Obfuscation to Defeat Static Analysis Using Cross-mode Coding

The cases missed in reverse engineering tools

Ke Sun **wildsator@gmail.com**
Xiaoning Li **ldpatchguard@gmail.com**

Source Seattle 2015

Agenda

- Obfuscation Overview
- Mode switch in 64-bit Windows
- Cross-mode coding obfuscation
- Example on static analysis tools
  - IDA / other disassembly tools
- Example on run-time binary translation tools
  - Dynamorio / Pintool
- Demo

Obfuscation Overview

- **<u>Why?</u>**

  Increase the difficulty and cost of reverse engineering.

- **<u>How?</u>**

  Mainstream focus on how to hide code using same machine code in target CPU.

Obfuscation Example - 1

- #9 of Flare-On Challenge 2015
  - Obfuscation with ROP

```
start_0         proc near               ; CODE XREF: start↓j

var_4           = dword ptr -4

                add     esp, 0FFFFFFFCh
                mov     dword ptr [esp], 1
                rol     dword ptr [esp], 16h
                sub     dword ptr [esp], 0FFFFEF6Fh
                retn
```

  - Mixed Data/Code

```
00401091                xor     eax, eax
00401093                jz      short near ptr loc_401095+1
00401095
00401095 loc_401095:
00401095                call    near ptr 9B4D323h
0040109A                push    ebp
```

```
00401096   mov     edx, eax
00401098   jz      short near ptr loc_4010A0+3
```

Obfuscation Example - 2

- EB FF pattern
  - Before

```
004010D3                 not     edi
004010D5                 mov     ecx, 6
004010DA                 push    edx
004010DB
004010DB loc_4010DB:                             ; CODE XREF: .text:004010F1↓j
004010DB                                         ; .text:loc_4010DB↑j
004010DB                 jmp     short near ptr loc_4010DB+1
```

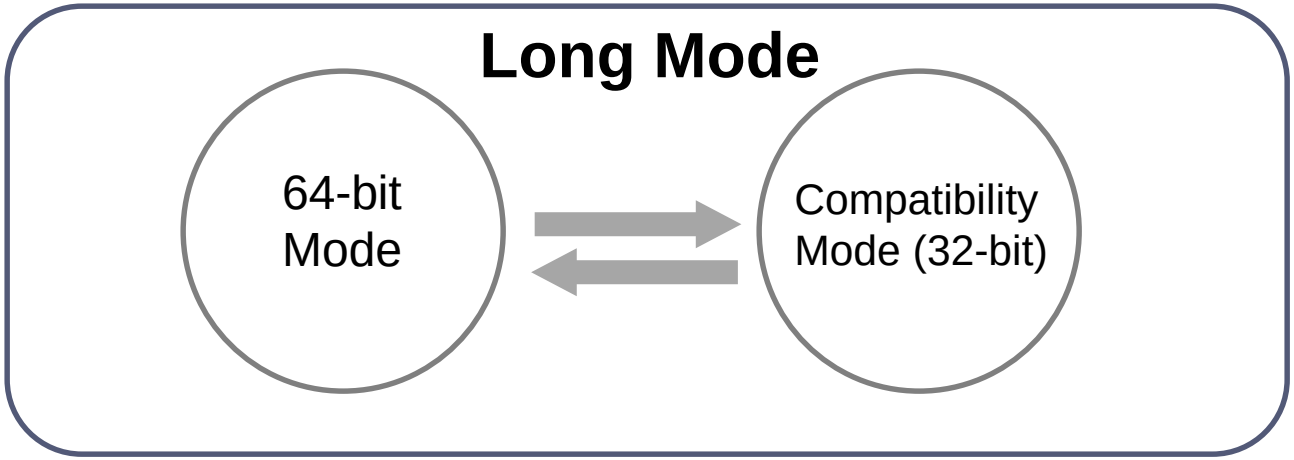
  - After

```
004010D3                 not     edi
004010D5                 mov     ecx, 6
004010DA                 push    edx
004010DA ; ---------------------------------------------------------------------
004010DB byte_4010DB     db 0EBh                 ; CODE XREF: .text:004010F1↓j
004010DC ; ---------------------------------------------------------------------
004010DC                 inc     eax
004010DE                 dec     eax
004010DF                 pop     eax
004010E0                 test    eax, eax
```

Mode Switch in 64-bit Windows

- All 64-bit versions of Windows support running 32-bit applications by providing the interfaces required to run unmodified 32-bit Windows applications on a 64-bit system.

