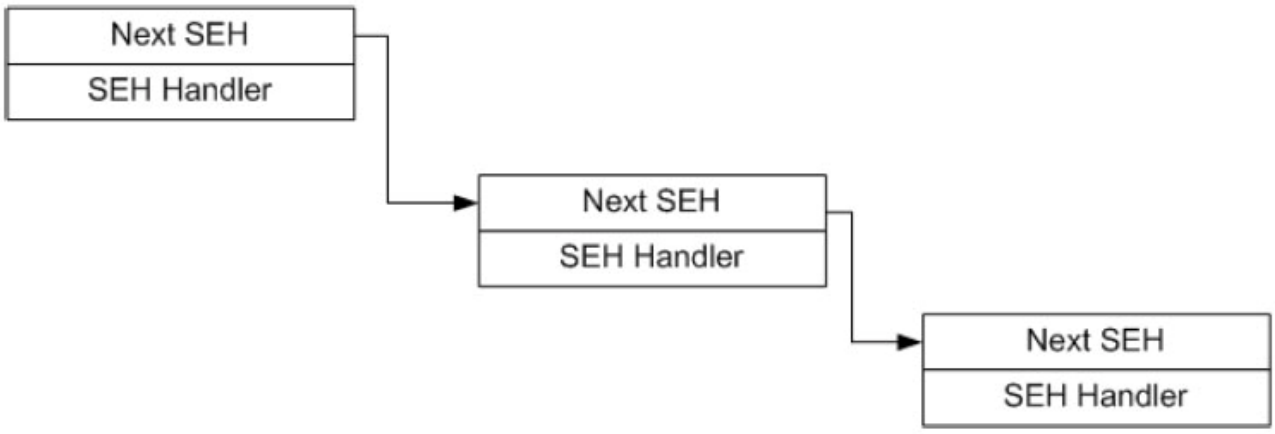
Estos sistemas son:

• Microsoft Windows 2008 SP0.
• Microsoft Windows Vista SP1.
• Microsoft Windows 7.

No se encontró ningún ataque conocido destinado a derrotar esta nueva característica, sino muchos papeles describiendo la función de sí misma y su robustez. En efecto, SEHOP parece ser tan fiable que Microsoft lanzó un parche para activar esta función de seguridad de forma predeterminada en todos los programas. ¿Ya es el final de desbordamientos de pila en Microsoft Windows? Todavía no, pero bajo algunas circunstancias, como explicaremos a continuación.

1. Especificaciones del SEHOP (versión corta)

SEHOP es una extensión de Manejo estructurado de excepciones e implementa más controles de seguridad en las estructuras SEH utilizadas por los programas. La característica principal de SEHOP comprueba el encadenamiento de todos las estructuras SEH presentes en la pila del proceso y sobre todo la última, que debe tener un valor de manejador especial apuntando justo a una función ubicada en ntdll. Aquí hay una cadena de SEH clásica:



Cada estructura de SEH apunta a la siguiente estructura y la última contiene un manejador específico que apunta a ntdll _except_handler4. Cuando explotamos sobrescribiendo una estructura SEH en la pila, el puntero del próximo SEH se sobrescribe con el fin de contener algún bytecode y el manejador SEH es sobrescrito para que apunte a una secuencia de instrucciones «POP POP RET» que se encuentran en un módulo sin SafeSEH.

