Proj 12x: Anti-Disassembly (Lab 15-1) (15 pts.)

What you need:

- A Windows machine with IDA Pro. The Win 2008 Server virtual machine we've been using works fine.

Purpose

Practice customizing IDA Pro disassembly to overcome the anti-disassembly techniques in chapter 15.

Indications of Anti-Disassembly

Open **Lab15-01.exe** file in IDA.

Click **Options**, **General**. Check "**Line Prefixes**". Enter a "Number of opcode bytes" of **6** and click **OK**.

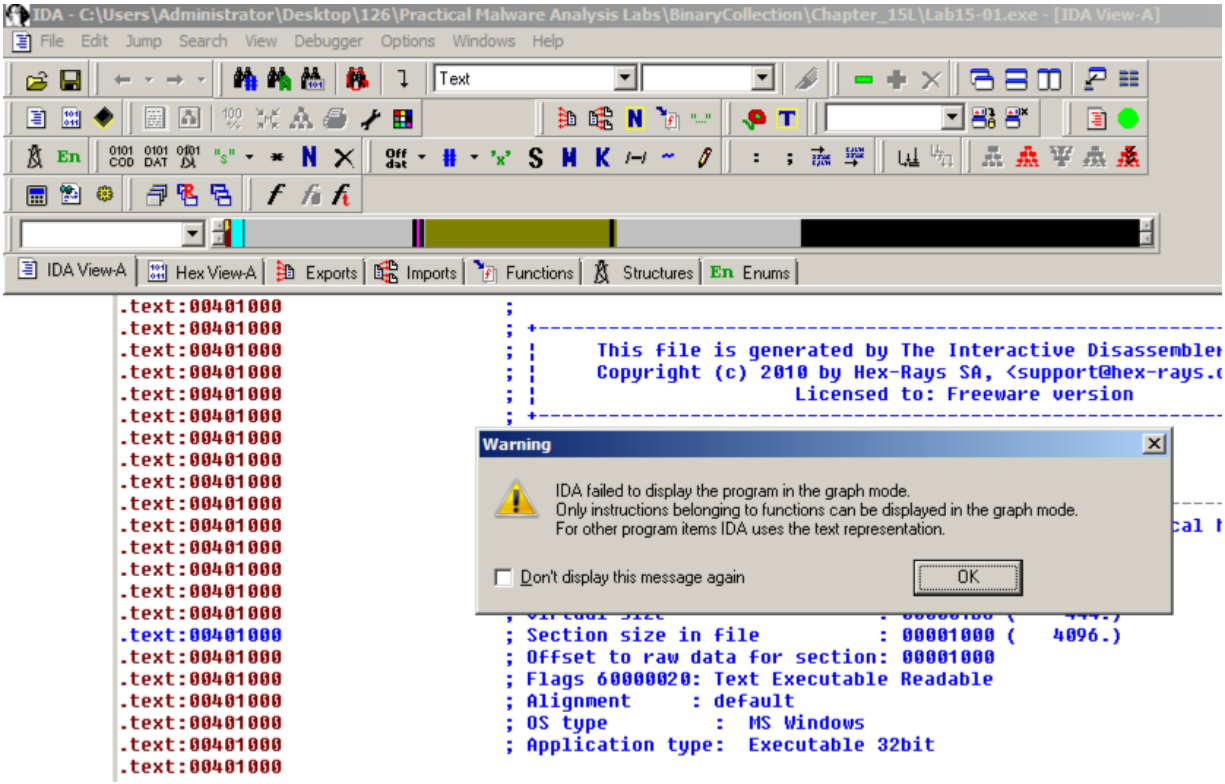
Maximize the "IDA View-A" window.

Notice that the display is not in Graph Mode, but just a long linear chart of disassembled code.

Click in the "IDA View-A" window. Press the **SPACEBAR**.

A Warning box appears, saying that IDA can't display the code in graph mode because it can't identify the functions, as shown below.

This is a clue that something is confusing IDA Pro.



In IDA, scroll down past the blue header comments to see the actual code.