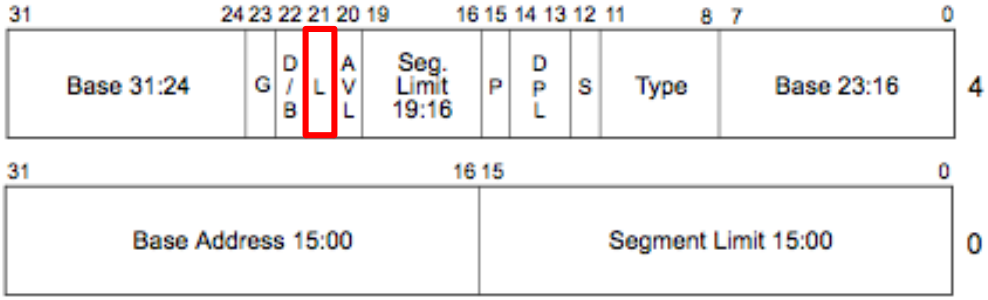
WoW64 (**W**indows 32-bit **o**n **W**indows **64**-bit)



Mode Switch in 64-bit Windows - 2

- CPU mode is determined by the “L” bit in the segment descriptor of the code segment (CS).

Segment Descriptors



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Mode Switch in 64-bit Windows - 3

- Every application has 2 code segments with overlapped 32-bit address space mapping
  - 0023: 32-bit (L=0)
  - 0033: 64-bit (L=1)

- Dynamic mode switch can be carried out by far branches to the corresponding segment

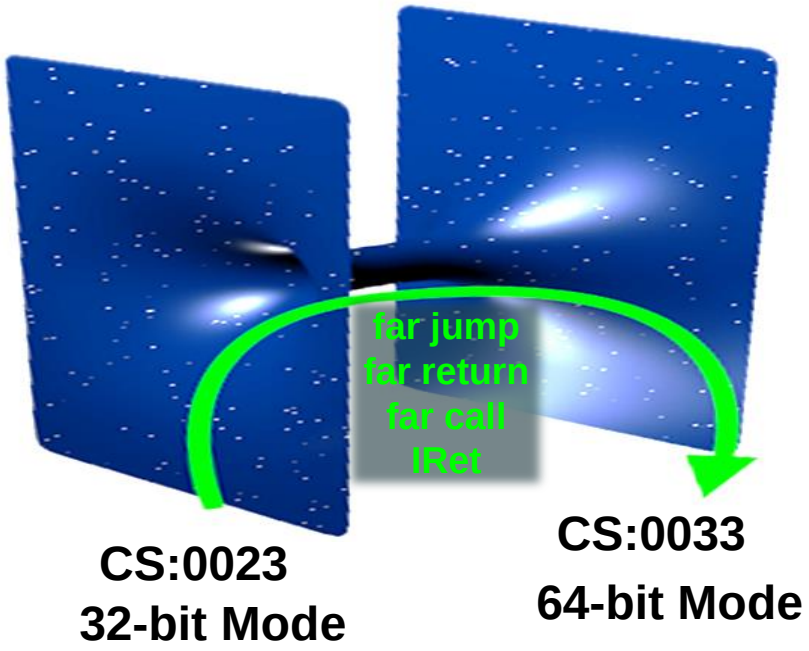
```
db 0eah
dd Enter64bit_Ret
db 033h
db 000h
```

jmp far 0033: Enter64bit_Ret

Switch from 32-bit to 64-bit mode

Mode Switch in 64-bit Windows - 4

- Far branch instructions can be used for mode switch
  - Far Jump
  - Far Call
  - Far Return
  - IRet



How can mode switching be used in obfuscation?

Cross-mode Coding Obfuscation

- Instruction compatibility
  - Compatible instructions

    Same binary code has same meaning under 32-bit/64-bit mode
  - Incompatible instructions

    Same binary code has different meaning under 32-bit/64-bit mode

MOV—Move

Opcode	Instruction	Op/	64-Bit	Compat/	Description
B0+ *rb ib*	MOV *r8, imm8*	OI	Valid	Valid	Move *imm8* to *r8*.
REX + B0+ *rb ib*	MOV *r8****, *imm8*	OI	Valid	N.E.	Move *imm8* to *r8*.
B8+ *rw iw*	MOV *r16, imm16*	OI	Valid	Valid	Move *imm16* to *r16*.
B8+ *rd id*	MOV *r32, imm32*	OI	Valid	Valid	Move *imm32* to *r32*.
REX.W + B8+ *rd io*	MOV *r64, imm64*	OI	Valid	N.E.	Move *imm64* to *r64*.
C6 */0 ib*	MOV *r/m8, imm8*	MI	Valid	Valid	Move *imm8* to *r/m8*.

compatible code

incompatible code

Compatible Instructions

- Compatible instructions has exactly the same binary & disassembly under 32-bit and 64-bit mode, but still can have different results due to different stack frame size.

