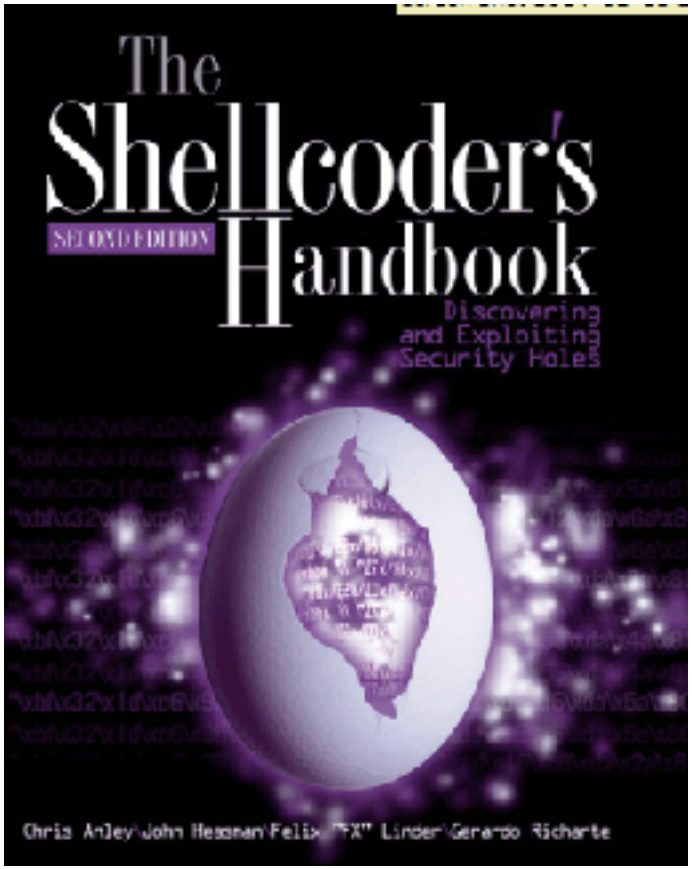
CNIT 127: Exploit Development
Ch 8: Windows Overflows
Part 1



Rev. 3-15-17

Topics

- Thread Environment Block and Process Environment Block
- Stack-Based Buffer Overflows
- Defeating ASLR (Address Space Layout Randomization)
- Frame-Based Exception Handlers
- SEH Protections
- Defenses in Server 2008 (not in textbook)

Thread Environment Block and Process Environment Block

Thread Environment Block (TEB)

- Stores information about the currently executing thread
- Also called the Thread Information Block
- Officially undocumented for Win 9x
- Partially documented for Win NT
- So many Win32 programs use undocumented TEB fields that they are effectively part of the API

Thread Environment Block (TEB)

- TEB can be used to get a lot of information about a process without calling the Win32 API
- Can get
  - Last error
  - Import Address Table
  - Process startup arguments
  - and more

Thread Environment Block (TEB)

- SEH pointer
- Info about the stack
  - Much more (Link Ch 8a)

Contents of the TIB (32-bit Windows) [edit]

Position	Length	Windows Versions	Description
FS:[0x00]	4	Win9x and NT	Current Structured Exception Handling (SEH) frame
FS:[0x04]	4	Win9x and NT	Stack Base / Bottom of stack (high address)
FS:[0x08]	4	Win9x and NT	Stack Limit / Ceiling of stack (low address)

FS: and GS:

- Segment Registers
- Left over from very early operating systems
- Not used much anymore for their original purpose
- On Windows 32-bit x86, FS: is used to point to the TEB
  - Links Ch 8e, 8y

Process Environment Block (PEB)

- An opaque data structure
  - Details are hidden from its users
  - Values are only intended to be manipulated by calling subroutines that can access the hidden information
- Used by Win NT internally
- Most fields intended only for internal OS use

Process Environment Block (PEB)

- Only a few fields are documented by Microsoft
- Contains data structures that apply across a whole process

Process Environment Block (PEB)

Fields of the PEB that are documented by Microsoft[2]

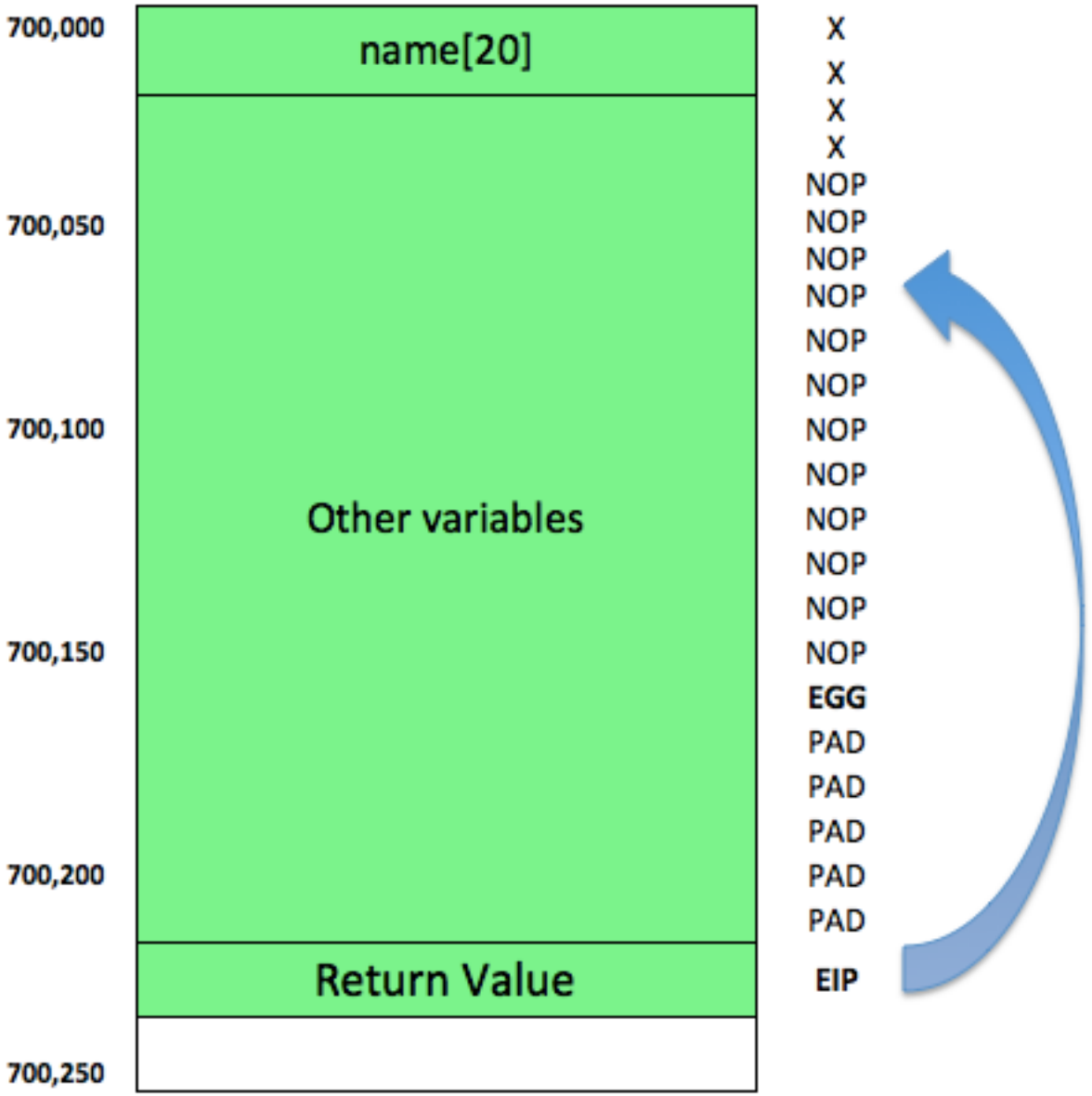
Field	meaning	notes
`BeingDebugged`	Whether the process is being debugged	Microsoft recommends not using this field but using the official Win32 `CheckRemoteDebuggerPresent()` library function instead.[2]
Ldr	A pointer to a `PEB_LDR_DATA` structure providing information about loaded modules	Contains the base address of kernel32 and ntdll.
ProcessParameters	A pointer to a `RTL_USER_PROCESS_PARAMETERS` structure providing information about process startup parameters	The `RTL_USER_PROCESS_PARAMETERS` structure is also mostly opaque and not guaranteed to be consistent across multiple versions of Windows.[4]
PostProcessInitRoutine	A pointer to a callback function called after DLL initialization but before the main executable code is invoked	This callback function is used on Windows 2000, but is not guaranteed to be used on later versions of Windows NT.[2]
SessionId	The session ID of the Terminal Services session that the process is part of	The `NtCreateUserProcess()` system call initializes this by calling the kernel's internal `MmGetSessionId()` function.[3]

