SEH overwrite and its exploitability

Shuichiro Suzuki
Fourteenforty Research Institute Inc.
Research Engineer
Agenda

• Theme and Goal
• Review of SEH overwrites
• Protection mechanisms for SEH overwrites
• Bypassing protection mechanisms.
• Demonstration
• Conclusion
Theme and goal

Theme

- SEH overwriting is one of the major methods for exploiting Windows software.
- We already have several protection mechanisms for SEH overwrites.
- Is the protection provided by DEP and SEHOP enough?
- How about the protection from SafeSEH and SEHOP?

Goal

- To reveal which combinations of the known protection mechanisms for SEH overwrites are really effective.
- On the way to the goal, I will show you that we can bypass SafeSEH and Software DEP and Hardware DEP and SEHOP, all at the same time, under certain conditions.
Target environment

- Windows XP SP3
- Windows Vista SP1
- Windows 7

- 32bit processes on x86
- 32bit processes on x64 (WOW64)

- Visual Studio 2008 was used to compile all the programs in this presentation.
Review of SEH Overwrite
About SEH

• SEH is “Structured Exception Handling”
• Exception handling system provided by Windows

```c
int test(void){
    __try{
        // Exception may occur here
    }
    __except( EXCEPTION_EXECUTE_HANDLER ){
        // This handles the exception
    }

    return 0;
}
```
typedef struct _EXCEPTION_RECORD{
    struct _EXCEPTION_RECORD *next;
    PEXCEPTION_ROUTINE _handler;
} EXCEPTION_RECORD;
SEH Overwrite Attack

Stack
- Buffer
  - 0x0012ff50
  - 0x0505690

Process Memory Space
- Module (EXE or DLL)
  - Handler1

Overflow!
Stack after an exception handler is called

```c
EXCEPTION_DISPOSITION __cdecl _except_handler (  
    struct _EXCEPTION_RECORD * _ExceptionRecord,  
    void * _EstablisherFrame,  
    struct _CONTEXT * _ContextRecord,  
    void * _DispatcherContext  
);  
```

Address to return

ESP

_ESCExceptionRecord

_ESCEstablisherFrame

_ESCContextRecord

_ESCDispatcherContext

ESP+8

0x909006eb

_handler

Attackers Arbitrary Code

pop eax

pop eax

retn
Protection mechanisms
Protection mechanisms

• /GS
• SafeSEH
• SoftwareDEP
• SEHOP
• Hardware DEP
• ASLR
/GS

- Stack Guard by Visual Studio compiler
- This checks if a buffer overflow occurs before returning from a function.
- This is irrelevant to SEH overwrites because an exception can be generated before the check.
Protection mechanisms

SEH overwrites specific
SafeSEH

A module protected by SafeSEH

Header

<table>
<thead>
<tr>
<th>Handler</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handler1</td>
<td>0x00002550</td>
</tr>
<tr>
<td>Handler2</td>
<td>0x00049080</td>
</tr>
<tr>
<td>Handler3</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Code

“pop,pop,ret” sequence

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00002550</td>
</tr>
<tr>
<td>Handler1</td>
</tr>
<tr>
<td>0x00049080</td>
</tr>
<tr>
<td>Handler2</td>
</tr>
</tbody>
</table>
Weakness of SafeSEH

Module
- SafeSEH Protected
- Not SafeSEH Protected

Data area
- without executable attribute
- with executable attribute

Memory Space
- pop pop ret
- Data area in a module
- pop pop ret
- pop pop ret
- pop pop ret
Modules not protected by SafeSEH

From Thunderbird 2.0.0.23
* All modules in Thunderbird 3.0.3 are compiled with /SafeSEH
pop pop pop ret in the wild

From thunderbird.exe
Additional check by Software DEP

- Module SafeSEH Protected
- Module Not SafeSEH Protected
- Data area without executable attribute
- Data area with executable attribute

Memory Space

- pop pop ret
- Data area in a module
- pop pop ret
- pop pop ret
- pop pop ret

Weekness
Software DEP

KPROCESS structure

0: kd> dt nt!_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 DisableExceptionChainValidation : Pos 6, 1 Bit
+0x000 Spare : Pos 7, 1 Bit
Software DEP

- **Module SafeSEH Protected**
- **Module Not SafeSEH Protected**
- **Data area without executable attribute**
- **Data area with executable attribute**

Memory Space:
- pop pop ret
- pop pop ret
- Data area in a module
- pop pop ret
- pop pop ret
- pop pop ret
SEHOP
SEH overwrites protection

SEH Chain
Stack

Memory Space

Overwrite!