64-bit mode

```
test!Callback64bit:
00000000`00b215c3 8bc4          mov     eax,esp
00000000`00b215c5 e800000000    call    test!Callback64bit+0x7
00000000`00b215ca 8bdc          mov     ebx,esp
00000000`00b215cc 2bc3          sub     eax,ebx
```

after code execution

eax = 8

32-bit mode

```
test!Callback64bit:
001615c3 8bc4          mov     eax,esp
001615c5 e800000000    call    test!Callback64bit+0x7
001615ca 8bdc          mov     ebx,esp
001615cc 2bc3          sub     eax,ebx
```

after code execution

eax = 4

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
8B */r*	MOV *r32,r/m32*	RM	Valid	Valid	Move *r/m32* to *r32*.
E8 *cd*	CALL *rel32*	M	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.

Incompatible Instructions

- Incompatible instructions, which are 32-bit or 64-bit specific, can be interpreted and executed differently under different mode.

64-bit mode

```
00000000`012a15a6 50        push    rax
00000000`012a15a7 53        push    rbx
00000000`012a15a8 4152      push    r10
00000000`012a15aa 488bc3    mov     rax,rbx
00000000`012a15ad 4c2bd0    sub     r10,rax
00000000`012a15b0 415a      pop     r10
00000000`012a15b2 5b        pop     rbx
00000000`012a15b3 58        pop     rax
```

32-bit mode

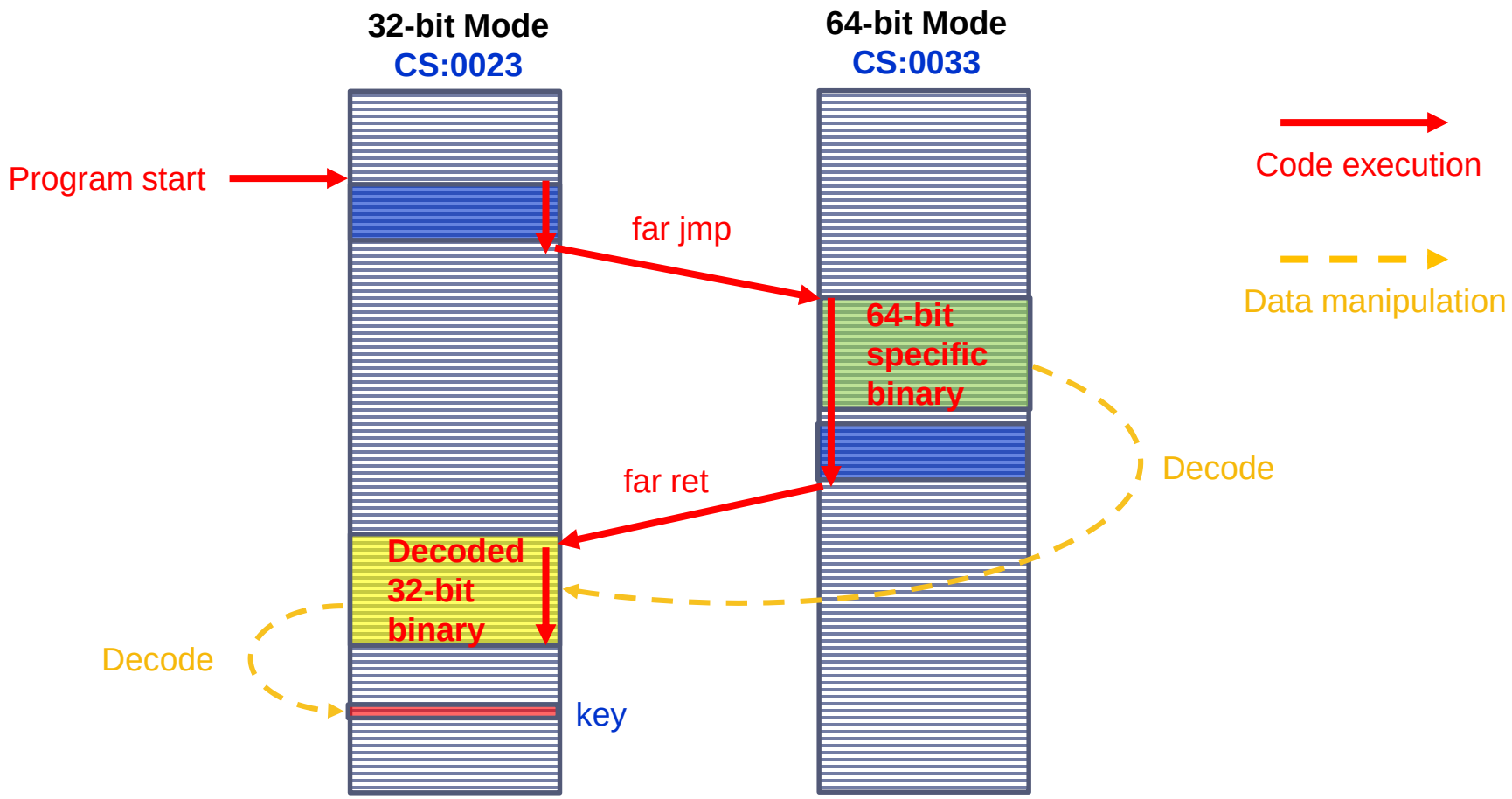
```
011a15a7 50        push    eax
011a15a8 53        push    ebx
011a15a9 41        inc     ecx
011a15aa 52        push    edx
011a15ab 48        dec     eax
011a15ac 8bc3      mov     eax,ebx
011a15ae 4c        dec     esp
011a15af 2bd0      sub     edx,eax
011a15b1 41        inc     ecx
011a15b2 5a        pop     edx
011a15b3 5b        pop     ebx
011a15b4 58        pop     eax
```