Stack-Based Buffer Overflows

Classic Technique

- Overwrite the saved return address
  - With a value that points back into the stack
- When the function returns, it copies the return address into EIP
- Return address points to NOP Sled

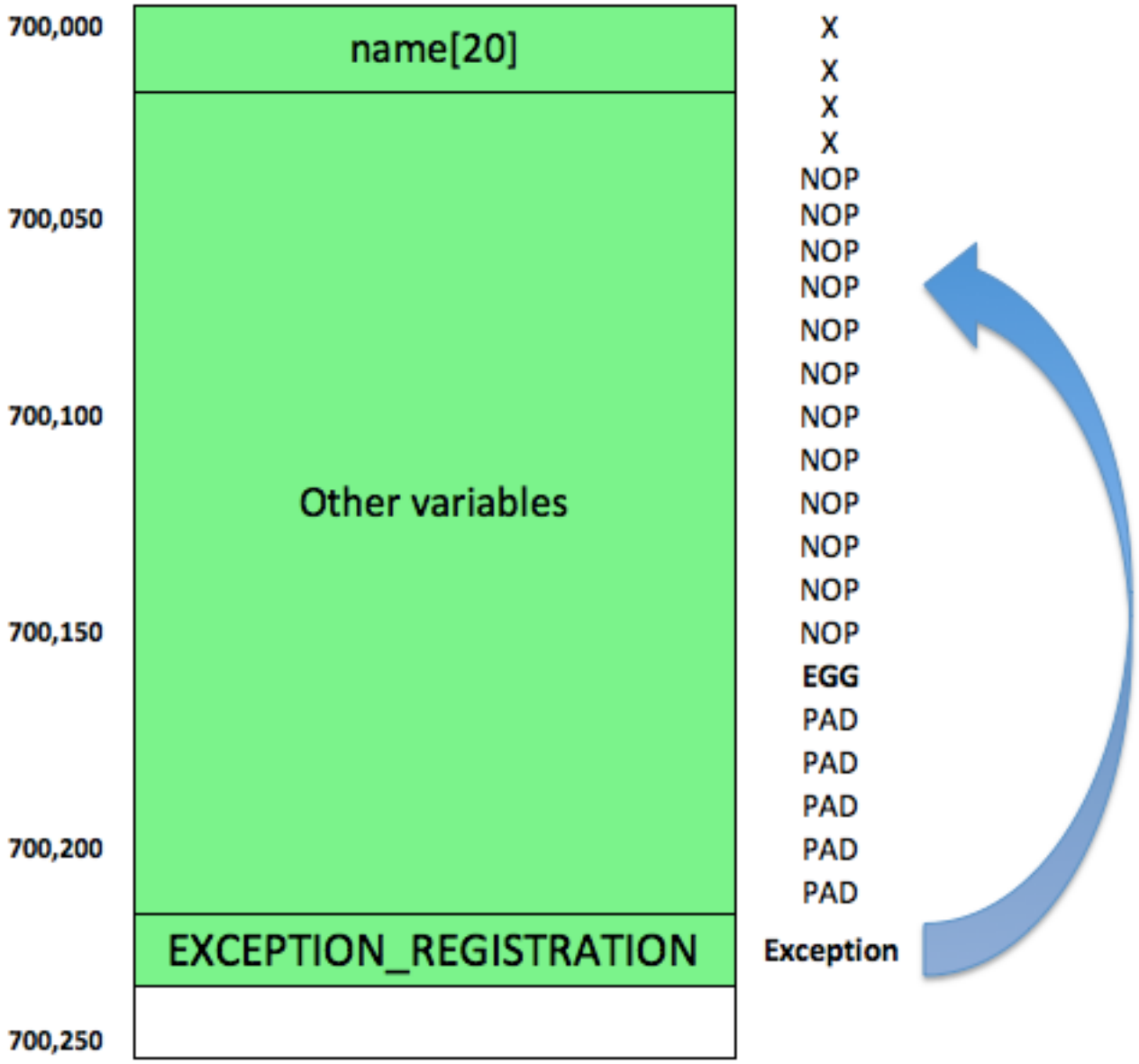
Buffer Overflow Using Return Value



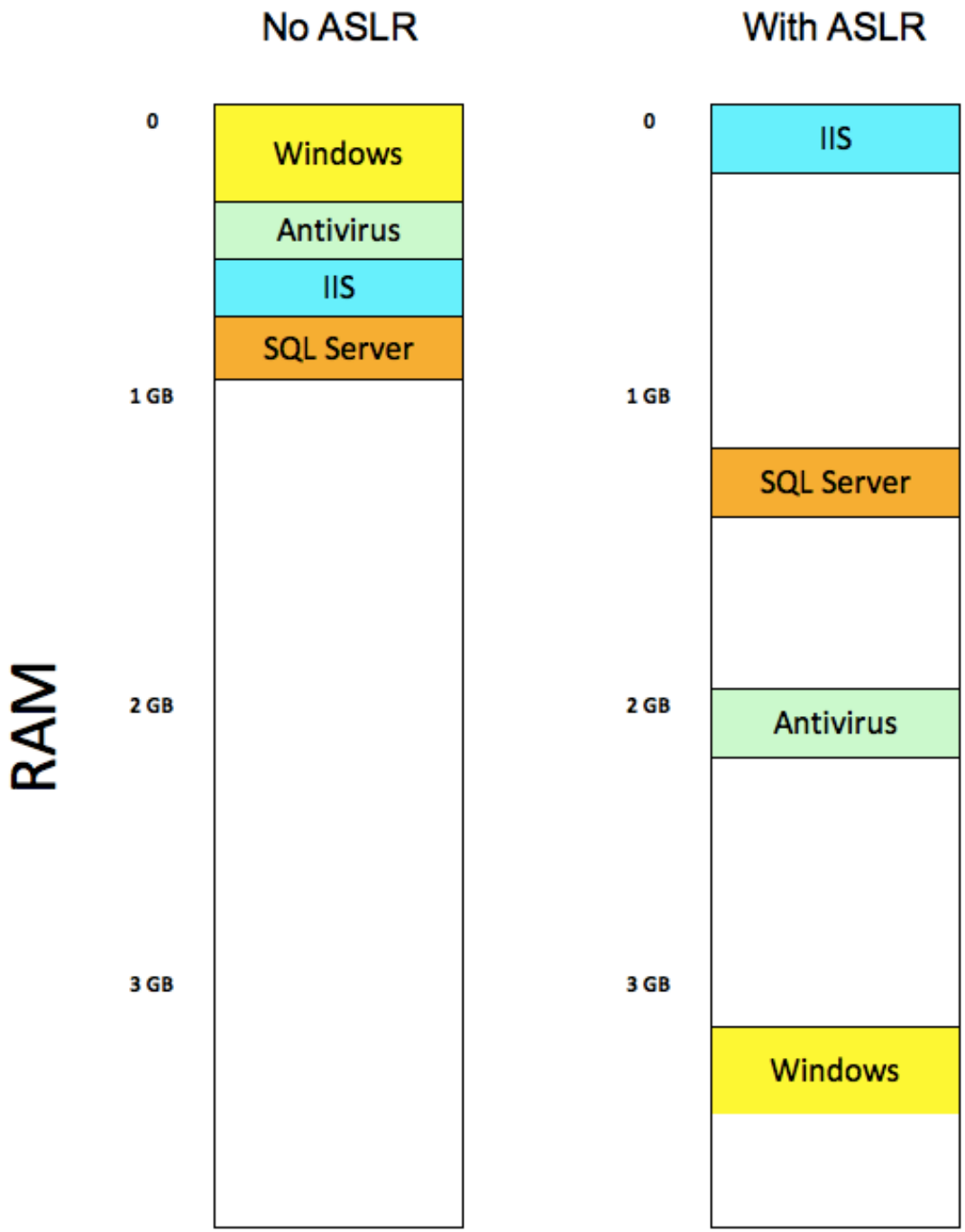
SEH Technique