Notice that the "CODE XREF" labels are in red, as shown below. That indicates that the actual reference points inside this instruction, not at its starting byte. This is another indication that IDA has not beem able to correctly disassemble the code.

```
.text:00401000
.text:00401000                  ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401000                  _main:                                  ; CODE XREF: start+DE↓p
.text:00401000 55                       push    ebp
.text:00401001 8B EC                    mov     ebp, esp
.text:00401003 53                       push    ebx
.text:00401004 56                       push    esi
.text:00401005 57                       push    edi
.text:00401006 83 7D 08 02              cmp     dword ptr [ebp+8], 2
.text:0040100A 75 52                    jnz     short loc_40105E
.text:0040100C 33 C0                    xor     eax, eax
.text:0040100E 74 01                    jz      short near ptr loc_401010+1
.text:00401010
.text:00401010                  loc_401010:                             ; CODE XREF: .text:0040100E↑j
.text:00401010 E8 8B 45 0C 8B           call    near ptr 8B4C55A0h
.text:00401015 48                       dec     eax
.text:00401016 04 0F                    add     al, 0Fh
.text:00401018 BE 11 83 FA 70           mov     esi, 70FA8311h
.text:0040101D 75 3F                    jnz     short loc_40105E
.text:0040101F 33 C0                    xor     eax, eax
.text:00401021 74 01                    jz      short near ptr loc_401023+1
.text:00401023
.text:00401023                  loc_401023:                             ; CODE XREF: .text:00401021↑j
.text:00401023 E8 8B 45 0C 8B           call    near ptr 8B4C55B3h
.text:00401028 48                       dec     eax
.text:00401029 04 0F                    add     al, 0Fh
.text:0040102B BE 51 02 83 FA           mov     esi, 0FA830251h
```

Fixing the Code at 401011

The first problem is easy to see: the instruction at address 40100E is a **jz** to address 401010+1, or 401011. Since the preceding instruction was **xor eax, eax**, the condition is always true, so the code will always skip over the byte at address 40100E.

To correct the disassembly, in the left column, click any of the **401010** addresses. All three of them turn yellow.

On the keyboard, press **d** to convert these five bytes to data. A "Please confirm" box pops up. Click **Yes**.

The code now shows five bytes of data after the **jz** instruction, as shown below.

```
.text:00401006 83 7D 08 02              cmp     dword ptr [ebp+8], 2
.text:0040100A 75 52                    jnz     short loc_40105E
.text:0040100C 33 C0                    xor     eax, eax
.text:0040100E 74 01                    jz      short near ptr unk_401011
.text:0040100E                  ; ---------------------------------------------------------------------------
.text:00401010 E8                       db 0E8h
.text:00401011 8B               unk_401011      db  8Bh ; ï             ; CODE XREF: .text:0040100E↑j
.text:00401012 45                       db  45h ; E
.text:00401013 0C                       db  0Ch
.text:00401014 8B                       db  8Bh ; ï
.text:00401015                  ; ---------------------------------------------------------------------------
.text:00401015 48                       dec     eax
.text:00401016 04 0F                    add     al, 0Fh
.text:00401018 BE 11 83 FA 70           mov     esi, 70FA8311h
```

The **jz** resumes execution at location 401011, so we must tell IDA to interpret that address as code.

In the left column, click the **401011** address. On the keyboard, press **c** to convert these five bytes to code.

Three bytes starting at 401011 are now interpreted as a **mov** instruction, as shown below.

However, the byte at 401014 is still interpreted as data, and that makes the disassembly below it inaccurate as well.

```
.text:00401006 83 7D 08 02              cmp     dword ptr [ebp+8], 2
.text:0040100A 75 52                    jnz     short loc_40105E
.text:0040100C 33 C0                    xor     eax, eax
.text:0040100E 74 01                    jz      short loc_401011
.text:0040100E                  ; ---------------------------------------------------------------------------
.text:00401010 E8                       db 0E8h
.text:00401011                  ; ---------------------------------------------------------------------------
.text:00401011
.text:00401011                  loc_401011:                             ; CODE XREF: .text:0040100E↑j
.text:00401011 8B 45 0C                 mov     eax, [ebp+0Ch]
.text:00401014 8B                       db  8Bh ; ï
.text:00401015                  ; ---------------------------------------------------------------------------
.text:00401015 48                       dec     eax
.text:00401016 04 0F                    add     al, 0Fh
.text:00401018 BE 11 83 FA 70           mov     esi, 70FA8311h
```

In the left column, click the **401014** address. On the keyboard, press **c** to convert these five bytes to code. A "Please confirm" box pops up. Click **Yes**.