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
REX.W + 8B *r*	MOV *r64,r/m64*	RM	Valid	N.E.	Move *r/m64* to *r64*.
REX.W + 2B *r*	SUB *r64, r/m64*	RM	Valid	N.E.	Subtract *r/m64* from *r64*.

Can reverse engineering tools handle cross-mode codes properly?

Case Study

- A simple obfuscation using cross-mode coding :

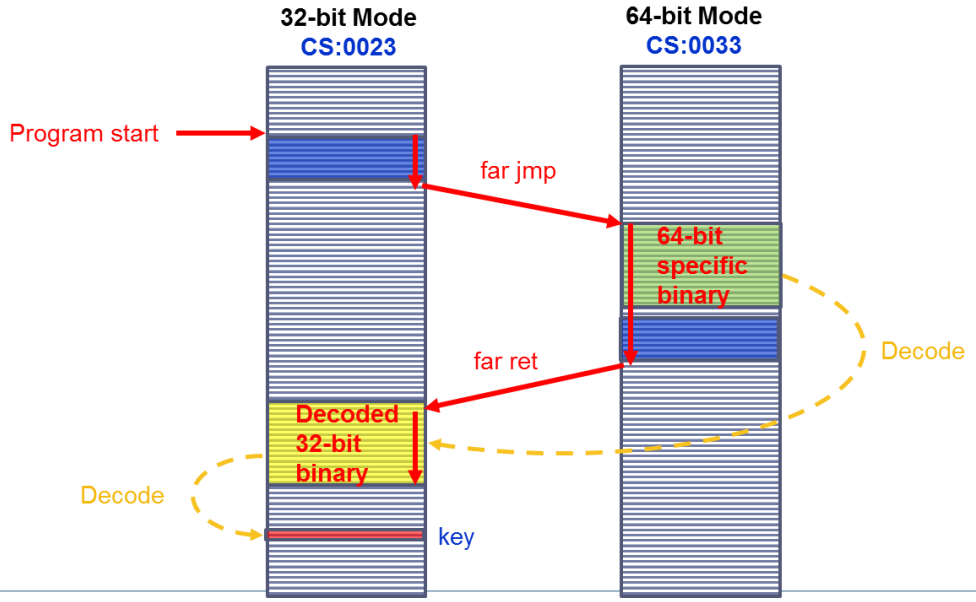


Case Study: example

- A simple obfuscation using cross-mode coding :

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\exe>xmodeobf.exe

Encoded Key = ╚╚Gà└^▲┘US↨¶‼^ЛOO
CS selctor = 23
CS selctor = 33
CS selctor = 23
Decoded key by Xmode Obfuscation = SOURCE Seattle 2015
```



Case Study: 32-bit IDA Pro

- 32-bit IDA Pro can not disassembly correctly the 64-bit code:

Run-time (64-bit mode)