- Overwrite the SEH
  - With a value that points back into the stack
- Trigger an exception
- Modified exception handler points to NOP Sled

Buffer Overflow Using SEH

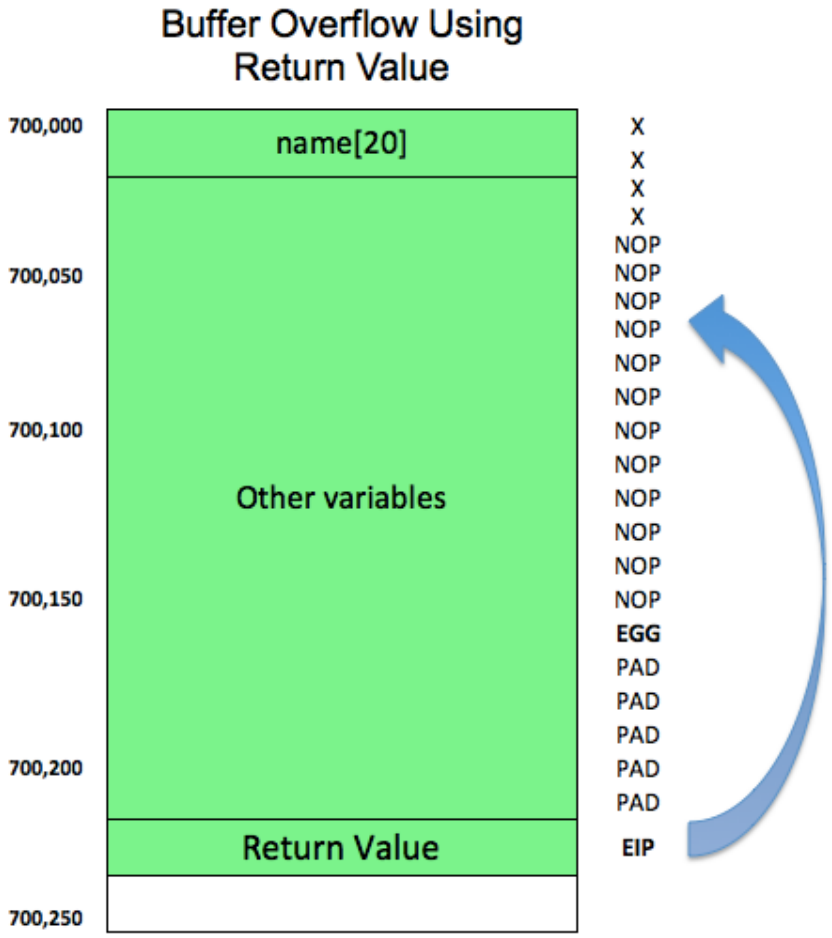


Defeating ASLR (Address Space Layout Randomization)


No ASLR
With ASLR
RAM
0
1 GB
2 GB
3 GB
Windows
Antivirus
IIS
SQL Server
0
1 GB
2 GB
3 GB
IIS
SQL Server
Antivirus
Windows