El algoritmo de validación utilizado en SEHOP ha sido expuesto por A. Sotirov durante el Black Hat en el año 2008 [3]. Vamos a echar un vistazo:

```
BOOL RtlIsValidHandler(handler)
{
if (handler is in an image) {
if (image has the IMAGE_DLLCHARACTERISTICS_NO_SEH flag set)
return FALSE;
if (image has a SafeSEH table)
if (handler found in the table)
return TRUE;
else
return FALSE;
if (image is a .NET assembly with the ILonly flag set)
return FALSE;
// fall through
}
if (handler is on a non-executable page) {
if (ExecuteDispatchEnable bit set in the process flags)
return TRUE;
else
// enforce DEP even if we have no hardware NX
raise ACCESS_VIOLATION;
}
if (handler is not in an image) {
if (ImageDispatchEnable bit set in the process flags)
return TRUE;
else
return FALSE; // don't allow handlers outside of images
}
// everything else is allowed
return TRUE;
}
[...]
// Skip the chain validation if the
DisableExceptionChainValidation bit is set
if (process_flags & 0x40 == 0) {
// Skip the validation if there are no SEH records on the
// linked list
if (record != 0xFFFFFFFF) {
// Walk the SEH linked list
do {
// The record must be on the stack
if (record < stack_bottom || record > stack_top)
goto corruption;
// The end of the record must be on the stack
if ((char*)record + sizeof(EXCEPTION_REGISTRATION) >
stack_top)
```

```
goto corruption;
// The record must be 4 byte aligned
if ((record & 3) != 0)

goto corruption;
handler = record->handler;
// The handler must not be on the stack
if (handler >= stack_bottom && handler < stack_top)
goto corruption;
record = record->next;
} while (record != 0xFFFFFFFF);
// End of chain reached
// Is bit 9 set in the TEB->SameTebFlags field?
// This bit is set in ntdll!RtlInitializeExceptionChain,
// which registers FinalExceptionHandler as an SEH handler
// when a new thread starts.
if ((TEB->word_at_offset_0xFCA & 0x200) != 0) {
// The final handler must be ntdll!FinalExceptionHandler
if (handler != &FinalExceptionHandler)
goto corruption;
}
}
}
```

Tenemos que tomar en cuenta algunas limitaciones:

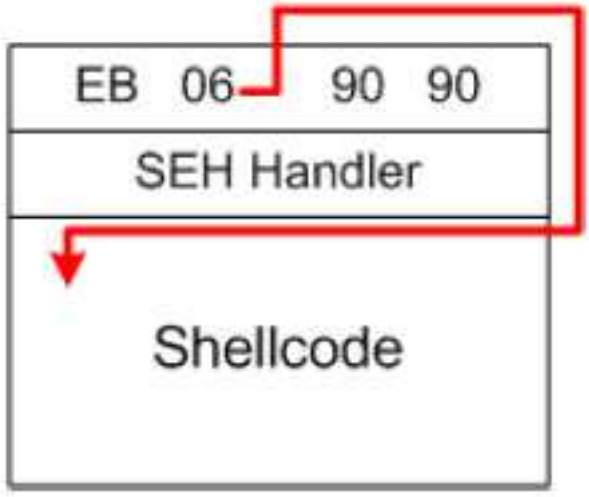
• El manejador SEH debe apuntar a un módulo sin SafeSEH.
• La página debe ser ejecutable.
• La cadena de SEH no debe ser alterada y debe terminar con una estructura que contenga un valor especial de SEH (0xFFFFFFFF) como puntero a la estructura del próximo SEH y un valor específico como manejador SEH).

• Todas las estructuras de SEH deben ser de 4 bytes alineados.
• La última estructura del manejador de SEH debe apuntar justo a ntdll en la rutina ntdll FinalExceptionHandler.
• Todos los punteros del SEH deben ser dirigidos a lugares de la pila.

# 2. Tratar con SEHOP al explotar un desbordamiento de pila.

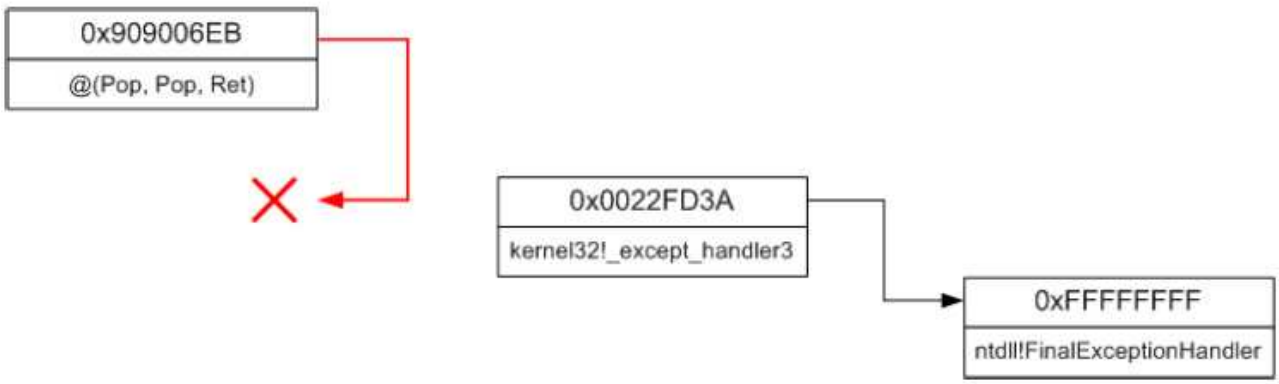
## 2.1 Romper con el esquema de explotación clásica.