This guides IDA to disassemble another instruction correctly, but leaves another byte abandoned and interpreted as data at 401017, as shown below.

```
.text:00401006 83 7D 08 02                cmp     dword ptr [ebp+8], 2
.text:0040100A 75 52                      jnz     short loc_40105E
.text:0040100C 33 C0                      xor     eax, eax
.text:0040100E 74 01                      jz      short loc_401011
.text:0040100E                    ; ---------------------------------------------------------------------------
.text:00401010 E8                         db 0E8h
.text:00401011                    ; ---------------------------------------------------------------------------
.text:00401011
.text:00401011                    loc_401011:                             ; CODE XREF: .text:0040100E↑j
.text:00401011 8B 45 0C                   mov     eax, [ebp+0Ch]
.text:00401014 8B 48 04                   mov     ecx, [eax+4]
.text:00401017 0F                         db  0Fh
.text:00401018                    ; ---------------------------------------------------------------------------
.text:00401018 BE 11 83 FA 70             mov     esi, 70FA8311h
.text:0040101D 75 3F                      jnz     short loc_40105E
.text:0040101F 33 C0                      xor     eax, eax
.text:00401021 74 01                      jz      short near ptr loc_401023+1
```

In the left column, click the **401017** address. On the keyboard, press **c** to convert these five bytes to code. A "Please confirm" box pops up. Click **Yes**.

Finally, IDA shows the correct code, without any stray "db" bytes in the middle, as shown below.

```
.text:00401011                    ; ---------------------------------------------------------------------------
.text:00401011
.text:00401011                    loc_401011:                             ; CODE XREF: .text:0040100E↑j
.text:00401011 8B 45 0C                   mov     eax, [ebp+0Ch]
.text:00401014 8B 48 04                   mov     ecx, [eax+4]
.text:00401017 0F BE 11                   movsx   edx, byte ptr [ecx]
.text:0040101A 83 FA 70                   cmp     edx, 70h
.text:0040101D 75 3F                      jnz     short loc_40105E
.text:0040101F 33 C0                      xor     eax, eax
.text:00401021 74 01                      jz      short near ptr loc_401023+1
.text:00401023
.text:00401023                    loc_401023:                             ; CODE XREF: .text:00401021↑j
.text:00401023 E8 8B 45 0C 8B             call    near ptr 8B4C55B3h
.text:00401028 48                         dec     eax
.text:00401029 04 0F                      add     al, 0Fh
.text:0040102B BE 51 02 83 FA             mov     esi, 0FA830251h
.text:00401030 71 75                      jno     short near ptr loc_4010A4+3
.text:00401032 2B 33                      sub     esi, [ebx]
```

Fixing the Code at 401024

Use the same technique to fix the code at 401024, with these steps:

1. Tell IDA to interpret the bytes starting at 401023 as data
2. Tell IDA to interpret the bytes starting at 401024 as code
3. Tell IDA to interpret the bytes starting at 401027 as code
4. Tell IDA to interpret the bytes starting at 40102A as code
5. Tell IDA to interpret the bytes starting at 40102E as code
6. Tell IDA to interpret the bytes starting at 401031 as code
7. Tell IDA to interpret the bytes starting at 401033 as code
8. Tell IDA to interpret the bytes starting at 401037 as code

This reveals the real assembly code through a **call near ptr** instruction at 401037, as shown below.

```
.text:00401024                    loc_401024:                             ; CODE XREF: .text:00401021↑j
.text:00401024 8B 45 0C                   mov     eax, [ebp+0Ch]
.text:00401027 8B 48 04                   mov     ecx, [eax+4]
.text:0040102A 0F BE 51 02                movsx   edx, byte ptr [ecx+2]
.text:0040102E 83 FA 71                   cmp     edx, 71h
.text:00401031 75 2B                      jnz     short loc_40105E
.text:00401033 33 C0                      xor     eax, eax
.text:00401035 74 01                      jz      short near ptr loc_401037+1
.text:00401037 E8 8B 45 0C 8B             call    near ptr 8B4C55C7h
.text:00401037                                                            ; CODE XREF: .text:00401035↑j
.text:0040103C 48                         dec     eax
.text:0040103D 04 0F                      add     al, 0Fh
.text:0040103F BE 51 01 83 FA             mov     esi, 0FA830151h
.text:00401044                            db      64h
.text:00401044 64 75 17                   jnz     short loc_40105E
.text:00401047 33 C0                      xor     eax, eax
.text:00401049 74 01                      jz      short near ptr loc_40104B+1
```

Fixing the Code at 401038

The same anti-disassembly technique was used, and you need to implement the same solution.