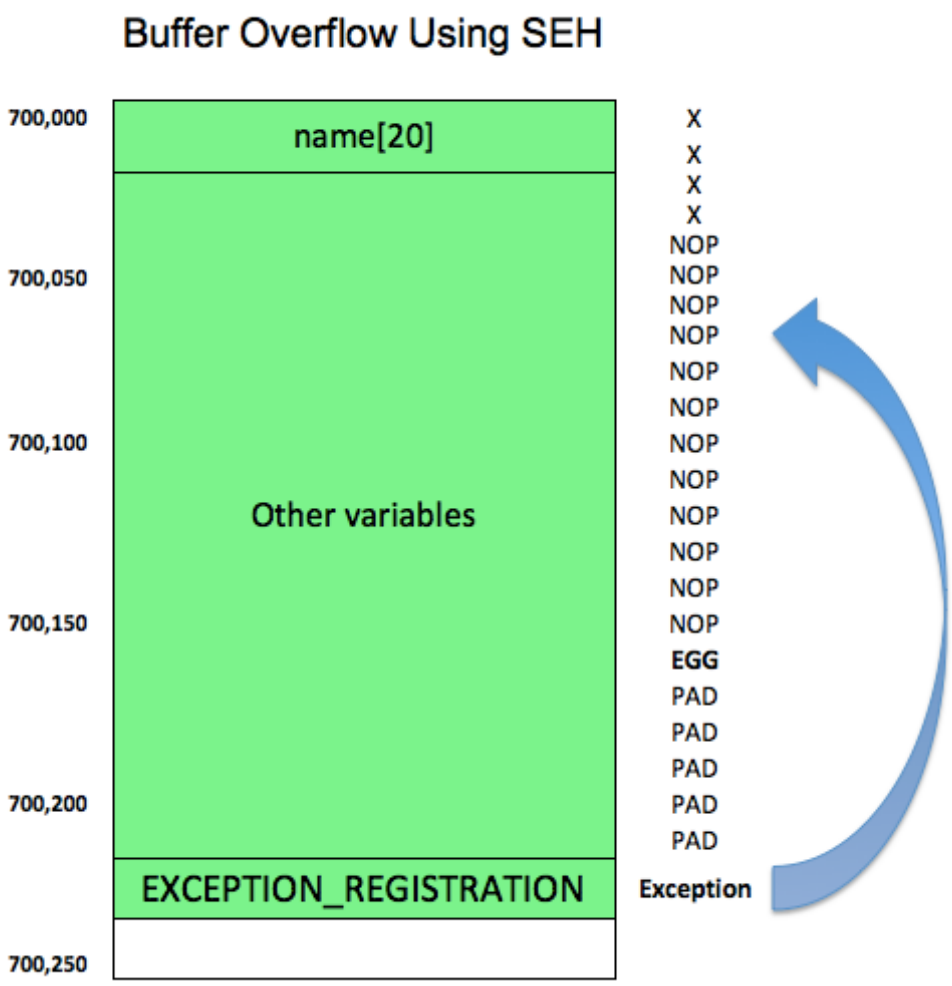
Using Return Value

- We're inside the function
- ESP points to the stack
- Find a JMP ESP and insert its address into the return value



Using SEH

- Old Windows versions left the address of the SEH in EBX
- Newer versions clear all registers
- But SEH address is at ESP+8
- Find POP, POP, RET and put its address in SEH



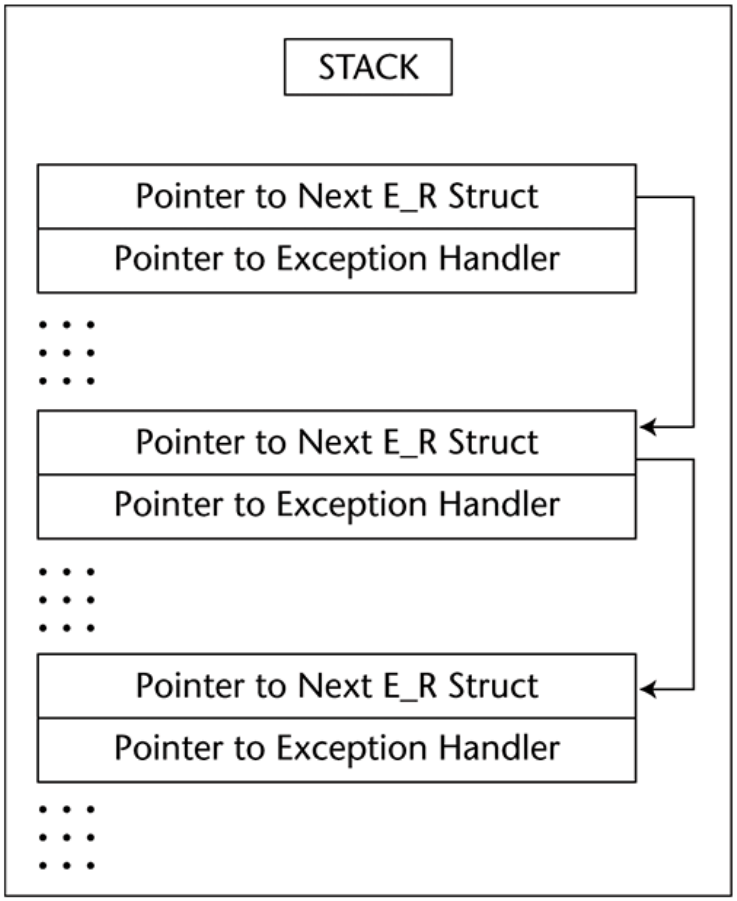
Frame-Based Exception Handlers

Exception Handler

- Code that deals with problems from within a running process
  - Such as access violation or divide by zero
- Frame-Based Exception Handler
  - Associated with a particular procedure
  - Each procedure sets up a new stack frame
- Every thread in a Win32 process has at least one frame-based exception handler

EXCEPTION_REGISTRATION

- Each thread's TEB has the address of the first EXCEPTION_REGISTRATION structure at fs:[0]
- When an exception occurs, the OS walks through the list until a suitable handler is found