Aquí está el esquema de explotación clásica utilizada:




El manejador SEH apunta a una secuencia de instrucciones «POP POP RET», y el primer miembro de la estructura contiene código en lugar de una dirección válida de pila. Mientras que manejamos una excepción, Windows Transfiere el control a manejadores de excepciones, siguiendo la cadena de SEH. Nuestro primer controlador de excepciones ha sido sobrescrito y el flujo de ejecución se redirige a la secuencia «POP POP RET» (en lugar de realmente manejar la excepción). Esta secuencia transfiere la ejecución a los dos primeros bytes del primer miembro de la estructura del SEH actual, que es una instrucción de salto (0xEB06) en nuestra Shellcode.

De esta manera, la cadena SEH se ha roto:



Los dos primeros bytes del primer DWORD de la estructura de SEH no se ejecutan nunca y podría ser alterada con el fin de hacer que el DWORD sea una dirección válida de pila. También es posible alterar los otros dos bytes para hacer que el DWORD apunte a una dirección válida de pila que controlemos y para definir los dos bytes como una instrucción de salto. El DWORD se puede evaluar tanto como una dirección de pila válida y una instrucción válida haciendo un salto al ser llamado.

Si ponemos un poco de estructura SEH falsa en la dirección apuntada por el primer DWORD, podemos simular una cadena SEH entera y controlar la segunda estructura SEH referenciada por la primera para validarla.

# 2.2 La parte difícil

Un problema permanece: ¿qué instrucción de salto puede ser codificada con un bytecode que representa una dirección de 4 bytes alineados?

Sólo una instrucción encaja aquí: JE (codificado como 0x74). Esta instrucción es un salto condicional: el salto se realiza sólo si el flag Z está activo. El flag Z no está configurado por defecto cuando Windows maneja una excepción, tenemos que configurarlo mediante la ejecución de una instrucción de cálculo de un valor cero, por ejemplo. «Xor» parece ser una buena opción, y nuestra solución es entonces clara: tenemos que saltar a una secuencia «XOR, POP, POP RET» con el fin de interpretar la instrucción JE como JMP. Esto fue lo difícil de esta técnica para evitar SEHOP.

Las instrucciones «XOR, POP, POP RET» no son tan difíciles de encontrar, podemos encontrar algunas secuencias mirando el final de las funciones que devuelven un resultado nulo:

```
XOR EAX, EAX
POP ESI
POP EBP
RET
```

Con el fin de garantizar la explotación, ponemos en nuestra prueba de concepto una secuencia «XOR, POP, POP, RET».

Digamos que tenemos una primera estructura de SEH en 0x0022FD48, se podría crear una segunda en 0x0022FD74 que contenga un puntero al próximo SEH con un valor de 0xFFFFFFFF y el manejador apuntando a ntdll FinalExceptHandler. Si dejamos al manejador de la estructura del primer SEH apuntando a una secuencia «XOR, POP, POP, RET », podemos redirigir el flujo de ejecución a donde queramos sin llegar a romper la cadena. En concreto, recreamos una cadena SEH falsa (funcional) que redirija la ejecución en una determinada Shellcode.

Una limitación importante reside en la característica de ASLR presente en Microsoft Windows 7 y Vista. La explotación se basa en el hecho de que conocemos la dirección de ntdll FinalExceptHandler, pero en la realidad la dirección cambia en cada reinicio de la máquina. La ImageBase de Ntdll es al azar en el arranque y hace más difícil la explotación. Hemos hecho algunas pruebas en el ASLR (sin invertirlo) y parece que sólo 9 bits de los 16 que representan a la ImageBase se eligen al azar, lo que representa 1 posibilidad de más de 512 a explotar con éxito un desbordamiento de pila evitando el SEHOP.

# 3. Prueba de Concento (PoC)

## 3.1 Programa objetivo y las limitaciones

Hemos creado un pequeño programa para demostrar nuestra técnica en Windows 7. Este programa sólo copia el contenido de un archivo («OwnMe.txt») en la memoria y rompe la pila durante esta operación, provocando una excepción capturada por el manejador de excepciones.

Todo lo que tenemos que hacer es:

• Crear una cadena SEH (funcional) falsa.
• Modificar el manejador de la última estructura de SEH para que apunte a ntdll FinalExceptionHandler.
• Poner una Shellcode en la pila (compatible con Windows 7).
• Buscar una secuencia «XOR, POP, POP, RET» en la memoria.