0x909006eb
0x04224550
0x0012ff90
0xffffffff

FinalExceptionHandler

Proper Handler
Module 1
pop pop ret
Module 2
NTDLL.DLL

SEH overwrites protection

Pop pop ret
Protection mechanisms

Generic protections
Hardware DEP and ASLR

• Hardware DEP prevents code without executable attribute from being executed.
• ASLR has several impacts on the SEH overwrites. (Explain later)
Bypassing protection mechanisms
SafeSEH and Software DEP

Memory Space

- Pop pop ret
- Pop pop ret
- Data area in a module
- Pop pop ret
- Pop pop ret
- Pop pop ret

Module SafeSEH Protected

Module Not SafeSEH Protected

Non module area (Data area)

Non module area with executable attribute
Bypassing SEHOP

• It is weak if the address of a buffer overflow and FinalExceptionHandler is known.
• Attackers can recreate a proper SEH chain.
Bypassing Hardware DEP

- Return-into-libc
- Return-oriented programming

[Diagram showing stack with ESP, VirtualAlloc, memcpy, and other function calls leading to buffer overflow.]
Bypass /GS, SafeSEH, Software DEP, Hardware DEP, SEHOP

- Recreate SEH Chain (bypassing SEHOP)
- Overwritten exception handler address must be in a module which is not SafeSEH enabled (bypassing SafeSEH)
- Create a stack to execute desired code (bypassing Hardware DEP)
- To execute the code using the stack above, we have to set an exception handler to some stack rewind and return instructions (bypassing Hardware DEP)
- Trigger an exception
Recreating the SEH Chain and Setting the Exception Handler Address

Stack

Buffer

0x0012ff4C

0x0505690

Attackers Arbitrary Code and Data

0xffffffff

FinalExceptionHandler

Process Memory Space

Module (EXE or DLL)

Handler1

Module (EXE or DLL) No SafeSEH

SEH Chain

Exception!
Good instruction to bypass Hardware DEP

Stack
- Exception Information
- Buffer
- 0x0012ff50
- 0x0505690
- Attackers Arbitrary Code
- 0xffffffff FinalExceptionHandler

Process Memory Space
- Module (EXE or DLL)
- Handler1
- Module (EXE or DLL) No SafeSEH
- add esp, 0x0C24
- retn
Create proper stack for return-into-libc

**Stack**
- Exception Information
- Buffer
- 0x0012ff50
- 0x0505690
- Attackers Arbitrary Code
  - 0x07c8622A4
  - 0x00401110
  - 0x30000000
  - 0x00000800
  - 0x00003000
  - 0x00000040
  - 0x30000000
  - 0x0012F758
  - 0x00000800
- 0xffffffff
- 0x0c24
- FinalExceptionHandler

**Process Memory Space**
- Module (EXE or DLL)
- Handler1
- Module (EXE or DLL) No SafeSEH
- add esp, 0x0C24
- retn

Address of VirtualAlloc
Address of memcpy
Address to be allocated
The size to be allocated
MEM_COMMIT|MEM_RESERVE
PAGE_EXECUTE_READWRITE
Address to return after memcpy
Dest to copy
Src of copy
The size to be copied
Exception handlers which can be used

Stack

Exception Information
Buffer
0x0012ff50
0x0505690
Attackers Arbitrary Code
0x7c8622A4
0x00401110
0x30000000
0x00000800
0x00003000
0x00000040
0x30000000
0x0012F758
0x00000800
0xffffffff