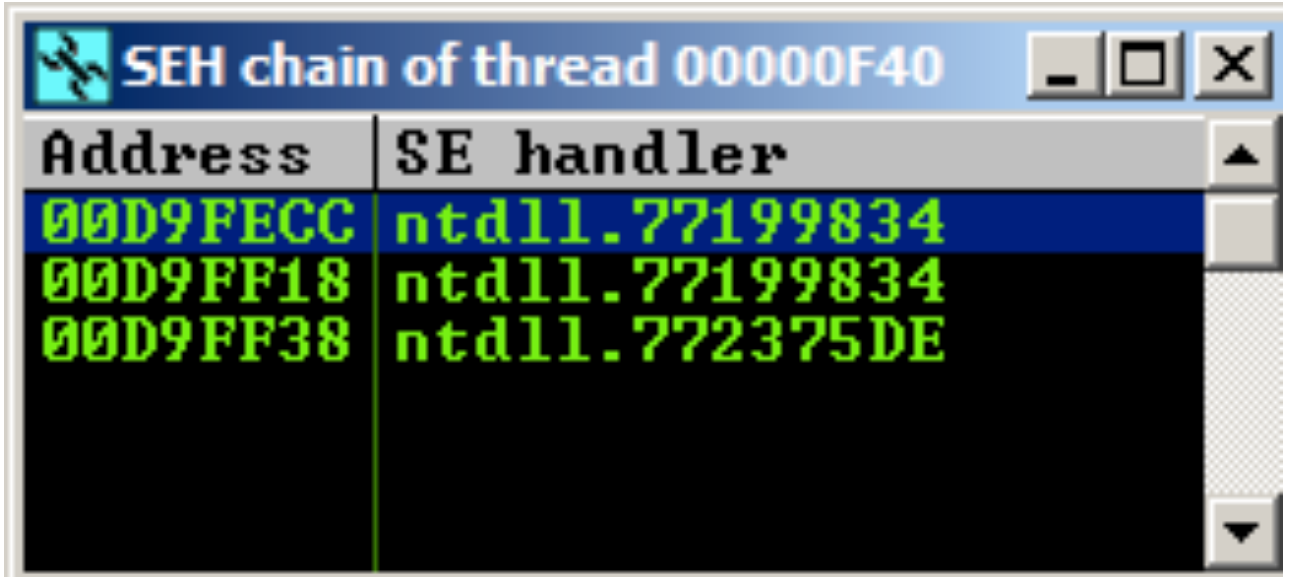


SEH Example Code

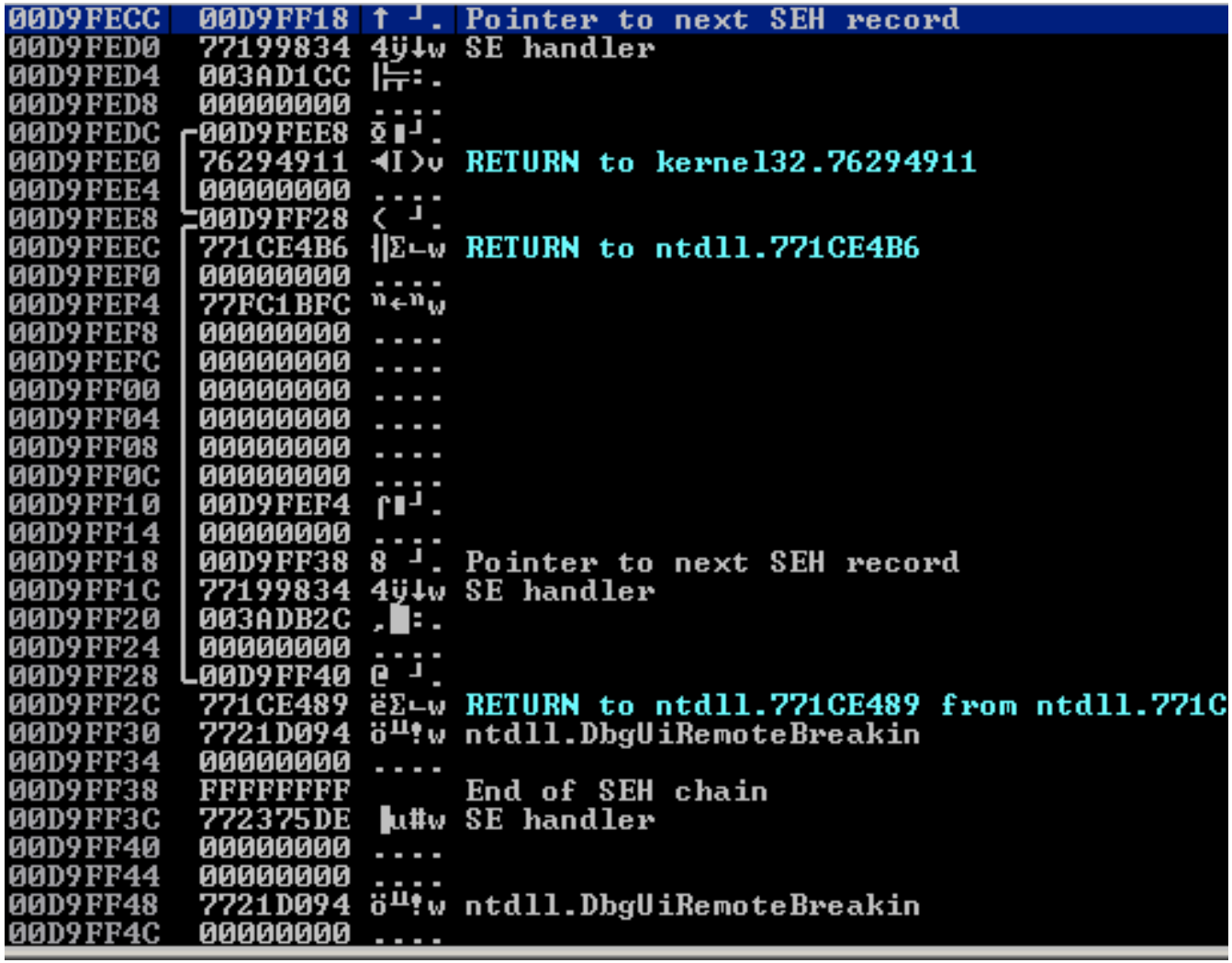
```
#include <stdio.h>
#include <windows.h>

DWORD MyExceptionHandler(void)
{
              printf("In exception handler....");
              ExitProcess(1);
              return 0;
}

int main()
{
               __try
               {
                    __asm
{
     // Cause an exception
                                  xor eax,eax
                                  call eax
}

             }
             __except(MyExceptionHandler())
             {
                     printf("oops...");
              }
              return 0;
}
```

SEH Chain in Immunity

- View, SEH Chain
- This is Notepad's SEH Chain



Follow Address in Stack



Exceptions in Stack Overflows

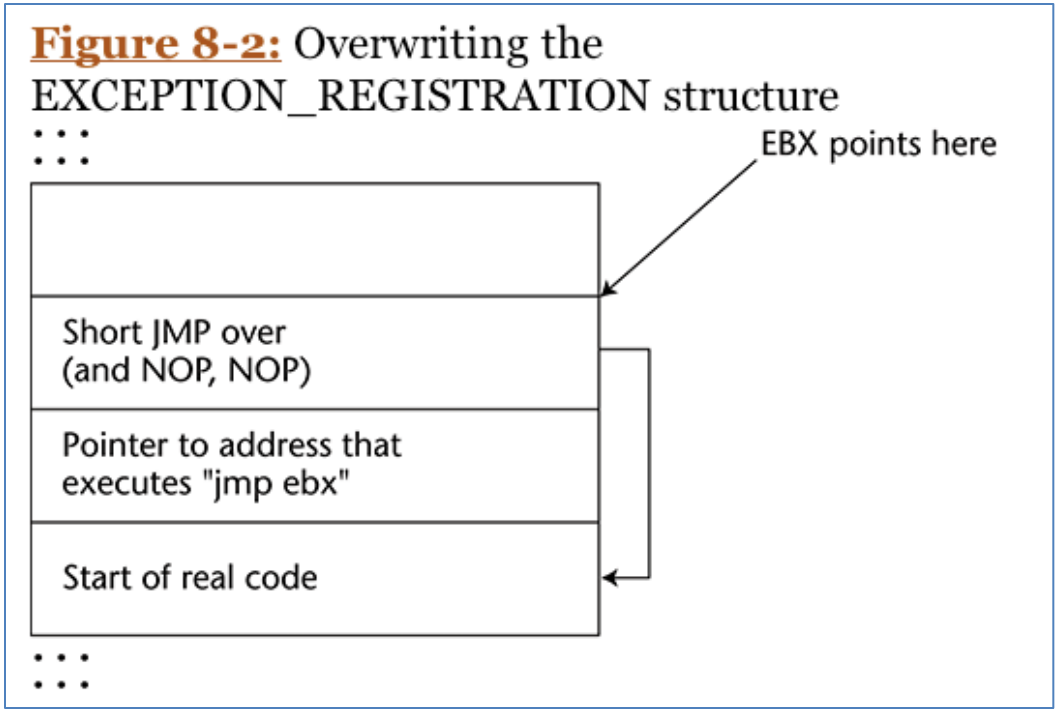
- Suppose we overflow the stack to overwrite the stored **return address**
- Unfortunately, we also overwrite other variables by accident (collateral damage)
- Those may lead to exceptions, as other instructions become invalid, before the function returns