```
013c16f6 50                push    rax
013c16f7 53                push    rbx
013c16f8 56                push    rsi
013c16f9 57                push    rdi
013c16fa be30173c01        mov     esi,offset test!UnpackCode+0x3b
013c16ff 48c7c103000000    mov     rcx,3
013c1706 488b1e            mov     rbx,qword ptr [rsi]
013c1709 48b814c75001e271a75e mov rax,5EA771E20150C714h
013c1713 4833d8            xor     rbx,rax
013c1716 48c1c303          rol     rbx,3
013c171a 48891e            mov     qword ptr [rsi],rbx
013c171d 4883c608          add     rsi,8
013c1721 48ffc9            dec     rcx
013c1724 75e0              jne     test!UnpackCode+0x11
013c1726 5f                pop     rdi
013c1727 5e                pop     rsi
013c1728 5b                pop     rbx
013c1729 58                pop     rax
```

Static DA (32-bit mode)

```
00411744 sub_411744      proc near
00411744                 jmp     far ptr 33h:41100Fh
00411744 sub_411744      endp

mov     esi, offset dword_411730
dec     eax
mov     ecx, 3
dec     eax
mov     ebx, [esi]
dec     eax
mov     eax, 150C714h
loop    loc_411782
cmpsd
pop     esi
dec     eax
xor     ebx, eax
dec     eax
rol     ebx, 3
dec     eax
mov     [esi], ebx
dec     eax
add     esi, 8
dec     eax
dec     ecx
jnz     short loc_411706
```

Case Study: 64-bit IDA Pro

- 64-bit IDA Pro still disassembly the cross-mode binary in 32-bit mode, and can not correctly handle the mode-switching far jump.

```
00411744 sub_411744      proc near               ; CODE XREF: sub_411032↑j
00411744                 jmp     far ptr byte_41133F
00411744 sub_411744      endp

0041133F byte_41133F     db 0D1h dup(0CCh)       ; CODE XREF: sub_411744↓J
00411410
00411410 ; =============== S U B R O U T I N E =======================================
00411410
```

How about other disassembly tools?

Case Study: Radare (32-bit)

- Radare is a command-line framework for reverse engineering on multiple platforms.
- Radare in 32-bit mode can not disassembly correctly the 64-bit spedific code embedded in 32-bit program.

64-bit mode

```
0x00411744    ea            invalid
0x00411745    0f104100      movups xmm0, xmmword [rcx]
0x00411749    3300          xor eax, dword [rax]
```

```
488b1e          mov rbx, qword [rsi]
48b814c75001.   movabs rax, 0x5ea771e20150c714
4833d8          xor rbx, rax
48c1c303        rol rbx, 3
48891e          mov qword [rsi], rbx
4883c608        add rsi, 8
48ffc9          dec rcx
75e0            jne 0x411772
5f              pop rdi
5e              pop rsi
5b              pop rbx
58              pop rax
b89c174100      mov eax, 0x41179c
```

32-bit mode

