

## INDEX

<b>Practical no.</b>	<b>Practical name</b>	<b>Page no</b>
<b>1</b>	<b>Using File and Strings</b>	<b>1</b>
<b>2</b>	<b>Basic Static Techniques</b>	<b>4</b>
<b>3</b>	<b>Reverse Engineering with IDA Pro Freeware</b>	<b>8</b>
<b>4</b>	<b>Harvesting Files from Packet Captures with Wireshark</b>	<b>17</b>
<b>5</b>	<b>Basic Dynamic Techniques</b>	<b>24</b>
<b>6</b>	<b>Introduction to Hopper</b>	<b>34</b>
<b>7</b>	<b>Using Jasmin to run x86 Assembly Code</b>	<b>51</b>
<b>8</b>	<b>Assembly Code Challenges</b>	<b>59</b>
<b>9</b>	<b>IDA Pro (Lab 5-1)</b>	<b>60</b>
<b>10</b>	<b>Disassembling C on Windows Part 3</b>	<b>64</b>

# Practical 1

## Using File and Strings

### What you need:

- A Windows Computer (real or virtual) with an Internet connection.
- The textbook: "Practical Malware Analysis" .

### Purpose:

You will use the 'file' and 'strings' commands to analyze files without filename extensions. These are native Linux commands, but they have been ported to Windows.

### Downloading 'file':

In a Web browser, go here:

<http://sourceforge.net/projects/gnuwin32/files/file/5.03/file-5.03-setup.exe/download>

Download the file-5.03-setup.exe file and execute it. install the software with the default options. Open an Administrator Command Prompt window, and execute this command:

```
set PATH=%PATH%;c:\program files\gnuwin32\bin
```

### Downloading 'strings'

In a Web browser, go here:

<http://technet.microsoft.com/en-us/sysinternals/bb897439>

Click the "Download Strings" link.

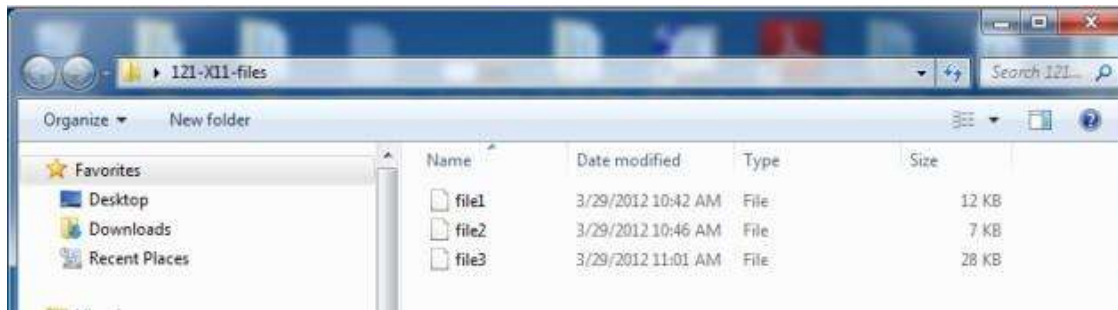
Save the Strings.zip file on your desktop. Unzip it, and copy strings.exe to the C:\Windows\System32 folder.

### Downloading the Files to Examine

In a Web browser, go here:

<http://samsclass.info/121/proj/121-X11-files.zip>

Save the **121-X11-files.zip** file on your desktop. Unzip it. A folder named 121-X11-files opens, containing three files, as shown below:



In a Command Prompt window, execute these commands, replacing 'Student' with your correct user name:

```
cd \Users\Student\Desktop\121-X11-files file *
```

The file types of the files is revealed, as shown below:

```
C:\Users\student\Desktop\121-X11-files>file *
file1: PNG image, 322 x 68, 8-bit/color RGB, non-interlaced
file2: Non-ISO extended-ASCII English text, with very long lines, with CRLF line
terminators
file3: PE32 executable for MS Windows (console) Intel 80386 32-bit
```

## Using Strings

In a Command Prompt window, execute this command:

```
strings file1 | more
```

You see the readable strings in the file, which are just nonsense, because this is an image file, as shown below:



```
Administrator: cmd - Shortcut

C:\Users\student\Desktop\121-X11-files>strings file1 | more

Strings v2.42
Copyright (C) 1999-2011 Mark Russinovich
Sysinternals - www.sysinternals.com

PNG
IHDR
iCCPICC Profile
TUA
Kwv?
+w\
<zGy
Mxd4
uy1
u<
B+?
!!a;8
KHp
H260f
-zHx
PB~
POz
<B8F
COL
: N
13<x7
n0&2
:2u
@B\#
wCKnK
/><
D^G
```

## Find Secret Messages

Now you have tools powerful enough to find the two secret messages I have hidden in those files.

To find them, I recommend this procedure:

- Change all file extensions to reflect the correct filetype you learned with file
- Open the files in the appropriate program for each type
- View the strings inside each file

You will find messages saying "The secret message is..." in two of the files. One of them contains no secret.

When you find the secrets, save them, as screen captures.

# Practical 2

## Basic Static Techniques

### What you need:

- A Windows computer (real or virtual) with an Internet connection
- Recommended: the textbook: "Practical Malware Analysis"

### Purpose

You will practice the techniques in chapter 1.

This project follows Lab 1-2 in the textbook. There are more detailed solutions in the back of the book.

### VirusTotal

Turn in an image showing your analysis of Lab01-02.exe as shown below. We will grade it by checking the last digits of the SHA256 value.



Press the **PrntScrn** key to capture an image of the whole desktop.

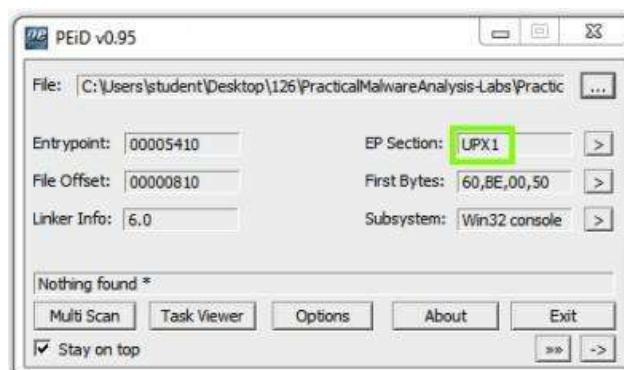
Open Paint and paste the image in with **Ctrl+V**.

Save this image with the filename "**Proj 2a from YOUR NAME**".

**YOU MUST SUBMIT WHOLE-DESKTOP IMAGES TO GET FULL CREDIT!**

### Unpacking the File

Run PEiD on the file. It shows that the file is packed with UPX, as shown in the "EP Section" below.



Download the UPX Zip file from here:

<http://upx.sourceforge.net/>

Download the **upx391w.zip** file, as shown below:

Download	
File	OS/Hardware
 upx391w.zip	Win32/i386
 upx-3.91-i386_linux.tar.bz2	Linux/i386
 upx-3.91-amd64_linux.tar.bz2	Linux/AMD64
 upx-3.91-armeb_linux.tar.bz2	Linux/ARM
 upx-3.91-mipsel_linux.tar.bz2	Linux/MIPS
 upx-3.91-powerpc_linux.tar.bz2	Linux/PPC
 upx391d.zip	DOS/i386
 upx391a.zip	Atari TOS-MiNT/m68k
 upx-3.91-src.tar.bz2	Source code (you will need UCL)

Just in case, here is the [archive of old versions](#).

Unzip it and put upx.exe in your C:\Windows\System32 folder. Open a Command Prompt window and execute this command:

UPX

You see a UPX help message, as shown below:

```
Administrator: cmd - Shortcut (2)
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\System32>upx
      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2013
UPX 3.09w      Markus Oberhumer, Laszlo Molnar & John Reiser   Feb 18th 2013
Usage: upx [-123456789dithVL] [-qvfk] [-o file] file..

Commands:
  -1      compress faster          -9      compress better
  -d      decompress              -l      list compressed file
  -t      test compressed file    -V      display version number
  -h      give more help          -L      display software license

Options:
  -q      be quiet
  -oFILE  write output to 'FILE'
  -f      force compression of suspicious files
  -k      keep backup files
file..   executables to (de)compress

Type 'upx --help' for more detailed help.
UPX comes with ABSOLUTELY NO WARRANTY; for details visit http://upx.sf.net
C:\Windows\System32>_
```

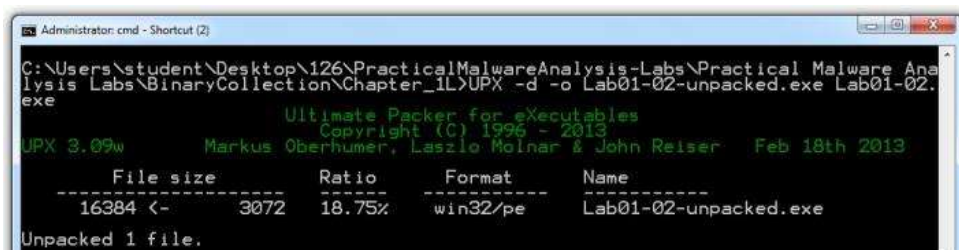
Use the CD command to move to the directory containing your malware samples.  
On my machine, I used this command:

```
cd "%Users\Administrator\Desktop\126\Practical Malware
AnalysisLabs\BinaryCollection\Chapter_1L"
```