Overwriting EXCEPTION_REGISTRATION

- To gain control of exception handling
- On Windows 2000 and Win XP (before SP1)
  - EBX points to the current EXCEPTION_REGISTRATION structure
- On later Windows versions, all the registers are zeroed when the SEH is called
  - In order to make exploitation more difficult

Old Windows Process

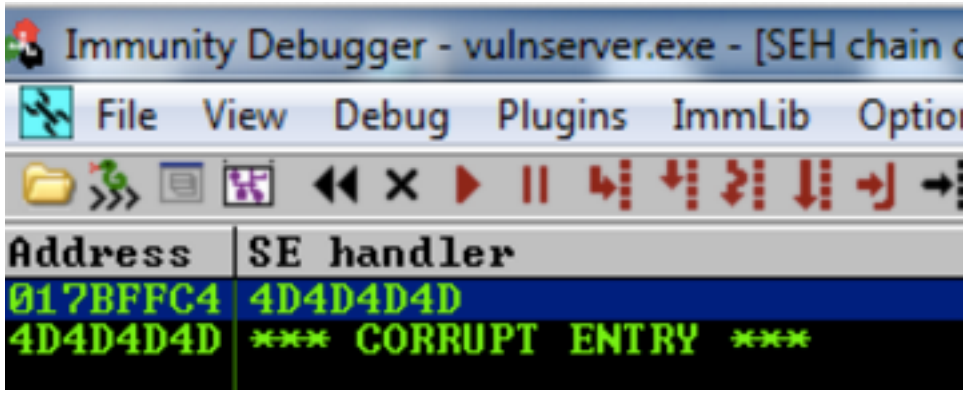
- At crash, EBX points to EXCEPTION_REGISTRATION structure
- Overwrite With "JMP EBX"



Modern Windows Process

- Stack at crash

- 3rd value points to EXCEPTION _REGISTRATION structure
- Use POP, POP, RET



SEH Protections

Win 2003 Server

- Attempts to ensure that handler is valid before using it, with these steps

1. If handler is on the stack -- INVALID
   - According to the TEB's entries FS:[4] and FS:{8]
2. If handler is in any loaded EXE or DLL – MAYBE
   - Otherwise VALID

Win 2003 Server

3. If the module is marked as "not allowed – INVALID
4. If a module has no "Load Configuration Directory", or one with a small size: VALID

Three Ways to Exploit SEH on Windows Server 2003

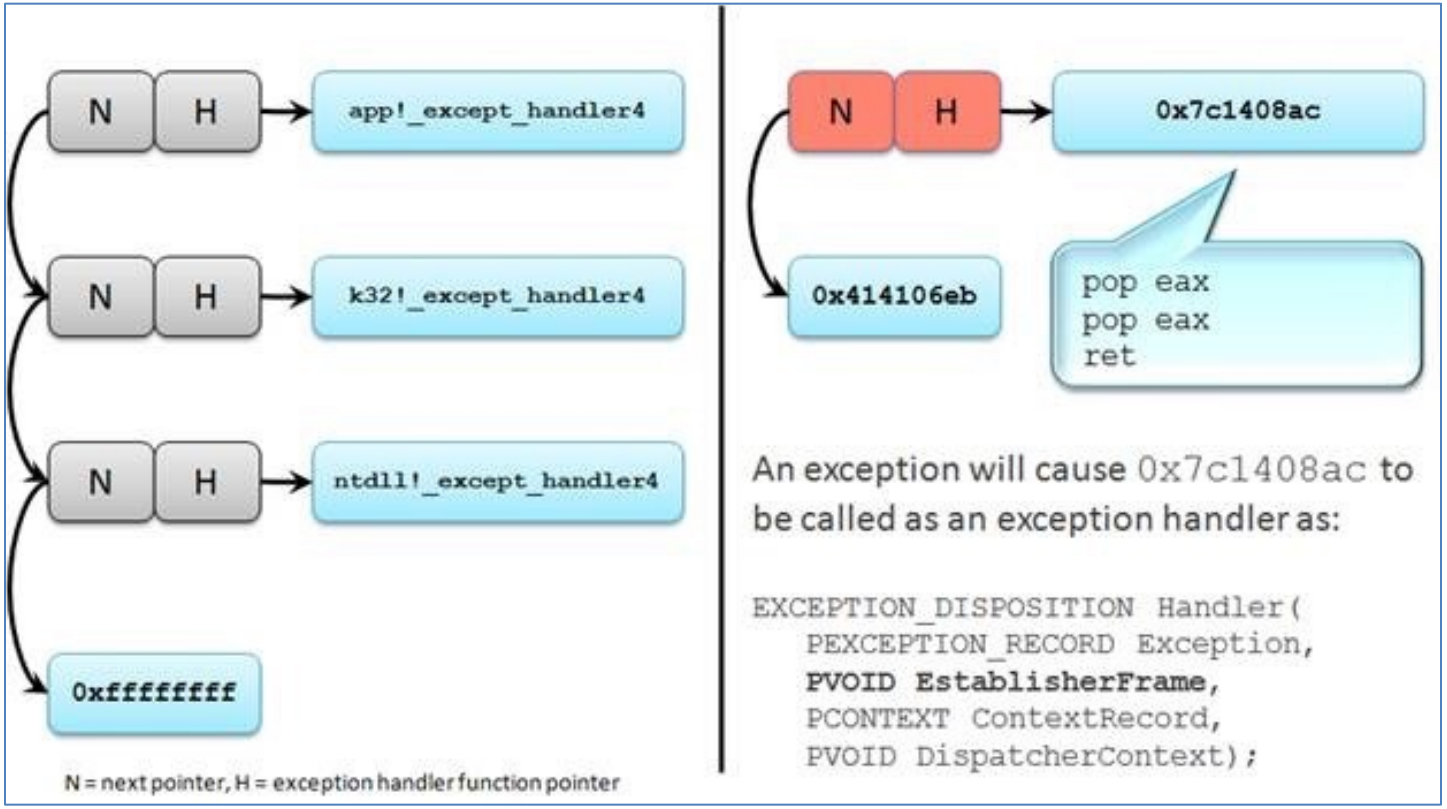
1. Abuse an existing handler
2. Use code in an address outside all modules
3. Use code in a module without a "Load Configuration Directory"