The correct disassembly shows a block of code starting at 401038 and ending with yet another use of the same trick to skip over a byte, as shown below.

```
.text:00401038
.text:00401038                  loc_401038:                             ; CODE XREF: .text:00401035↑j
.text:00401038 8B 45 0C                 mov     eax, [ebp+0Ch]
.text:0040103B 8B 48 04                 mov     ecx, [eax+4]
.text:0040103E 0F BE 51 01              movsx   edx, byte ptr [ecx+1]
.text:00401042 83 FA 64                 cmp     edx, 64h
.text:00401045 75 17                    jnz     short loc_40105E
.text:00401047 33 C0                    xor     eax, eax
.text:00401049 74 01                    jz      short near ptr loc_40104B+1
.text:0040104B
.text:0040104B                  loc_40104B:                             ; CODE XREF: .text:00401049↑j
.text:0040104B E8 68 10 30 40           call    near ptr 407020B8h
.text:00401050 00 FF                    add     bh, bh
.text:00401052 15 00 20 40 00           adc     eax, offset printf
.text:00401057 83 C4 04                 add     esp, 4
.text:0040105A 33 C0                    xor     eax, eax
.text:0040105C EB 15                    jmp     short loc_401073
.text:0040105E                  ; ---------------------------------------------------------------------------
```

Fixing the Code at 40104C

The same anti-disassembly technique was used, and you need to implement the same solution.

The correct disassembly shows a block of code starting at 401038 and ending with yet another use of the same trick to skip over a byte, as shown below.

```
.text:0040104B E8                       db 0E8h
.text:0040104C                  ; ---------------------------------------------------------------------------
.text:0040104C
.text:0040104C                  loc_40104C:                             ; CODE XREF: .text:00401049↑j
.text:0040104C 68 10 30 40 00           push    offset aGoodJob ; "Good Job!"
.text:00401051 FF 15 00 20 40 00        call    ds:printf
.text:00401057 83 C4 04                 add     esp, 4
.text:0040105A 33 C0                    xor     eax, eax
.text:0040105C EB 15                    jmp     short loc_401073
.text:0040105E                  ; ---------------------------------------------------------------------------
.text:0040105E
.text:0040105E                  loc_40105E:                             ; CODE XREF: .text:0040100A↑j
.text:0040105E                                                          ; .text:0040101D↑j ...
.text:0040105E 33 C0                    xor     eax, eax
.text:00401060 74 01                    jz      short near ptr loc_401062+1
.text:00401062
.text:00401062                  loc_401062:                             ; CODE XREF: .text:00401060↑j
.text:00401062 E8 68 1C 30 40           call    near ptr 40702CCFh
.text:00401067 00 FF                    add     bh, bh
.text:00401069 15 00 20 40 00           adc     eax, offset printf
```

Fixing the Code at 401063

The same anti-disassembly technique is used, starting at 40105E--this code will skip the byte at 401062.

Guide IDA to disassemble the code correctly.

The correct disassembly prints a message and ends with four **pop** instructions and a **retn**, as shown below.

```
.text:00401063                  ; ---------------------------------------------------------------------------
.text:00401063
.text:00401063                  loc_401063:                             ; CODE XREF: .text:00401060↑j
.text:00401063 68 1C 30 40 00           push    offset aSonIAmDisappoi ; "Son, I am disappoint."
.text:00401068 FF 15 00 20 40 00        call    ds:printf
.text:0040106E 83 C4 04                 add     esp, 4
.text:00401071 33 C0                    xor     eax, eax
.text:00401073
.text:00401073                  loc_401073:                             ; CODE XREF: .text:0040105C↑j
.text:00401073 5F                       pop     edi
.text:00401074 5E                       pop     esi
.text:00401075 5B                       pop     ebx
.text:00401076 5D                       pop     ebp
.text:00401077 C3                       retn
```