```
48              dec eax
8b1e            mov ebx, dword [esi]
48              dec eax
b814c75001      mov eax, 0x150c714
e271            loop 0x4117ee
a7              cmpsd dword [esi], dword ptr es:[edi]
5e              pop esi
48              dec eax
33d8            xor ebx, eax
48              dec eax
c1c303          rol ebx, 3
48              dec eax
891e            mov dword [esi], ebx
48              dec eax
83c608          add esi, 8
48              dec eax
ffc9            dec ecx
75e0            jne 0x411772
5f              pop edi
5e              pop esi
5b              pop ebx
58              pop eax
b89c174100      mov eax, 0x41179c
```

Case Study: Radare (64-bit)

- Radare in 64-bit mode also has trouble correctly disassembly 32-bit specific code.

64-bit mode

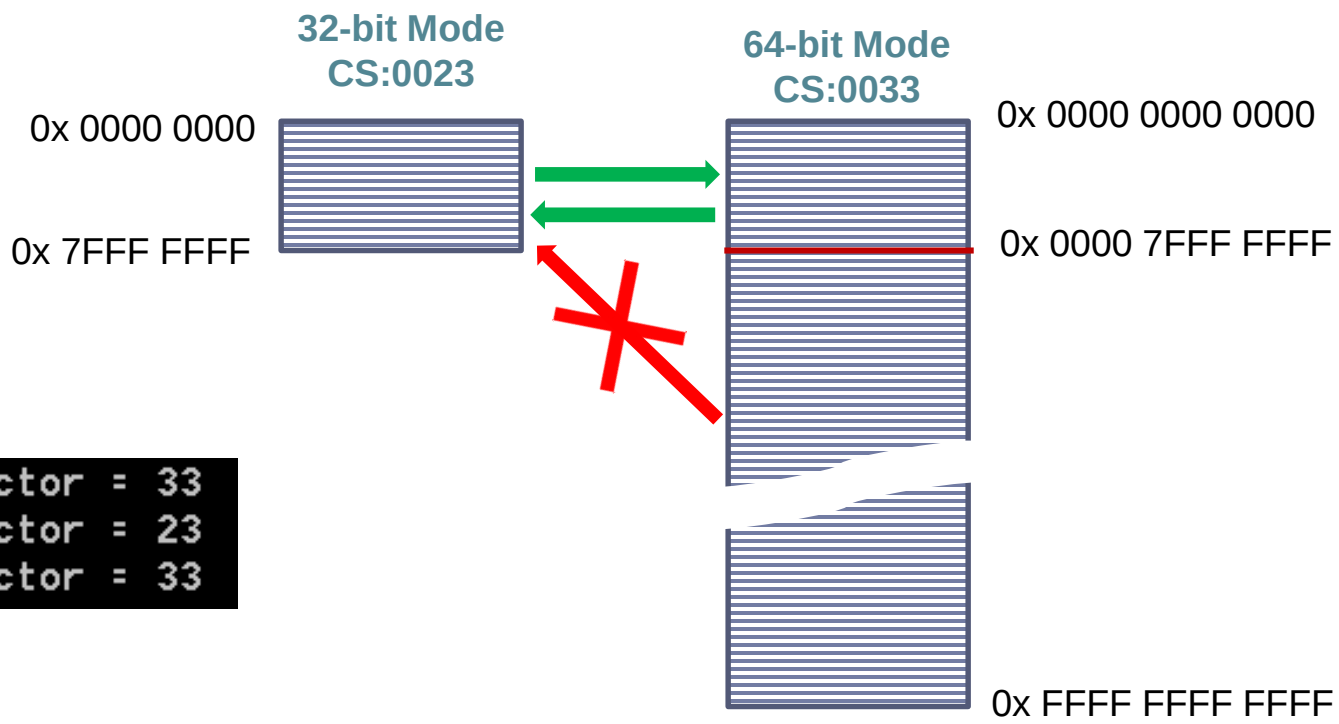
```
55              push rbp
8bec            mov ebp, esp
81ec18010000    sub esp, 0x118
53              push rbx
56              push rsi
57              push rdi
8dbde8feffff    lea edi, [rbp - 0x118]
b946000000      mov ecx, 0x46                    ; 'F'
b8cccccccc      mov eax, 0xcccccccc
f3ab            rep stosd dword [rdi], eax
a12480410033.   movabs eax, dword [0x4589c53300418024]
fc              cld
c745c4000000.   mov dword [rbp - 0x3c], 0
33c0            xor eax, eax
```

32-bit mode (by IDA

```
A1 24 80 41 00  push    ebp
                mov     ebp, esp
                sub     esp, 118h
                push    ebx
                push    esi
                push    edi
                lea     edi, [ebp+var_118]
                mov     ecx, 46h
                mov     eax, 0CCCCCCCCh
                rep stosd
                mov     eax, __security_cookie
                xor     eax, ebp
                mov     [ebp+var_4], eax
                mov     [ebp+var_3C], 0
                xor     eax, eax
```

Starting with 64-bit mode

- Starting with 64-bit code and cross into 32-bit mode also works.
- Need to make sure 64-bit program only use the first 2G space




```
CS selector = 33
CS selector = 23
CS selector = 33
```

How about run-time tools?

DynamoRio (32-bit)

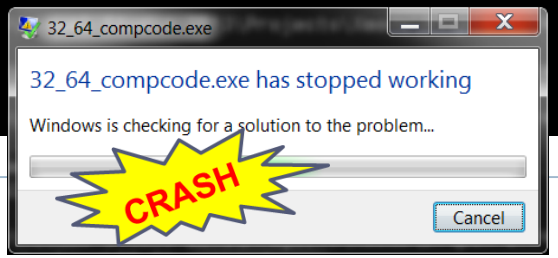
- Use 32-bit DynamoRio on obfuscated code with 32-bit starting mode:
  - DynamoRio executed the embedded 64-bit code as 32-bit, and crash later due to address violation

Execute by Command Line

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\exe>32_64_compco
de.exe
CS selctor = 23
Return value under32bit = 4
CS selector = 33
Return value under64bit = 8
```

Execute by DynamoRio (32-bit)