Defenses in Win Server 2008

Not in Book

Microsoft's Defenses

- Normal SEH attack (link Ch 8f)



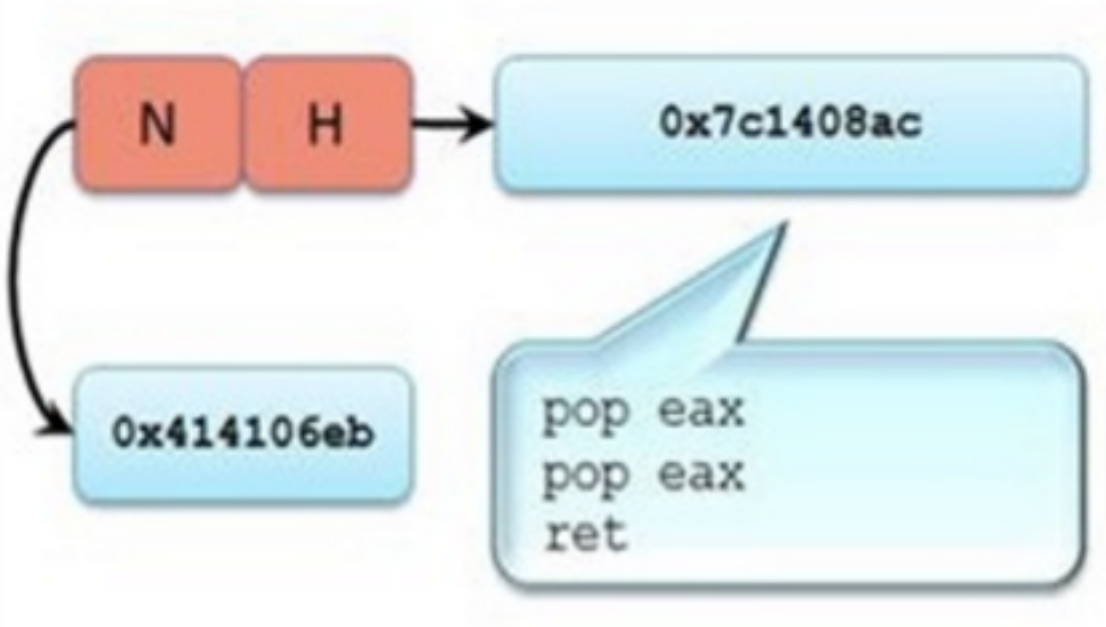
SAFESEH

- Microsoft added this compiler switch to Visual Studio 2003
- It creates a whitelist of exception handler addresses
- BUT this depends on the developer using the switch
  - All legacy code must be recompiled

SEHOP

- Verifies that the exception handler is intact before using it
- The pointer to the next handler comes before the pointer to the registration record
- So an exploit will usually damage the *Next pointer

EXCEPTION_REGISTRATION_RECORD

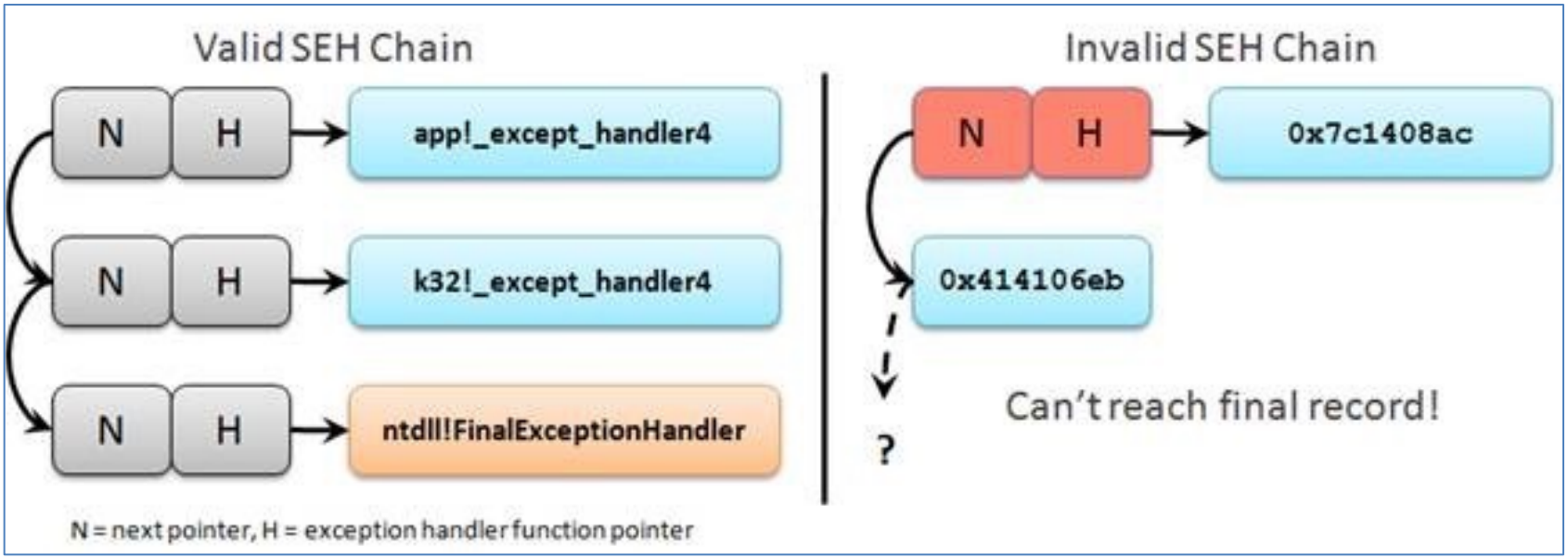



```
typedef struct _EXCEPTION_REGISTRATION_RECORD
{
    struct _EXCEPTION_REGISTRATION_RECORD *Next;
    PEXCEPTION_ROUTINE                    Handler;
} EXCEPTION_REGISTRATION_RECORD, *PEXCEPTION_REGISTRATION_RECORD;
```