Execute this command to unpack the file:

```
UPX -d -o Lab01-02-unpacked.exe Lab01-02.exe
```

The file unpacks, as shown below:



```
Administrator: cmd - Shortcut (2)
C:\Users\student\Desktop\126\PracticalMalwareAnalysis-Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_1L>UPX -d -o Lab01-02-unpacked.exe Lab01-02.exe
UPX 3.09w      Ultimate Packer for eXecutables
                Copyright (C) 1996 - 2013
                Markus Oberhumer, Laszlo Molnar & John Reiser   Feb 18th 2013

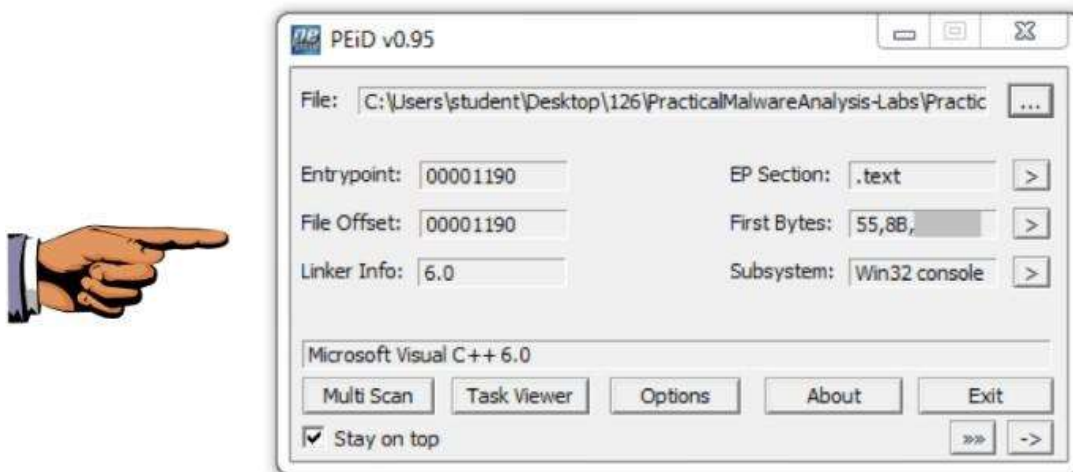
-----
File size      Ratio      Format      Name
-----
16384 <-      3072      18.75%     win32/pe     Lab01-02-unpacked.exe

Unpacked 1 file.
```

Analyze the unpacked file with PEiD. It now is recognized as a "Microsoft Visual C++ 6.0" file, as shown below.

Turn in the image showing your analysis of **Lab01-02-unpacked.exe** as shown below.

We will grade it based on the "First Bytes".

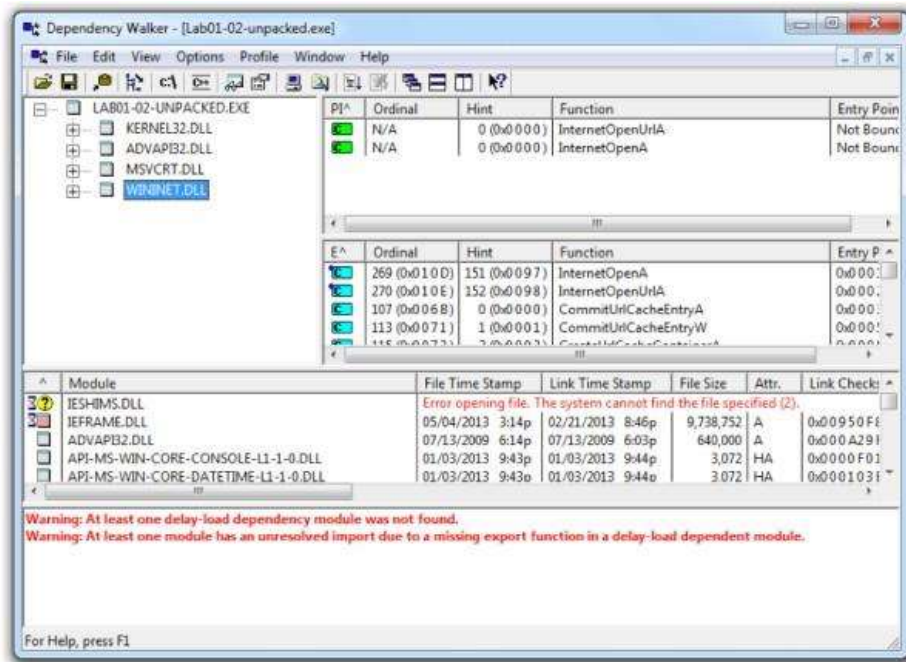


Save this image with the filename **"Proj 2b from YOUR NAME"**.

## Imports

Find the unpacked file's imports with Dependency Walker.

Turn in the image showing the two functions **InternetOpenUrlA** and **InternetOpenA** as shown in the upper right pane of the image below.



Save this image with the filename "Proj 2c from YOUR NAME".

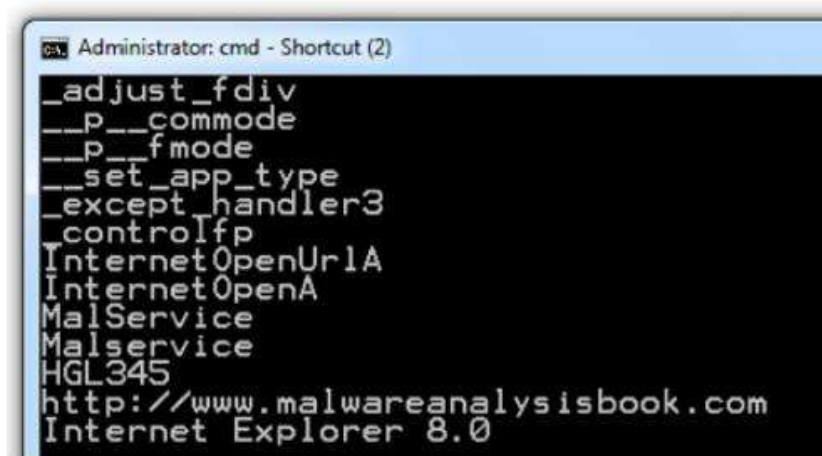
## Strings

Find the strings in the unpacked file.

You should see MalService and <http://www.malwareanalysisbook.com> as shown below.

These suggest that infected machines will connect to

<http://www.malwareanalysisbook.com> and will show a running service named **MalService**.



Save this image with the filename "Proj 2d from YOUR NAME".



# Practical 3

## Reverse Engineering with IDA Pro Freeware

### What you need:

- A Windows computer (real or virtual) with an Internet connection

### Purpose

You will use IDA Pro Free to disassemble and analyze Windows executable files.

### Downloading an EXE to Examine

Create a working directory C:\IDA.

Download this file and move it to C:\IDA

- crackme-121-1.exe

### Downloading IDA Pro Free

Open a Web browser and go to

[http://www.hex-rays.com/products/ida/support/download\\_freeware.shtml](http://www.hex-rays.com/products/ida/support/download_freeware.shtml)

At the bottom of the page, click the "IDA Freeware (16mb)" link.

Install the software with the default options. I saw an error message saying something about a single-quote directory not found, but just closed it and it seemed not to matter.

When you see the IDA window shown below, click the OK button.

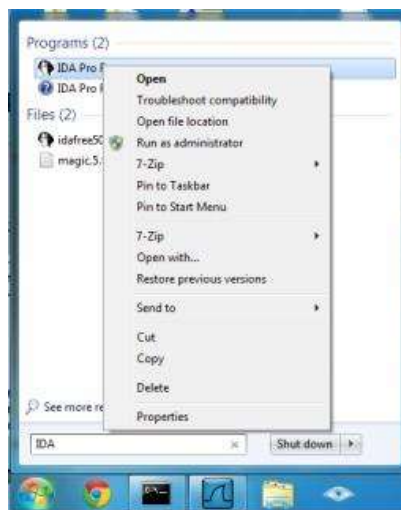


Click "I Agree".

In the "Welcome to IDA!" box, as shown below, click the New button.



If you are using Windows 7, IDA crashes. It needs Administrator privileges. Click Start, type IDA, right-click "IDA Pro Free", and click "Run as Administrator", as shown below:



If a "User Account Control" box pops up, click Yes. In the "About" box, click the OK button.

## Loading the EXE File

In the "Welcome to IDA" box, click the New button.

In the "New disassembly database" box, click "PE Executable", and then click OK, as shown below:



In the "Select PE Executable to disassemble" box, navigate to the **crackme-121-1.exe** file you saved earlier in the C:\IDA directory and double click it.