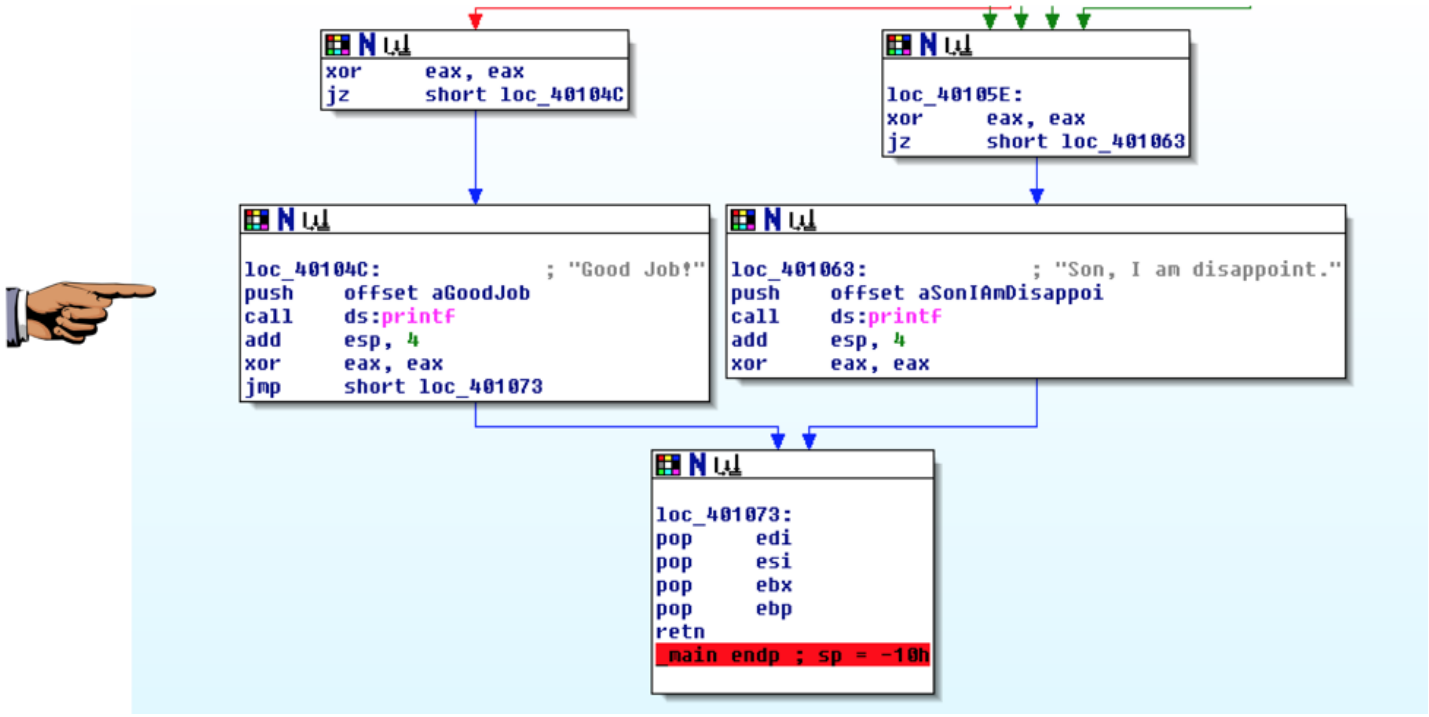
Entering Graph Mode

Press the **SPACEBAR**. The same box pops up, saying IDA can't identify the functions. Click **OK**.

Drag the mouse to highlight all the code from 401000 through the **retn** instruction at 401077. Then press **p** to tell IDA this is a function.

IDA finally switches to Graph Mode!

Scroll down and find the functions that prints out the message "**Good Job!**", as shown below.



Saving a Full-Desktop Image

Save a full-desktop image showing the message "**Good Job!**", with the filename "**Proj 12x from YOUR NAME**".

Turning in your Project

Email the image to cnit.126sam@gmail.com with the subject line: **Proj 12x from YOUR NAME**

Last modified 5-2-16