How SEHOP Works

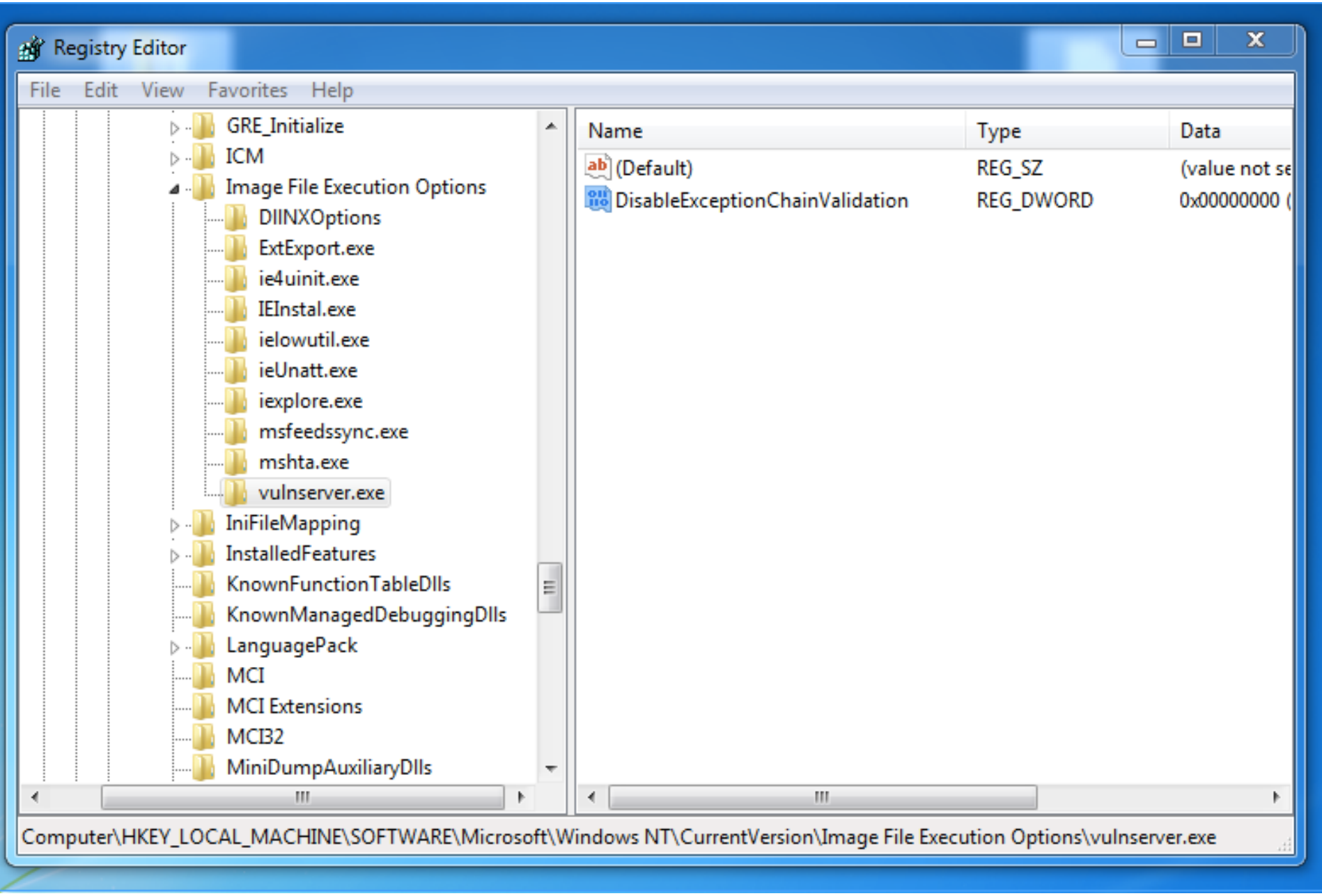
- Adds an extra registration record at the end of the list
- Walks the exception handler list to ensure that the added record can be reached
- Will detect corruption of *Next pointers

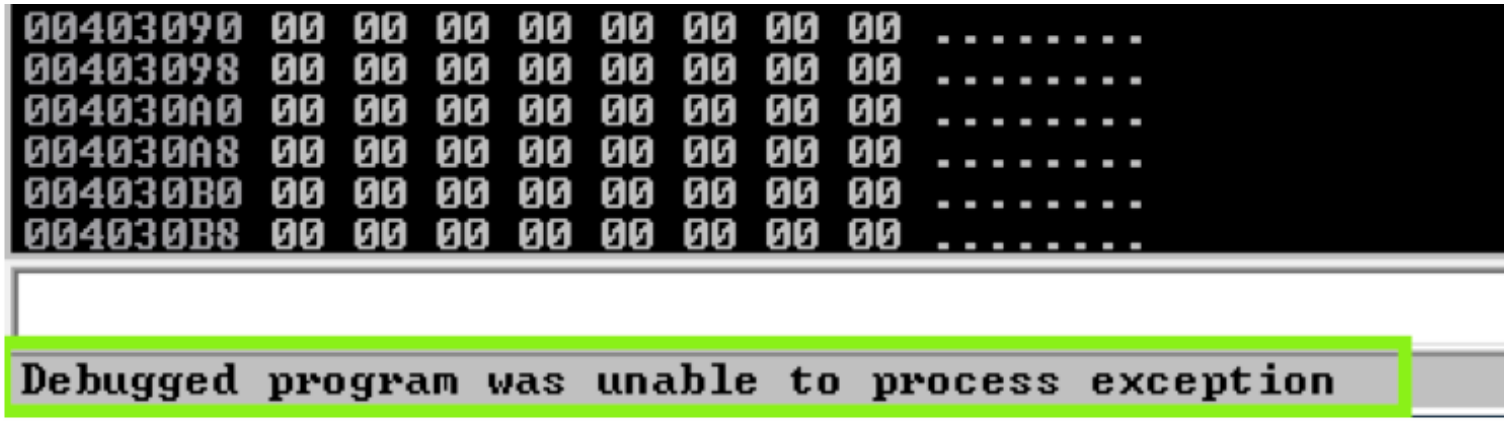
SEHOP


Valid SEH Chain
N
H
app!_except_handler4
N
H
k32!_except_handler4
N
H
ntdll!FinalExceptionHandler
N = next pointer, H = exception handler function pointer
Invalid SEH Chain
N
H
0x7c1408ac
0x414106eb
?
Can't reach final record!

Windows Versions

- SEHOP is enabled by default in Windows Server 2008
- Disabled by default in Windows Vista
  - Because it breaks some legacy code
- Also disabled by default on Windows 7
  - But it can be enabled in the Registry
  - Link Ch 8k


Registry Editor
File Edit View Favorites Help
GRE_Initialize
ICM
Image File Execution Options
DllNXOptions
ExtExport.exe
ie4uinit.exe
IEInstal.exe
ielowutil.exe
ieUnatt.exe
iexplore.exe
msfeedssync.exe
mshta.exe
vulnserver.exe
IniFileMapping
InstalledFeatures
KnownFunctionTableDlls
KnownManagedDebuggingDlls
LanguagePack
MCI
MCI Extensions
MCI32
MiniDumpAuxiliaryDlls
Name
Type
Data
(Default)
REG_SZ
(value not se
DisableExceptionChainValidation
REG_DWORD
0x00000000 (
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\vulnserver.exe


00403090 00 00 00 00 00 00 00 00 ....
00403098 00 00 00 00 00 00 00 00 ....
004030A0 00 00 00 00 00 00 00 00 ....
004030A8 00 00 00 00 00 00 00 00 ....
004030B0 00 00 00 00 00 00 00 00 ....
004030B8 00 00 00 00 00 00 00 00 ....
Debugged program was unable to process exception