En primer lugar, la secuencia «XOR, POP, POP RET» es conocida porque lo ponemos en el código programa. Podemos encontrar esta secuencia en la dirección de memoria 0x004018E1.

## 3.2 Error y Explotación

Cuando el programa da error (crashea), tenemos:

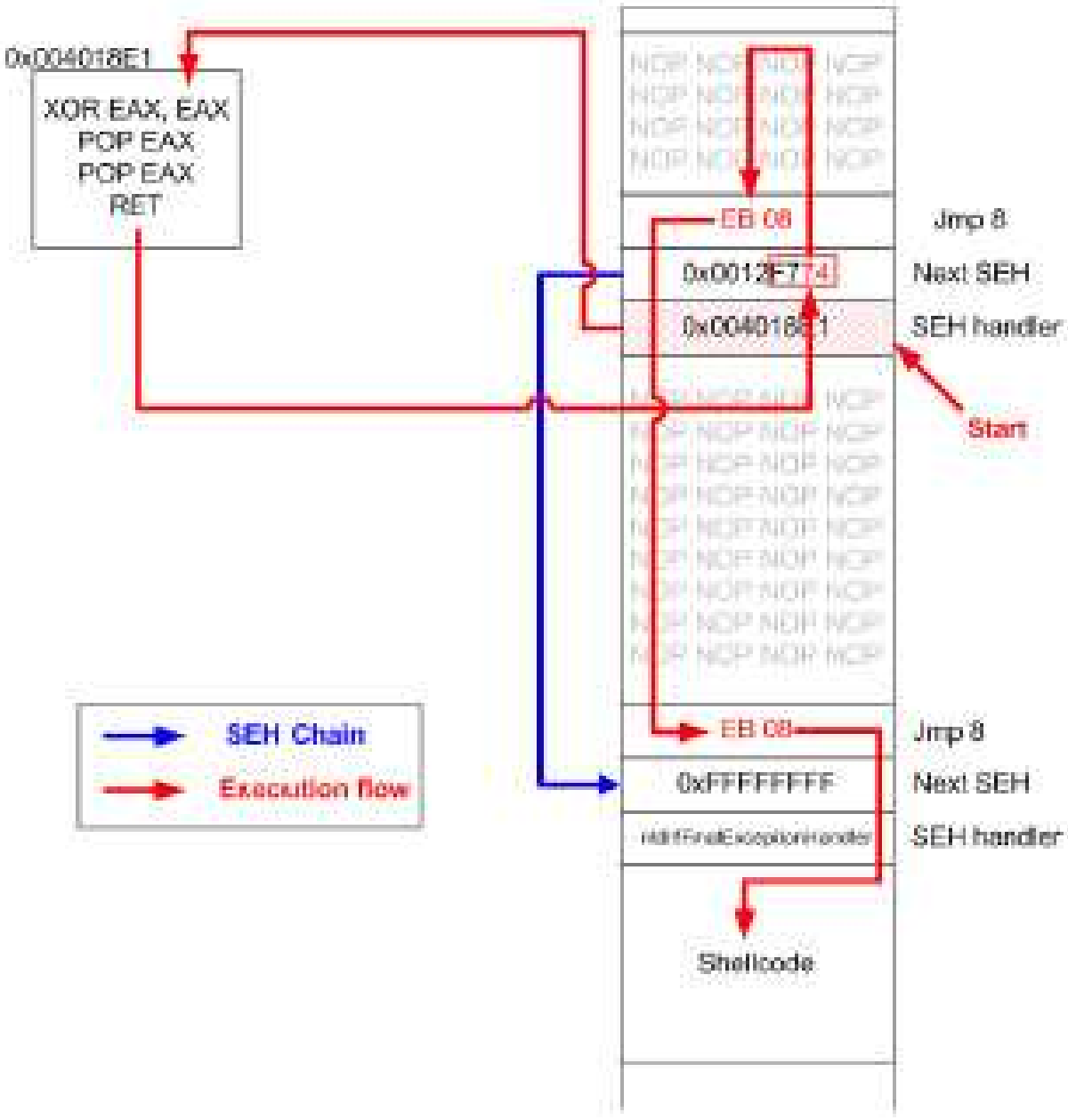
0012F700 41414141 Pointer to next SEH record (puntero al próximo registro SEH).
0012F704 41414141 SE handler (Manejador SE).