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\BT\DynamoRIO-Win
dows-5.1.0-RC1\bin32>drrun.exe ..\..\..\exe\32_64_compcode.exe
CS selctor = 23
Return value under32bit = 4
CS selector = 23
Return value under64bit = 4
```



DynamoRio (64-bit)

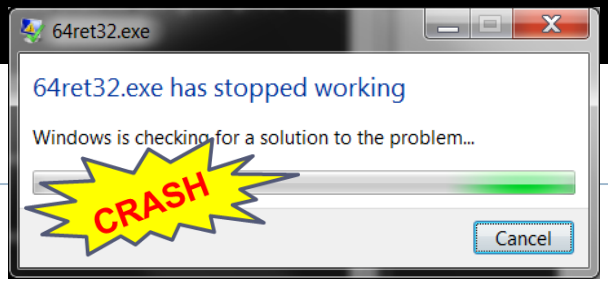
- Use 64-bit DynamoRio on obfuscated code with 64-bit starting mode:
  - 64-bit DynamoRio crashed at the first cross mode switch to 32-bit

Execute by Command Line

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\exe>64ret32.exe
CS selector = 33
CS selector = 23
CS selector = 33
```

Execute by DynamoRio (64-bit)

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\BT\DynamoRIO-Win
dows-5.1.0-RC1\bin64>drrun.exe ..\..\..\exe\64ret32.exe
CS selector = 33
```



Pintool (32-bit code)

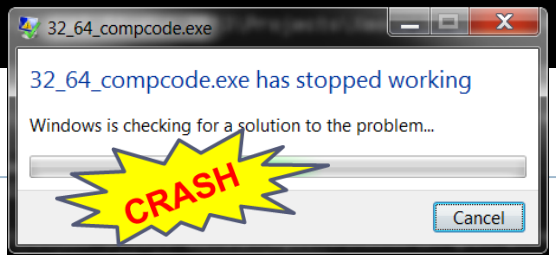
- Use Pintool on obfuscated code with 32-bit starting mode:
  - Pintool crashed on the first cross mode switch to 64-bit

Execute by Command Line

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\exe>32_64_compco
de.exe
CS selctor = 23
Return value under32bit = 4
CS selector = 33
Return value under64bit = 8
```

Execute by Pintool

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\BT\pin-2.14-7131
3-msvc12-windows>pin.exe -- ..\..\exe\32_64_compcode.exe
CS selctor = 23
Return value under32bit = 4
```



Pintool (64-bit code)

- Use Pintool on obfuscated code with 64-bit starting mode:
  - Pintool stopped at first cross mode switch to 32-bit with message:
    "Pin doesn't support FAR RET with transfer to different code segment"

Execute by Command Line

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\exe>64ret32.exe
CS selector = 33
CS selector = 23
CS selector = 33
```

Execute by Pintool

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\BT\pin-2.14-7131
3-msvc12-windows>pin.exe -- ..\..\exe\64ret32.exe
CS selector = 33
E: Pin doesn't support FAR RET (IP 0x0004011c3) with transfer to different code
segment (from 0x0033 to 0x0000)
```

Segment Selectors Obfuscation

- Segment selectors other than CS:0033 & CS:0023 are available for cross-mode far branches.

```
db 0eah
dd Enter64bit_Ret
;cs:32
db 032h
db 000h
```

```
CS selector = 23
CS selector = 33
CS selector = 23
```

```
eip     dc158a
cs      23
efl     244
esp     29fdf4
ss      2b
```

```
00dc1588 c3                ret
test!Enter64bit:
00dc1589 cc                int     3
00dc158a ea0a10dc003200    jmp     0032:00DC100A
test!Backto32bit:
00dc1591 50                push    eax
00dc1592 50                push    eax
```

```
cs      33
ds      2b
es      2b
fs      53
gs      2b
ss      2b
```

```
test!ILT+5(_Enter64bit_Ret):
00000000`00dc100a e976050000      jmp     test!Enter64bit_Ret
test!ILT+10(_wmain):
00000000`00dc100f e9dc030000      jmp     test!wmain (00000000
test!ILT+15(__RTC_GetErrDesc):
00000000`00dc1014 e9c71a0000      jmp     test!_RTC_GetErrDesc
test!ILT+20(___crtSetUnhandledExceptionFilter):
00000000`00dc1019 e9942c0000      jmp     test!_crtSetUnhandle
```

Issues in Debugger

- Debuggers have issue debugging the x-mode coded exe:
  - WinDbg (64-bit) can not single step the code written under x-mode
  - OllyDbg (32-bit) can not continue debug after mode switch