In the "Welcome to the PE Executable file loading Wizard" box, click the **Next** button, as shown below:



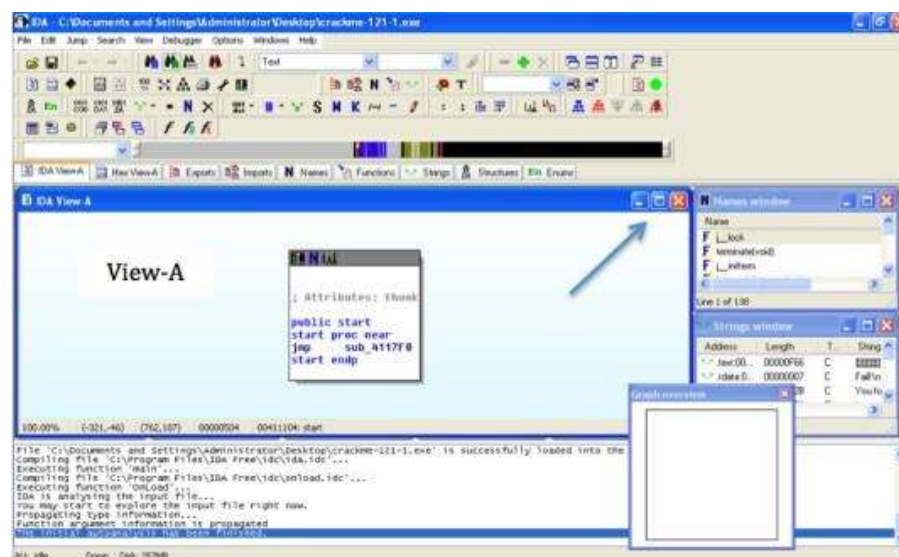
In the "Segment Creation" box, click **Yes**. In the "File loading" box, click **Finish**.

A box pops up saying "...the input file was linked with debug information...", as shown below. Click the Yes button.



## Viewing Disassembled Code

In IDA Pro, find the "View-A" pane, which shows boxes containing code linked to other boxes in a flowchart style. Maximize this pane, by clicking the button indicated by the arrow in the figure below:



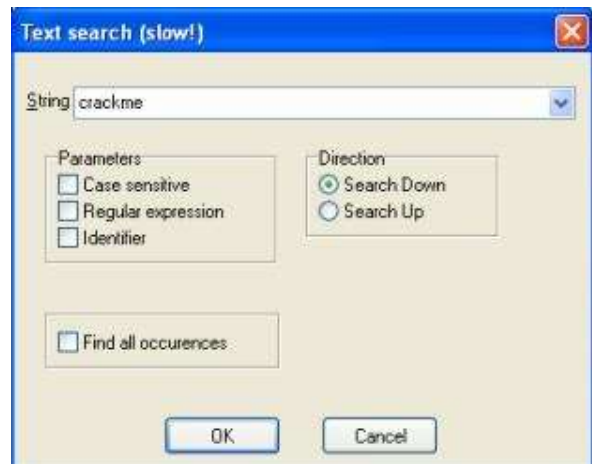
Close the "Graph Overview" box in the lower right corner.

Drag the lower border of the "View-A" pane down, to make as large a viewable area as possible.

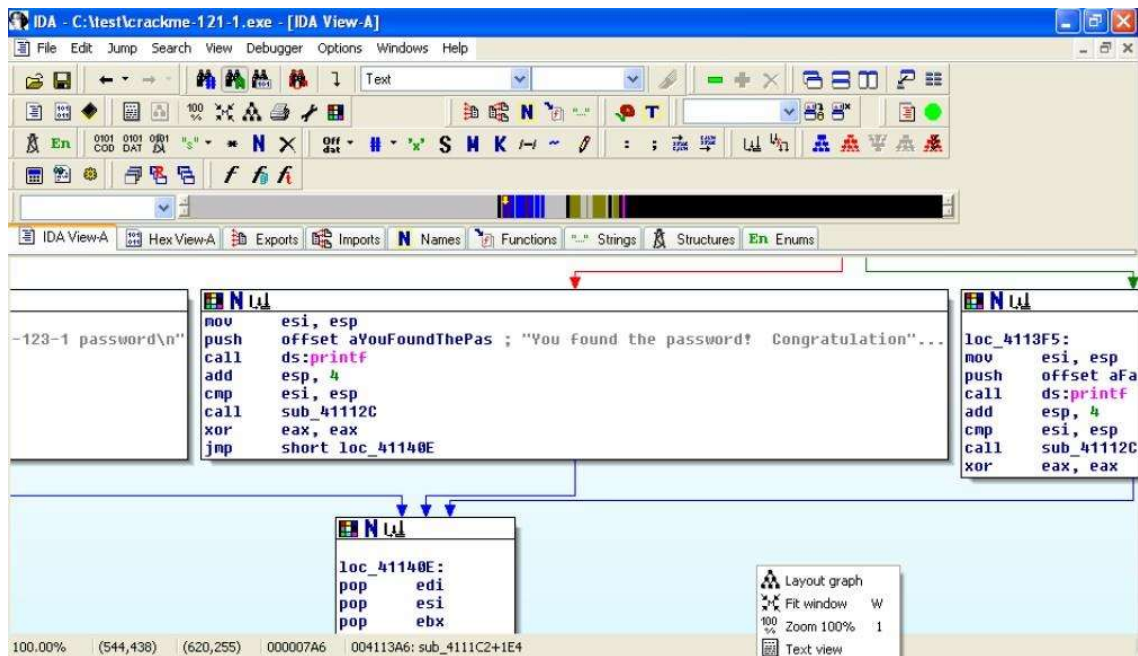
From the IDA menu bar, click **Search, Text**.

Search for crackme as shown below.

Click OK



Right-click in the "View-A" box and click "Fit window", as shown below:



You should now see the entire program shown as six boxes connected by lines, as shown below. (Ignore the two extra boxes at the upper left):



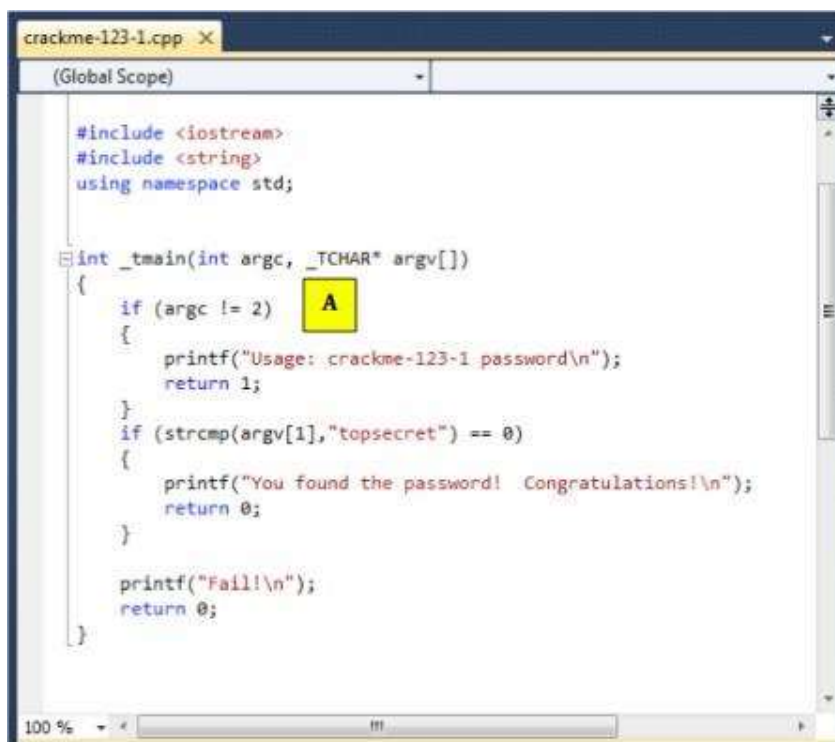
The assembly code is hard to read, but you don't need to understand it all. Focus on the last two instructions:

```
cmp [ebp+arg_0], 2
jz short loc_4113C2
```

This compares some number to 2 with the `cmp` (Compare) operation, and jumps to a different module if it is 2, using the `jz` (Jump if Zero) operation.

## C Source Code

Here is the actual C source code for the file you are disassembling. Module A is the assembly code for the first "if" statement, labelled with the yellow "A" box below:



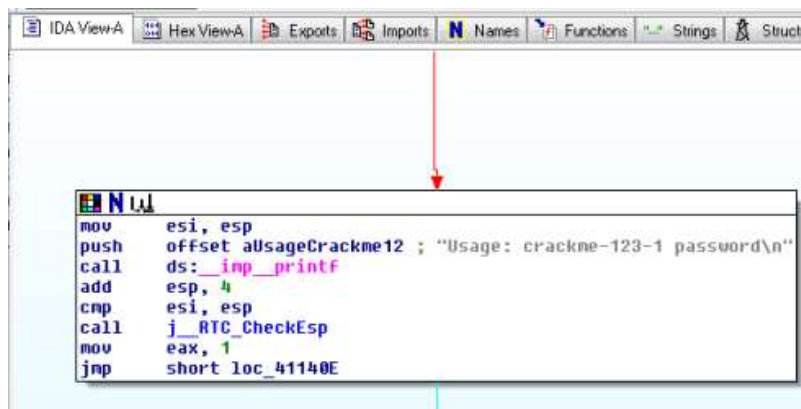
```
crackme-123-1.cpp X
(Global Scope)

#include <iostream>
#include <string>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    if (argc != 2) A
    {
        printf("Usage: crackme-123-1 password\n");
        return 1;
    }
    if (strcmp(argv[1], "topsecret") == 0)
    {
        printf("You found the password! Congratulations!\n");
        return 0;
    }

    printf("Fail!\n");
    return 0;
}
```

Drag the "View-A" display to make Module C visible, as show below:



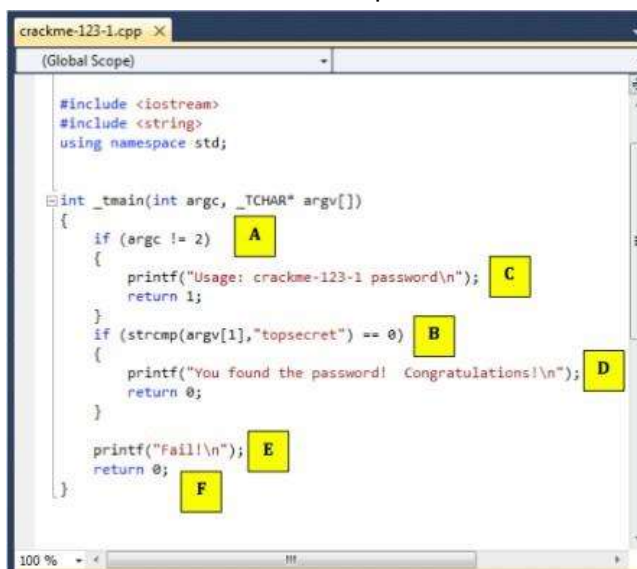
```
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structs

mov     esi, esp
push    offset aUsageCrackme12 ; "Usage: crackme-123-1 password\n"
call    ds:__imp_printf
add     esp, 4
cmp     esi, esp
call    j__RTC_CheckEsp
mov     eax, 1
jmp     short loc_41140E
```

Notice the gray readable text on the right side, saying "Usage: crackme-121-1 password". This module pushes those characters onto the stack with a **push** command, and then calls the `printf` function with the `call ds: __imp_printf` command.



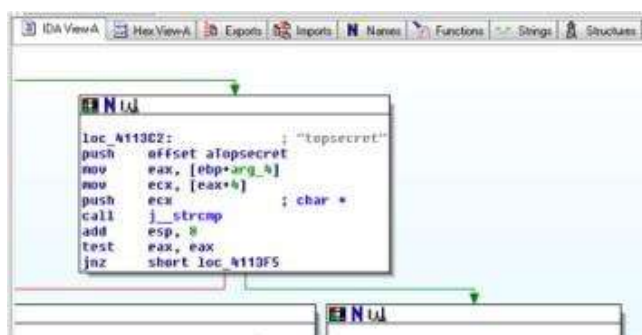
The figure below shows the C statements that compile to the "C" module:



Follow along in IDA Pro and make sure you see what each of the six modules do, and how they correspond to the C source code.

## Saving the Image

Drag the "View-A" screen to show module "B", as shown below:



Make sure the gray "topsecret" text is visible.

Save this image with the filename **Proj 2xa from YOUR NAME**

## Running the Executable

Click **Start**, type in **CMD**, and press Enter to open a Command Prompt window.

In the Command Prompt window, execute these commands:

```
cd \IDA
```

```
crackme-121-1
```

You should see the message "Usage: crackme-121-1 password", as shown below:



If you see a message saying "This application has failed to start because MSVCR100D.dll was not found", download that file here, and put it in the same folder as the .exe file:

`msvcr100d.dll`

This message is telling you that you need to add a password after the "crackme-121-1".

In the Command Prompt window, execute this command:

```
crackme-121-1 wrongpassword
```

You should see the message "Fail!".

In the Command Prompt window, execute this command:

```
crackme-121-1 topsecret
```

You should see the message "You found the password!", as shown below:



## Saving the Image

Make sure the "You found the password!" text is visible.

Save this image with the filename **Proj 2xb from YOUR NAME**

## Point Value

Those two images are worth a total of ten points. You can now earn more points by using the same technique to crack more files, as explained below.

### **crackme-121-2 (10 points)**

Download this file:

`crackme-121-2.exe`

It is very similar to crackme-121-1. Perform these steps:

1. Load the executable in IDA Pro
2. Find the module containing the password, and save a screen capture of it
3. Run the program at a command prompt and save an image of it congratulating you for finding the password.

### **crackme-121-3 (10 points)**

This one is a little more complicated, with two passwords instead of just one.

Download this file:

`crackme-121-3.exe`

Perform these steps:

1. Load the executable in IDA Pro
2. Find the modules containing the passwords, and save a screen capture of them
3. Run the program at a command prompt and save an image of it congratulating you for finding the passwords.

### **crackme-121-4 (10 points)**

This one is a little more complicated--you need to do more than just provide a password.

Download this file:

`crackme-121-4.exe`

Perform these steps:

1. Load the executable in IDA Pro



2. Find the modules that perform string comparisons (`strcmp`) and try to guess what they are referring to.
3. Run the program at a command prompt and save an image of it congratulating you for solving the puzzle.

# Practical 4

## Harvesting Files from Packet Captures with Wireshark

### What you need:

- A computer (any OS, real or virtual) with an Internet connection

### Purpose

You will use Wireshark to collect files from a packet capture.

### Stop your Antivirus

This is a real Java attack I performed with Metasploit. It's not very dangerous, because it has a hard-coded attacker IP address of 192.168.198.135 in it.

Unless you have a real attacker at that IP address, running this file won't do any harm. But it WILL set off antivirus software, which will prevent you completing the project.

So disable your antivirus, or use a virtual machine without any antivirus installed.

### Downloading the Packet Capture to Examine

Download this file and save it on your desktop:

- pX12-121.pcap (1.2 MB)

### Installing Wireshark

If you don't have Wireshark, open a Web browser and go to

<http://www.wireshark.org/> to get the appropriate version for your system. Download and install it.

### Loading the Packet Capture in Wireshark

Start Wireshark. From the Wireshark menu bar, click File, Open. Navigate to your desktop and double-click the **pX12-121.pcap** file.

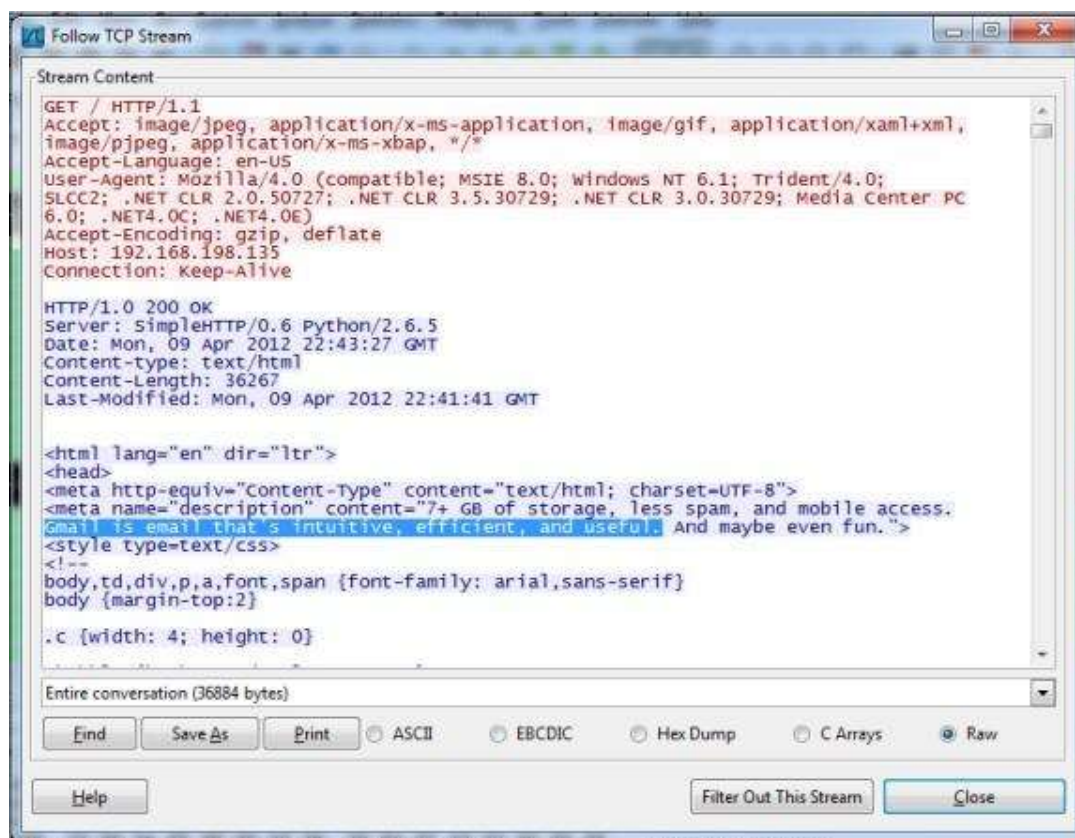
From the Wireshark menu bar, click **Statistics, Conversations**. In the "Conversations: pX12-121.pcap" window, click the **TCP:21** tab. You see the 21 conversations in the capture, as shown below:

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets B->A	Bytes B->A	Rel. Start	Duration	App. A->B	App. B->A
192.168.198.135	55037	84.19.178.7	9001	8	2 800	4	820	4	1 980	0.000623000	29.9040	219.37	529.69
192.168.198.149	1553	192.168.198.135	80	40	36 080	10	982	30	38 096	3.523080000	0.4879	16101.78	624689.22
192.168.198.149	1540	207.46.140.21	80	5	2 090	2	1 188	3	902	3.493720000	0.1304	72880.64	55325.30
192.168.198.149	1552	138.108.6.20	80	1	60	0	0	1	60	3.507887000	0.0000	N/A	N/A
192.168.198.149	1554	74.125.224.41	80	20	15 659	7	807	13	14 852	3.888428000	0.6521	9900.94	182216.64
192.168.198.149	1555	74.125.224.149	443	23	7 810	10	2 131	13	5 679	3.884425000	16.6367	1024.72	2730.81
192.168.198.149	1556	74.125.224.149	443	19	8 133	8	1 266	11	6 867	3.865980000	2.5445	3980.37	21500.19
192.168.198.149	1557	173.194.79.103	443	18	5 806	8	1 265	10	4 541	3.892376000	1.5986	6330.37	22724.29
192.168.198.149	1558	74.125.224.149	443	15	6 763	6	1 131	9	5 632	5.444805000	1.1086	8161.79	40642.98
192.168.198.149	1559	199.7.57.72	80	10	2 244	5	513	5	1 731	6.046345000	0.4293	9559.10	32254.98
192.168.198.149	1560	199.7.57.72	80	10	2 244	5	513	5	1 731	6.063612000	0.3751	10940.38	36915.79
192.168.198.149	1561	199.7.57.72	80	10	2 244	5	513	5	1 731	6.223310000	0.4039	10160.75	34285.12
192.168.198.149	1562	74.125.224.149	443	13	3 413	6	1 115	7	2 298	6.392761000	0.2645	33718.91	69494.22
192.168.198.149	1563	192.168.198.135	80	12	5 266	5	576	7	4 690	15.275198000	0.3602	12791.54	104153.37
192.168.198.149	1564	192.168.198.135	80	67	77 863	11	811	56	77 052	19.277452000	0.2038	31839.35	3025013.13
192.168.198.149	1565	74.125.224.151	443	16	4 840	7	1 094	9	3 746	19.919993000	0.5127	17071.71	58455.78
192.168.198.149	1566	173.194.64.84	443	29	19 472	10	1 682	19	17 790	20.476589000	0.8527	15780.13	166901.60
192.168.198.149	1567	192.168.198.135	443	767	998 851	84	5 938	683	992 913	20.513143000	6.0496	7852.40	1313027.02
192.168.198.149	1568	199.7.48.72	80	10	2 248	5	517	5	1 731	20.894049000	0.2804	14750.78	49388.00
192.168.198.135	50416	188.138.68.130	443	4	1 400	2	700	2	700	28.652263000	0.6990	8011.01	8011.01
192.168.198.135	52155	131.130.199.36	9001	4	1 400	2	700	2	700	28.652265000	0.1907	29367.34	29367.34

Click the second conversation, the one that exchanges 40 packets with 192.168.198.135 on port 80, as shown above. Click the Follow Stream button.

A "Follow TCP Stream" box pops up, as shown below. You can see the outgoing data in red--an HTTP GET request. The reply is in blue. This is a Gmail login page--you can see the description of the page highlighted in the image below.

You can already see that this looks suspicious--why is a Gmail login page coming from a private address like that, and not from Google?



In the "Follow TCP Stream" box, click the **Close** button.

Click the "Conversations: pX12-121.pcap" window to bring it to the front.

Click the conversation that exchanges 12 packets with 192.168.198.135 on port 80. It's near the end of the list, as shown below. Click the **Follow Stream** button.

Conversations: pX12-121.pcap

Ethernet: 13 Fibre Channel FDDI IPv4: 25 IPv6: 5 IPX JXTA NCP RSVP SCTP TCP: 21 Token Ring UDP: 21 USB WLAN

TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A-B	Bytes A-B	Packets A-B	Bytes A-B	Rel Start	Duration	bps A-B	bps A-B
192.168.198.135	55037	84.19.178.7	9001	8	2 800	4	820	4	1 980	0.000623000	29.9040	219.37	529.69
192.168.198.149	1553	192.168.198.135	80	40	39 080	10	982	30	38 098	3.323080000	0.4879	16101.76	624689.23
192.168.198.149	1540	207.46.140.21	80	5	2 090	2	1 188	3	902	3.493720000	0.1304	72880.64	55335.30
192.168.198.149	1552	138.108.6.20	80	1	60	0	0	1	60	3.507887000	0.0000	N/A	N/A
192.168.198.149	1554	74.125.224.41	80	20	15 659	7	807	13	14 852	3.808428000	0.6521	9900.94	182216.64
192.168.198.149	1555	74.125.224.149	443	23	7 810	10	2 131	13	5 679	3.864425000	16.6367	1024.72	2730.83
192.168.198.149	1556	74.125.224.149	443	19	8 133	8	1 266	11	6 867	3.865998000	2.5445	3980.37	21590.19
192.168.198.149	1557	173.194.79.103	443	18	5 806	8	1 265	10	4 541	3.892376000	1.5986	6330.37	22724.29
192.168.198.149	1558	74.125.224.149	443	15	6 763	6	1 131	9	5 632	5.444805000	1.1086	8161.79	40642.98
192.168.198.149	1559	199.7.57.72	80	10	2 244	5	513	5	1 731	6.046345000	0.4293	9559.10	32254.98
192.168.198.149	1560	199.7.57.72	80	10	2 244	5	513	5	1 731	6.063612000	0.3751	10940.38	36915.79
192.168.198.149	1561	199.7.57.72	80	10	2 244	5	513	5	1 731	6.223310000	0.4039	10160.75	34285.12
192.168.198.149	1562	74.125.224.149	443	13	3 413	6	1 115	7	2 298	6.392761000	0.2645	33718.91	69494.22
192.168.198.149	1563	192.168.198.135	80	12	5 260	5	576	7	4 680	15.275198000	0.3602	12791.54	104133.37
192.168.198.149	1564	192.168.198.135	80	67	77 863	11	811	56	77 052	19.277452000	0.2038	31839.35	3025013.13
192.168.198.149	1565	74.125.224.181	443	16	4 840	7	1 094	9	3 746	19.919993000	0.5127	17071.71	58455.78
192.168.198.149	1566	173.194.64.84	443	29	19 472	10	1 682	19	17 790	20.476589000	0.8527	15780.13	166901.60
192.168.198.149	1567	192.168.198.135	443	767	998 851	84	5 938	683	992 913	20.513143000	6.0496	7852.40	1313027.02
192.168.198.149	1568	199.7.48.72	80	10	2 248	5	517	5	1 731	20.894049000	0.2804	14750.78	49388.00
192.168.198.135	50416	188.138.88.130	443	4	1 400	2	700	2	700	28.652263000	0.6990	8011.01	8011.01
192.168.198.135	52155	131.130.199.36	9001	4	1 400	2	700	2	700	28.652265000	0.1907	29367.34	29367.34

☒ Name resolution ☐ Limit to display filter

Help Copy Follow Stream Close

The "Follow TCP Stream" box shows a request for a file named "Signed\_update.jar", as highlighted in the image below.

Follow TCP Stream

Stream Content

```

GET /Signed_update.jar HTTP/1.1
accept-encoding: pack200-gzip, gzip
content-type: application/x-java-archive
User-Agent: Mozilla/4.0 (windows 7 6.1) Java/1.6.0_30
Host: 192.168.198.135
Accept: text/html, image/gif, image/jpeg, */*; q=.2, */*; q=.2
Connection: keep-alive
Cookie: TZ=420

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.6.5
Date: Mon, 09 Apr 2012 22:43:39 GMT
Content-type: application/java-archive
Content-Length: 4087
Last-Modified: Mon, 09 Apr 2012 22:42:11 GMT

PK.....?.....META-INF/MANIFEST.MF.M..LK-..
K-".R0.3..r.JM,IM.u.....(h...).f&...W.....+x%.i.r.r.%..Z)x%.%.$...r,
{8...d...RH.O-W
..
...s.p.
I.L*I.t..i..PK...S.....PK.....?.....META-INF/
SIGNAPPL.SFm..N.@...M...{.....&.J.].Z5.d^vy.<'w.)_....\f.@.Z...2....A..^!...P
+.....@..s...).7W.DK...C."t]60.\.....F.S+>...~...h...2
[.Y.6.=P...?.L.Z?...j.2YL.....t]>...6*f.p...k.t...MG.j.....k..D...PK..}
Z.....PK.....?.....META-INF/
SIGNAPPL.DSA3hb.e..j.h.....U..I.....QA...A...A...L.LL,~7'+.....
'a.j(.g.ea.f

6.6..q8....32..K.....@.....%9..B...a.N.....7.1f$1.9q^CCC#.c.cc.#.