Process Memory Space

Module(EXE or DLL)
Handler1
Module(EXE or DLL) No SafeSEH
add esp, 0x0C24
retn
add esp, 0x0CF8
retn

Address of VirtualAlloc
Address of memcpy
Address to be allocated
The size to be allocated
MEM_COMMIT|MEM_RESERVE
PAGE_EXECUTE_READWRITE
Address to return after memcpy
Dest to copy
Src of copy
The size to be copied

Doesn’t work
Too small rewind

May be too big.
Depends on the stack size.
Example of “add esp + retn”

From Thunderbird 2.0.0.23
Summarize the condition

• The address of the stack where buffer overflow occurs is known.
• The address of the FinalExceptionHandler in ntdll.dll is known.
• A process has a module not protected by /SafeSEH
• “add esp + retn” instructions which matches followings can be found in the module.
  – The amount of “add esp” rewind is larger than the stack made by windows exception dispatcher.
  – The amount of “add esp” rewind is smaller than the stack size when the exception handler is called.
• The address of VirtualAlloc and memcpy is known (or some alternatives can be used).
• Attacker can write 0x00 on the stack by using buffer overflow, which means string functions can’t be used to exploit. (At least the method I showed here can not be used if this doesn’t match)
Demonstration

• Demonstration on Windows 7
• I assumed that “add esp,0x0c24 – retn” can be found in a non /SafeSEH protected module.
• /GS, Software DEP, Hardware DEP, SEHOP are all enabled.
• ASLR is disabled.
Importance of ASLR

• Makes it difficult to recreate proper SEH chain
• Makes it difficult to find “add esp + retn” instructions at a fixed address.
• Makes it difficult to set FinalExceptionHandler address in the last element of SEH chain.
## Conclusion

<table>
<thead>
<tr>
<th>Protections</th>
<th>Windows XP SP3</th>
<th>Windows Vista SP1</th>
<th>Windows 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>/GS + SafeSEH</td>
<td>Exploitable by using data area as an exception handler</td>
<td>Exploitable by using data area as an exception handler</td>
<td>Exploitable by using data area as an exception handler</td>
</tr>
<tr>
<td>/GS + SafeSEH + Software DEP</td>
<td>If all modules are SafeSEH protected its difficult to exploit</td>
<td>If all modules are SafeSEH protected its difficult to exploit</td>
<td>If all modules are SafeSEH protected its difficult to exploit</td>
</tr>
<tr>
<td>/GS + Software DEP + Hardware DEP</td>
<td>Exploitable by Return-into-libc or Return-oriented programming</td>
<td>Exploitable by Return-into-libc or Return-oriented programming</td>
<td>Exploitable by Return-into-libc or Return-oriented programming</td>
</tr>
<tr>
<td>/GS + Software DEP + SEHOP</td>
<td>-</td>
<td>Exploitable by recreating proper SEH chain</td>
<td>Exploitable by recreating proper SEH chain</td>
</tr>
<tr>
<td>/GS + SafeSEH + SEHOP</td>
<td>-</td>
<td>Exploitable by recreating proper SEH chain and using data area as an exception handler</td>
<td>Exploitable by recreating proper SEH chain and using data area as an exception handler</td>
</tr>
<tr>
<td>/GS + Software DEP + SEHOP + Hardware DEP</td>
<td>-</td>
<td>Exploitable by recreating proper SEH Chain and using data area and return-oriented programming</td>
<td>Exploitable by recreating proper SEH Chain and using data area and return-oriented programming</td>
</tr>
<tr>
<td>/GS + SEHOP + ASLR</td>
<td>-</td>
<td>Difficult to exploit</td>
<td>Difficult to exploit</td>
</tr>
<tr>
<td>/GS + Software DEP + SEHOP + Hardware DEP + ASLR</td>
<td>-</td>
<td>Difficult to exploit</td>
<td>Difficult to exploit</td>
</tr>
</tbody>
</table>
ASLR + DEP

• “Bypassing Browser Memory Protections” (Alexander Sotirov, Mark Dowd) shows how to bypass them.

• But it was without SEHOP
Questions?