Entonces, tenemos que crear una cadena (funcional) SEH falsa con una segunda estructura de SEH almacenada en 0x0012F774 que contenga 0xFFFFFFFF como primer miembro de estructura de (puntero al próximo SEH) y ntdll

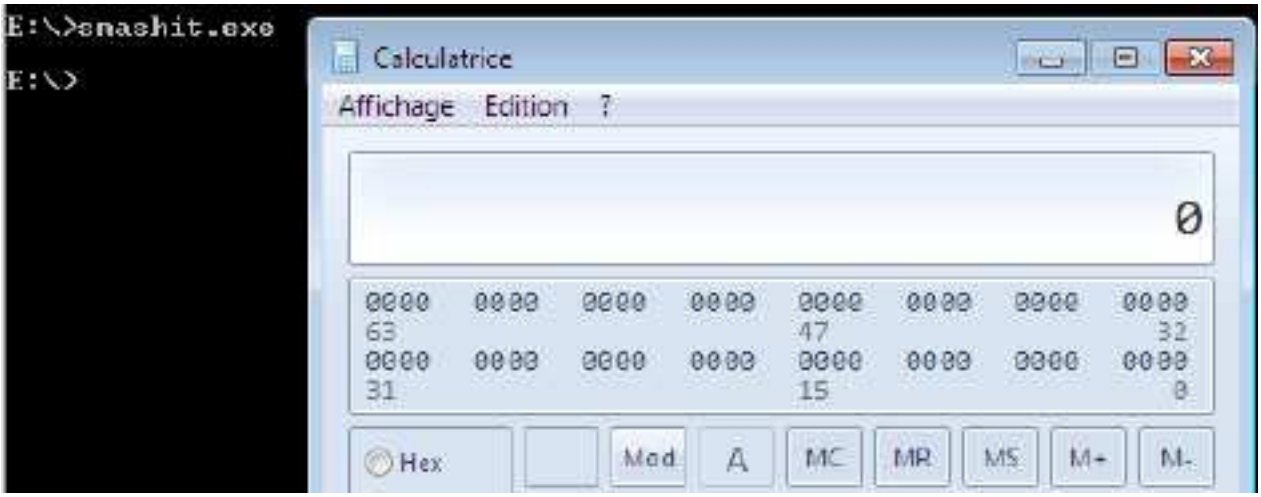
FinalExceptionHandler como manejador SEH. La estructura del SEH ubicado en 0x0012F700 luego es modificada para apuntar a la estructura de SEH falsa que hemos creado, y el manejador es modificado hacia 0x004018E1.

Una instrucción de salto (JMP+8) se coloca antes de cada estructura del SEH con el fin de evitar la ejecución de datos.

Cuando explotamos, la ejecución debería funcionar de la siguiente manera:



Esta Shellcode sólo se ejecutará la calc.exe (calculadora de Windows):




Nota: El programa objetivo y una exploit funcional están disponibles en el zip unido a este documento. También se puede descargar desde el sitio web Sysdream [4].

# 4. Conclusión

SEHOP no es la última protección contra desbordamientos de pila si se usa solo. Desde que esta extensión de SafeSEH fue liberada, demasiadas personas lo consideran como irrompible. Acabamos de demostrar que es posible saltarse la protección de sobrescritura del manejo estructurado de excepciones bajo algunas circunstancias. Pero SEHOP es una excelente característica de seguridad nativa cuando se utiliza junto con el ASLR y DEP, no podemos negarlo.

# 5. Creditos

http://sysdream.com/
http://ghostsinthestack.org/
http://virtualabs.fr/

# 6. Bibliografía

[1] Preventing the Exploitation of Structured Exception Handler (SEH) Overwrites with SEHOP:

http://blogs.technet.com/srd/archive/2009/02/02/preventing-the-exploitation-of-seh-overwriteswith-sehop.aspx

[2] SEHOP per-process opt-in support in Windows 7:

http://blogs.technet.com/srd/archive/2009/11/20/sehop-per-process-opt-in-support-in-windows-7.aspx

[3] Bypassing Browser Memory Protections:
http://taossa.com/archive/bh08sotirovdowd.pdf

[4] ZIP containing our target program & exploit
http://www.sysdream.com/SEHOP.zip

Tutorial original en Inglés:

http://www.shell-storm.org/papers/files/760.pdf

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