```

Entire conversation (4582 bytes)

Find Save As Print ASCII EBCDIC Hex Dump C Arrays Raw

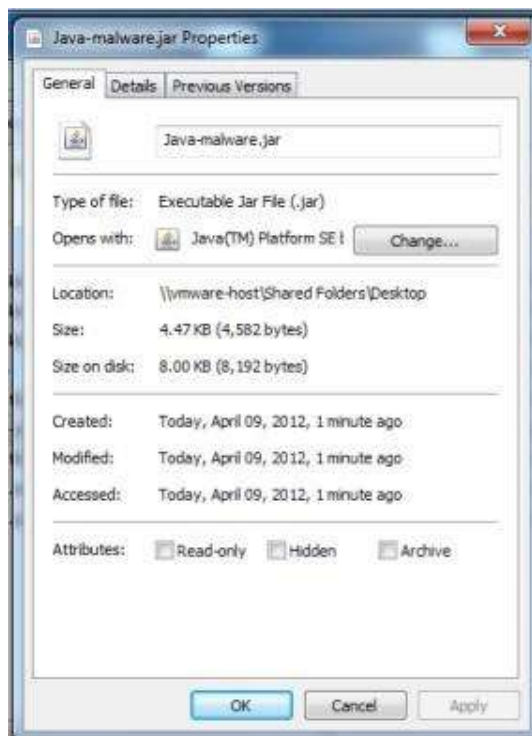
Help Filter Out This Stream Close

This is a Java archive. And the real Gmail page doesn't use Java. Also, if it is really a Java Update, it should be coming from Oracle, not from the same server that delivered the Gmail login page.

You can see the file contents in blue in the image above--it's a binary file so it's unreadable. You can, however, save the file in its binary form. In the "Follow TCP Stream" box, click the Save As button. Save the file on your desktop as "Java-malware.jar".

## Checking the Size of the Java Malware File

On your desktop, right-click the "Java-malware.jar" file and click Properties. The file size should be exactly **4,582 bytes**, as shown below:



## Saving the Image

Make sure the file size of **4,582 bytes** is visible.

Save this image with the filename **Proj 3xa from YOUR NAME**

In the "Follow TCP Stream" box, click the **Close** button.

Click the "Conversations: pX12-121.pcap" window to bring it to the front.

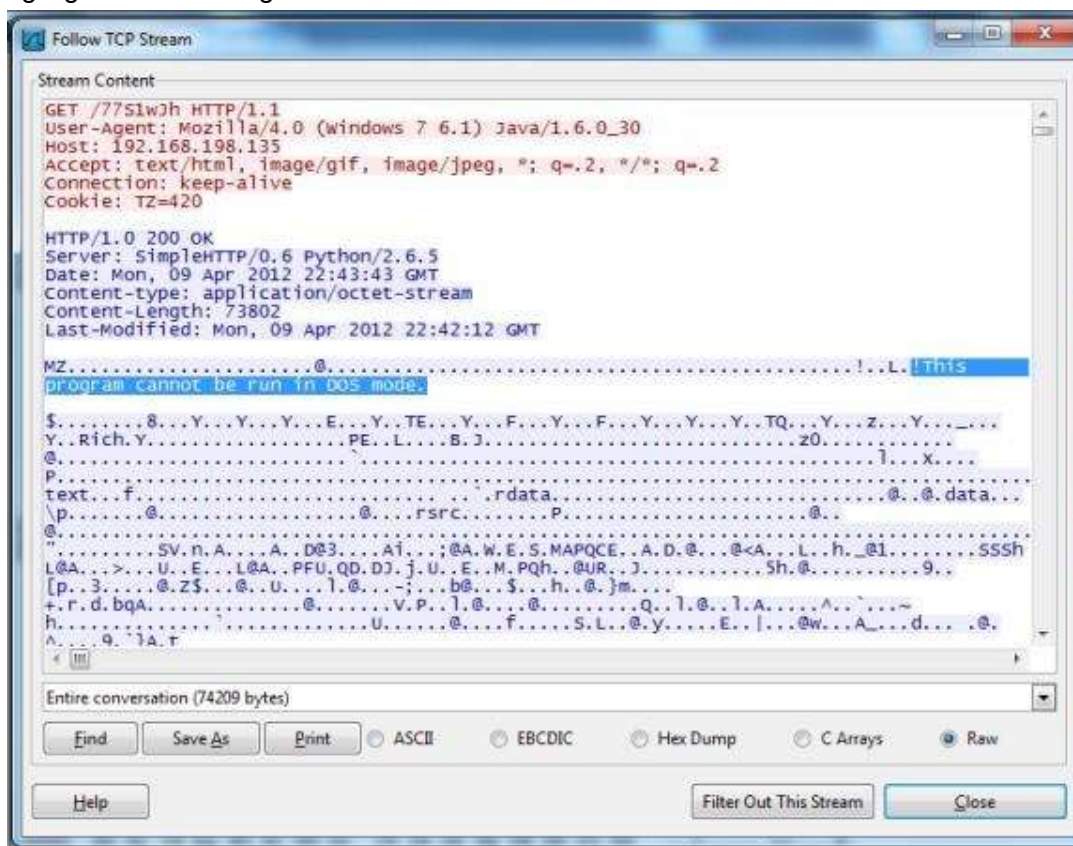
Click the conversation that exchanges 67 packets with 192.168.198.135 on port 80. It's near the end of the list, as shown below.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A-B	Bytes A-B	Packets B-A	Bytes B-A	Rel Start	Duration	bps A-B	bps B-A
192.168.198.135	5937	84.19.178.7	9001	8	2 900	4	920	4	1 980	0.000623000	29.9040	219.37	529.69
192.168.198.149	1553	192.168.198.135	80	40	39 080	10	982	30	38 098	3.323080000	0.4879	16101.76	624689.23
192.168.198.149	1540	207.46.140.21	80	5	2 090	2	1 188	3	902	3.493720000	0.1304	72880.64	55335.30
192.168.198.149	1552	138.108.6.20	80	1	60	0	0	1	60	3.507887000	0.0000	N/A	N/A
192.168.198.149	1554	74.125.224.41	80	20	15 659	7	807	13	14 852	3.808420000	0.6521	9900.94	182236.64
192.168.198.149	1555	74.125.224.149	443	23	7 820	10	2 131	13	5 679	3.864425000	16.6367	1024.72	2730.83
192.168.198.149	1556	74.125.224.149	443	19	8 133	8	1 266	11	6 867	3.865980000	2.5445	3980.37	21590.19
192.168.198.149	1557	173.194.79.103	443	18	5 806	8	1 265	10	4 541	3.892376000	1.5986	6330.37	22724.29
192.168.198.149	1558	74.125.224.149	443	15	6 763	6	1 131	9	5 632	3.444895000	1.1086	8161.79	40642.98
192.168.198.149	1559	199.7.57.72	80	10	2 244	5	513	5	1 731	6.040345000	0.4293	9550.10	31254.98
192.168.198.149	1560	199.7.57.72	80	10	2 244	5	513	5	1 731	6.063612000	0.3751	10940.38	36915.79
192.168.198.149	1561	199.7.57.72	80	10	2 244	5	513	5	1 731	6.223310000	0.4039	10160.75	34285.12
192.168.198.149	1562	74.125.224.149	443	13	3 413	6	1 115	7	2 298	6.392761000	0.2645	33718.91	69494.22
192.168.198.149	1563	192.168.198.135	80	12	5 286	5	576	7	4 690	15.275198000	0.3602	12791.54	104153.37
192.168.198.149	1564	192.168.198.135	80	12	5 286	5	576	7	4 690	15.275198000	0.3602	12791.54	104153.37
192.168.198.149	1565	74.125.224.181	443	16	4 840	7	1 094	9	3 746	19.910993000	0.5127	17071.71	58455.78
192.168.198.149	1566	173.194.64.84	443	29	19 472	10	1 682	19	17 790	20.476589000	0.8527	15780.13	166901.60
192.168.198.149	1567	192.168.198.135	443	767	998 851	84	5 938	683	992 913	20.513143000	0.0496	7852.40	1313027.02
192.168.198.149	1568	199.7.48.72	80	10	2 248	5	517	5	1 731	20.894039000	0.2804	14750.78	49388.00
192.168.198.135	50416	198.138.88.130	443	4	1 400	2	700	2	700	28.652263000	0.6990	8011.01	8011.01
192.168.198.135	52155	131.130.199.36	9001	4	1 400	2	700	2	700	28.652265000	0.1907	29367.34	29367.34



Click the **Follow Stream** button.

This conversation requests a file named "77S1wJh" -- it's not obvious what it is. But the blue response shows a clue--the text "This program cannot be run on DOS mode" is readable, as highlighted in the image below.



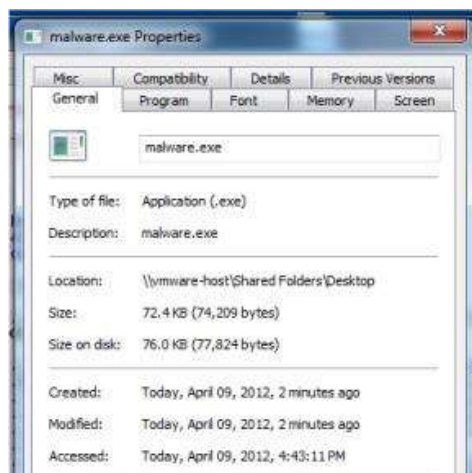
This is a Windows executable file.

In the "Follow TCP Stream" box, click the Save As button. Save the file on your desktop as **"malware.exe"**

## Checking the Size of the Executable Malware File

On your desktop, right-click the **"malware.exe"** file and click **Properties**.

The file size should be exactly **74,209 bytes**, as shown below:



## Saving the Image

Make sure the file size of **74,209 bytes** is visible.

Save this image with the filename **Proj 3xb from YOUR NAME**

In the "Follow TCP Stream" box, click the **Close** button.

Click the "Conversations: pX12-121.pcap" window to bring it to the front.