```
0041181c b914000000   mov   ecx,14h
00411821 8a06         mov   al,byte ptr [esi]
00411823 32c1         xor   al,cl
00411825 c0c002       rol   al,2
00411828 8807         mov   byte ptr [edi],al
0041182a 46           inc   esi
0041182b 47           inc   edi
0041182c 49           dec   ecx
0041182d 75f2         jne   test!UnpackCode+0x3f (00411821)
0041182f 90           nop
00411830 90           nop
00411831 90           nop
00411832 90           nop
00411833 90           nop
00411834 90           nop
00411835 90           nop
00411836 90           nop
00411837 83c40c       add   esp,0Ch
0041183a c3           ret
```

```
0041181b c3           ret
0041181c b914000000   mov   ecx,14h
00411821 8a06         mov   al,byte ptr [esi]
00411823 32c1         xor   al,cl
00411825 c0c002       rol   al,2
00411828 8807         mov   byte ptr [edi],al
0041182a 46           inc   esi
0041182b 47           inc   edi
0041182c 49           dec   ecx
0041182d 75f2         jne   test!UnpackCode+0x3f (00411821)
0041182f 90           nop
00411830 90           nop
00411831 90           nop
00411832 90           nop
00411833 90           nop
00411834 90           nop
00411835 90           nop
00411836 90           nop
00411837 83c40c       add   esp,0Ch
```

OllyDbg

```
004117C4 Enter64bit   EA 0F104100   JMP FAR 0033:0041100F        Far jump
004117CB Backto32bit  50            PUSH EAX
004117CC              50            PUSH EAX
004117CD              8B4424 10     MOV EAX,DWORD PTR SS:[ESP+10]
004117D1              894424 0C     MOV DWORD PTR SS:[ESP+C],EAX
004117D5              C74424 10 2   MOV DWORD PTR SS:[ESP+10],23
```

```
76C212EA              83C4 04       ADD ESP,4
76C212ED              C2 1400       RETN 14
76C212F0              E8 07000000   CALL kernel32.WriteConsoleA
76C212F5              ^EB D5        JMP SHORT kernel32.76C212CC
76C212F7              90            NOP
```

Demo

Summary

- Currently most reverse engineering tools can not correctly analyze cross-mode coded program.
- Common binary translation tools also have trouble handling binaries with run-time mode switch.
- Mainstream debuggers either have single stepping issue or simply incapable of debugging when debug cross-mode codes.
- Cross-mode obfuscation can defeat static analysis tools, cause issue for BT tools and debuggers and can be used for BT detection.

Thank You!

wildsator@gmail.com
ldpatchguard@gmail.com

Thanks to Haifei Li and to Rodrigo Branco ‘s Review!

Reference

- The Performance Cost of Shadow Stacks and Stack Canaries. Thurston H.Y. Dang, Petros Maniatis, David Wagner. ASIACCS 2015
- Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications. Felix Schuster, Thomas Tendyck, Christopher Liebchen, Lucas Davi, Ahmad-Reza Sadeghi, Thorsten Holz. 36th IEEE Symposium on Security and Privacy (Oakland), May 2015
- Exploring Control Flow Guard in Windows 10. Jack Tang. Trend Micro Threat Solution Team, 2015
- ROP is Still Dangerous: Breaking Modern Defenses. Nicholas Carlini and David Wagner. 23rd USENIX Security Symposium, Berkeley 2014
- Windows 10 Control Flow Guard Internals. MJ0011, POC 2014
- Write Once, Pwn Anywhere. Yu Yang. Blackhat 2014
- Hardware-Assisted Fine-Grained Control-Flow Integrity: Towards Efficient Protection of Embedded Systems Against Software Exploitation. Lucas Davi, Patrick Koeberl, and Ahmad-Reza Sadeghi. DAC 2014
- Transparent ROP Exploit Mitigation Using Indirect Branch Tracing. Vasilis Pappas, Michalis Polychronakis, and Angelos D. Keromytis. Columbia University, 22nd USENIX Security Symposium 2013
- kBouncer: Efficient and Transparent ROP Mitigation. Vasilis Pappas. Columbia University 2012
- Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters. Liwei Yuan, Weichao Xing, Haibo Chen, Binyu Zang. APSYS 2011
- Transparent Runtime Shadow Stack: Protection against malicious return address modifications. Saravanan Sinnadurai, Qin Zhao, and Weng-Fai Wong. 2008
- Control-Flow Integrity Principles, Implementations, and Applications. Martín Abadi, Mihai Budiu, Úlfar Erlingsson, Jay Ligatti. CCS2005
- IROP – interesting ROP gadgets, Xiaoning Li/Nicholas Carlini, Source Boston 2015