Click the conversation that exchanges 767 packets with 192.168.198.135 on port 443. It's near the end of the list, as shown below.

Conversations: pX12-121.pcap

Ethernet: 13 Fibre Channel FDDI IPv4: 25 IPv6: 5 IPX JXTA NCP RSVP SCTP TCP: 21 Token Ring UDP: 21 USB WLAN

TCP Conversations

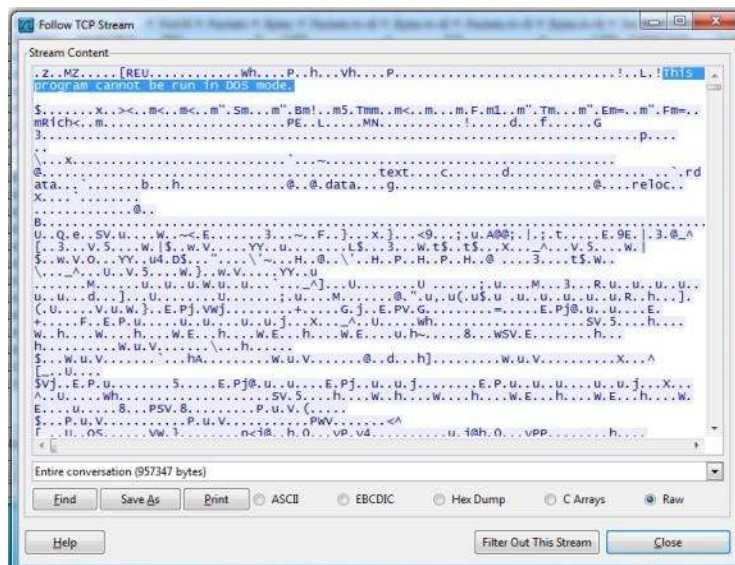
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A-B	Bytes A-B	Packets A-B	Bytes A-B	Rel Start	Duration	bps A-B	bps A-B
192.168.198.135	55037	84.19.178.7	9001	8	2 800	4	820	4	1 980	0.000623000	29.9040	219.37	529.69
192.168.198.149	1553	192.168.198.135	80	40	39 080	10	982	30	38 098	3.323080000	0.4879	16101.76	624689.23
192.168.198.149	1540	207.46.140.21	80	5	2 090	2	1 188	3	902	3.493720000	0.1304	72880.64	55335.30
192.168.198.149	1552	138.108.6.20	80	1	60	0	0	1	60	3.507887000	0.0000	N/A	N/A
192.168.198.149	1554	74.125.224.41	80	20	15 659	7	807	13	14 852	3.808428000	0.6521	9900.94	182216.64
192.168.198.149	1555	74.125.224.149	443	23	7 810	10	2 131	13	5 679	3.864425000	16.6367	1024.72	2730.83
192.168.198.149	1556	74.125.224.149	443	19	8 133	8	1 266	11	6 867	3.865998000	2.5445	3980.37	21590.19
192.168.198.149	1557	173.194.79.103	443	18	5 806	8	1 265	10	4 541	3.892376000	1.5986	6330.37	22724.29
192.168.198.149	1558	74.125.224.149	443	15	6 763	6	1 131	9	5 632	5.444805000	1.1086	8161.79	40642.98
192.168.198.149	1559	199.7.57.72	80	10	2 244	5	513	5	1 731	6.046345000	0.4293	9559.10	32254.98
192.168.198.149	1560	199.7.57.72	80	10	2 244	5	513	5	1 731	6.063612000	0.3751	10940.38	36915.79
192.168.198.149	1561	199.7.57.72	80	10	2 244	5	513	5	1 731	6.223310000	0.4039	10160.75	34285.12
192.168.198.149	1562	74.125.224.149	443	13	3 413	6	1 115	7	2 298	6.392761000	0.2645	33718.91	69494.22
192.168.198.149	1563	192.168.198.135	80	12	5 266	5	576	7	4 690	15.275198000	0.3602	12791.54	104153.37
192.168.198.149	1564	192.168.198.135	80	67	77 863	11	811	56	77 052	19.277452000	0.2038	31839.35	3025013.13
192.168.198.149	1565	74.125.224.181	443	16	4 840	7	1 094	9	3 746	19.919993000	0.5127	17071.71	58455.78
192.168.198.149	1566	173.194.64.44	443	29	19 472	10	1 682	19	17 790	20.476589000	0.8527	15780.13	166901.60
192.168.198.149	1567	192.168.198.135	443	767	998 851	84	5 938	683	992 913	20.313143000	0.0496	7852.40	1313027.02
192.168.198.149	1568	199.7.48.72	80	10	2 248	5	517	5	1 731	20.894049000	0.2804	14750.78	49388.00
192.168.198.135	50416	188.138.88.130	443	4	1 400	2	700	2	700	28.652263000	0.6990	8011.01	8011.01
192.168.198.135	52155	131.130.199.36	9001	4	1 400	2	700	2	700	28.652265000	0.1907	29367.34	29367.34

☒ Name resolution ☐ Limit to display filter

Help Copy Follow Stream Close

Click the **Follow Stream** button.

The target machine was under hostile control by this time, so it was sent a file without needing to send an HTTP request first. And, as you can see below, even though this traffic is coming in from port 443, it is not encrypted—you can read the usual message that indicates a Windows executable file, as shown below:

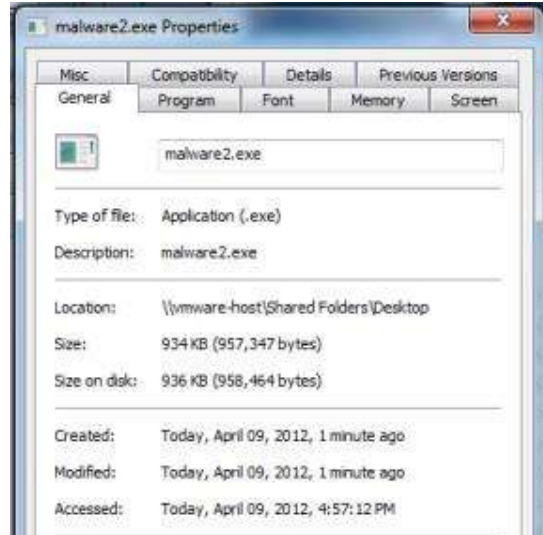


In the "Follow TCP Stream" box, click the Save As button. Save the file on your desktop as "malware2.exe"

### Checking the Size of the Executable Malware File

On your desktop, right-click the "malware2.exe" file and click **Properties**.

The file size should be exactly **957,347 bytes**, as shown below:



### Saving the Image

Make sure the file size of **957,347 bytes** is visible.

Save this image with the filename **Proj 3xc from YOUR NAME**



# Practical 5

## Basic Dynamic Techniques

### What you need:

- A Windows 2008 Server virtual machine with a Kali virtual machine running INetSim, which you preped in the previous project.

**NOTE: Windows 7 will not work for this project!**

- Recommended: the textbook: "Practical Malware Analysis"

### Purpose

You will practice the techniques in chapter 3.

This project follows Lab 3-1 in the textbook. There are more detailed solutions in the back of the book.

### Downloading Software

At the end of the previous project, you ended up with your Windows 2008 Server machine's DNS address set to your Kali machine's IP address, which means it cannot reach the Internet.

In order to download software, you need to configure a real DNS server, such as 8.8.8.8.

### Setting the DNS Server to 8.8.8.8

On your Windows VM, in Control Panel, open "Network Connections". Right-click "Local Area Connection" and click Properties.

Double-click "Internet Protocol (TCP/IP)".

Set your DNS server to 8.8.8.8

### Required Downloads

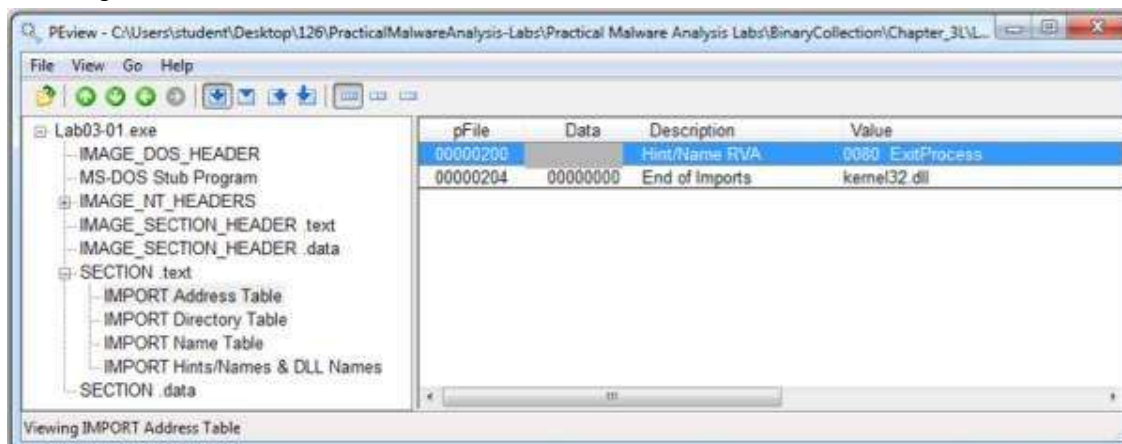
Make sure you have these items:

- **Lab Files** from <http://practicalmalwareanalysis.com/labs/> -- download and unzip them.
- **PEview** from <http://wjradsburn.com/software/> -- download and install
- **Strings** from <http://technet.microsoft.com/en-us/sysinternals/bb897439> -- Click "**Download Strings**" to get **Strings.zip**; unzip it, and copy **strings.exe** to the **C:\Windows\System32** folder.
- **Process Monitor** from <http://technet.microsoft.com/en-us/sysinternals/bb896645> -- download and unzip
- **Process Explorer** from <http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx> -- download and unzip
- **Wireshark** from <http://www.wireshark.org/> -- download and install

## Using PView

Open **Lab03-01.exe** in PView. As shown below, the only DLL imported is kernel32.dll, and the only function imported is ExitProcess. That doesn't tell us much--perhaps this malware is packed and the real imports will come at runtime.

Turn in the image showing the imports of **Lab03-01.exe** as shown below. We will grade it by checking the Data value.



Press the PrntScrn key to capture an image of the whole desktop.

Open Paint and paste the image in with Ctrl+V.

Save this image with the filename "Proj 4a from YOUR NAME".

**YOU MUST SUBMIT WHOLE-DESKTOP IMAGES TO GET FULL CREDIT!**

## Using Strings

Examine the strings in **Lab03-01.exe** and find these items, as shown below.

- SOFTWARE\Classes\http\shell\open\commandV -- A registry location
- www.practicalmalwareanalysis.com -- a URL
- VideoDriver

These readable strings are surprising--if the malware were packed, the strings would not be readable.

Above "advpack" there is a string starting with "j".

We will grade it by checking that string.

```
>>*K
48j
QQUP
ucj
j
advpack
hk?
Pj
<2f
StubPath
SOFTWARE\Classes\http\shell\open\commandV
Software\Microsoft\Active Setup\Installed Components\
test
www.practicalmalwareanalysis.com
admin
VideoDriver
WinUMX32-
umx32to64.exe
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Ph?
U5h
U>U
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
PWj
AppData
jch
UQj
UiU
UxX_
```

Save this image with the filename **"Proj 4b from YOUR NAME"**.

## Preparing for Dynamic Analysis

Dynamic analysis will help us to understand this malware better.

Here is the process detailed below:

1. Set up INetSim to simulate the Internet
2. Setting the DNS Server
3. Run Process Explorer
4. Run Wireshark
5. Run Process Monitor

### 1. Start INetSim

Start both the Windows and Linux VMs.

In Linux, start inetsim, as you did in the previous project.

Set the Windows DNS server to the Linux machine's IP address, as you did in the previous project.

Test it by opening a Web browser to this URL: **YOURNAME.com**

You should see the "INetSIM HTTP server" page, as shown below:

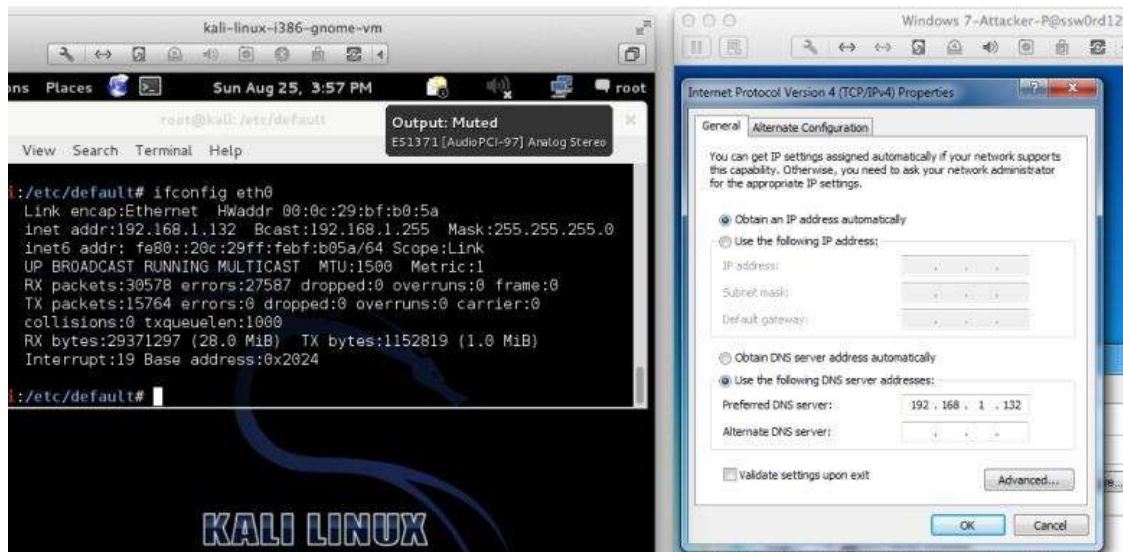


### 2. Setting the DNS Server

On your Windows VM, in Control Panel, open "Network Connections". Right-click **"Local Area Connection"** and click **Properties**.

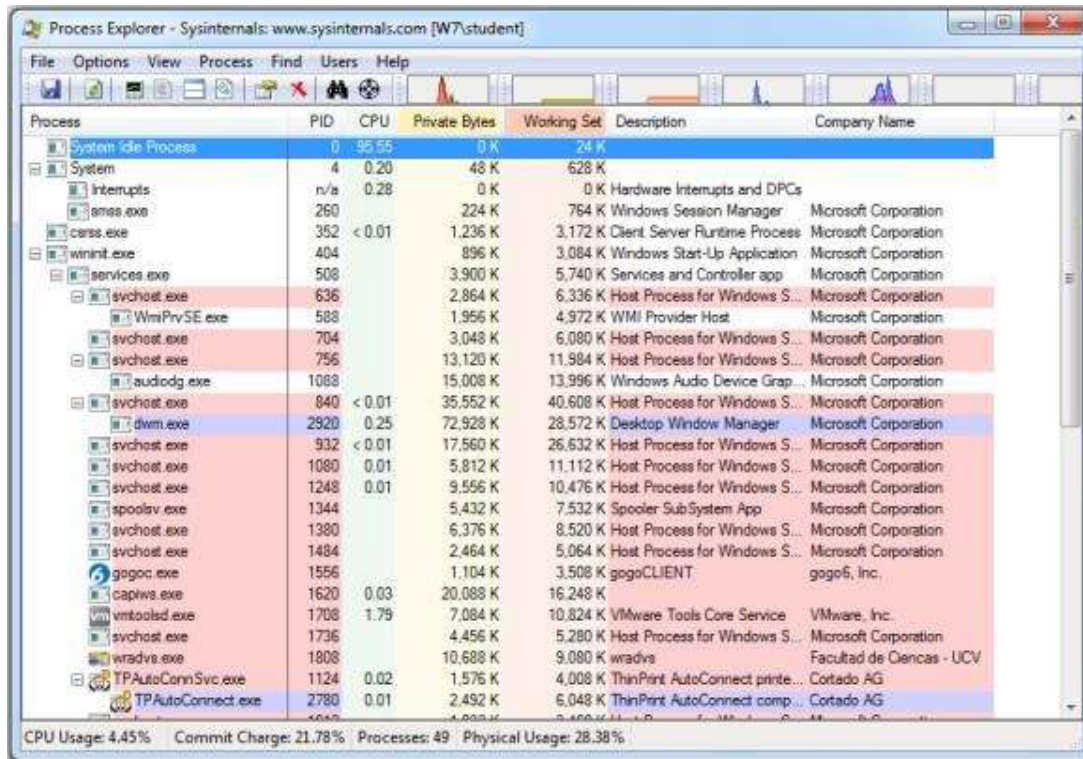
Double-click **"Internet Protocol (TCP/IP)"**.

Set your DNS server to the Kali Linux machine's IP address, as show below:



### 3. Run Process Explorer

Open Process Explorer, as shown below:



### 4. Run Wireshark

Start Wireshark and begin capturing packets from the interface that goes to the Linux machine, which is normally "Local Area Connection".

### 5. Start Process Monitor

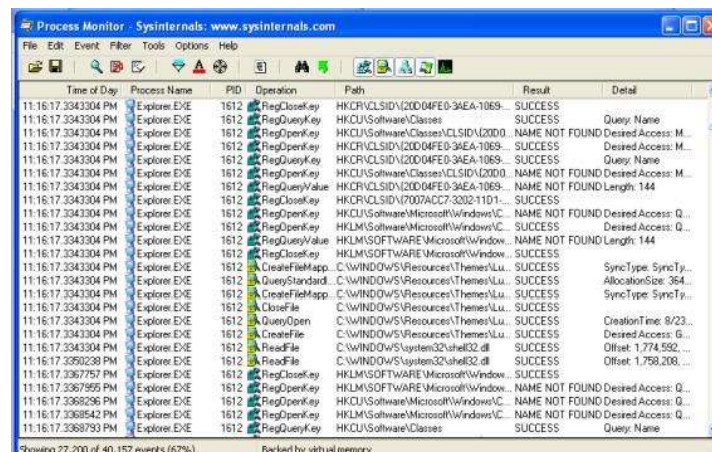
It's best to start Process Monitor last, so you can exclude all the harmless processes the other tools are using.

In the folder you unzipped Process Monitor into, double-click Procmon.exe.

If a Security Warning box pops up, allow the software to run.

Agree to the license.

You should see Process Monitor, with a lot of processes visible, as shown below:

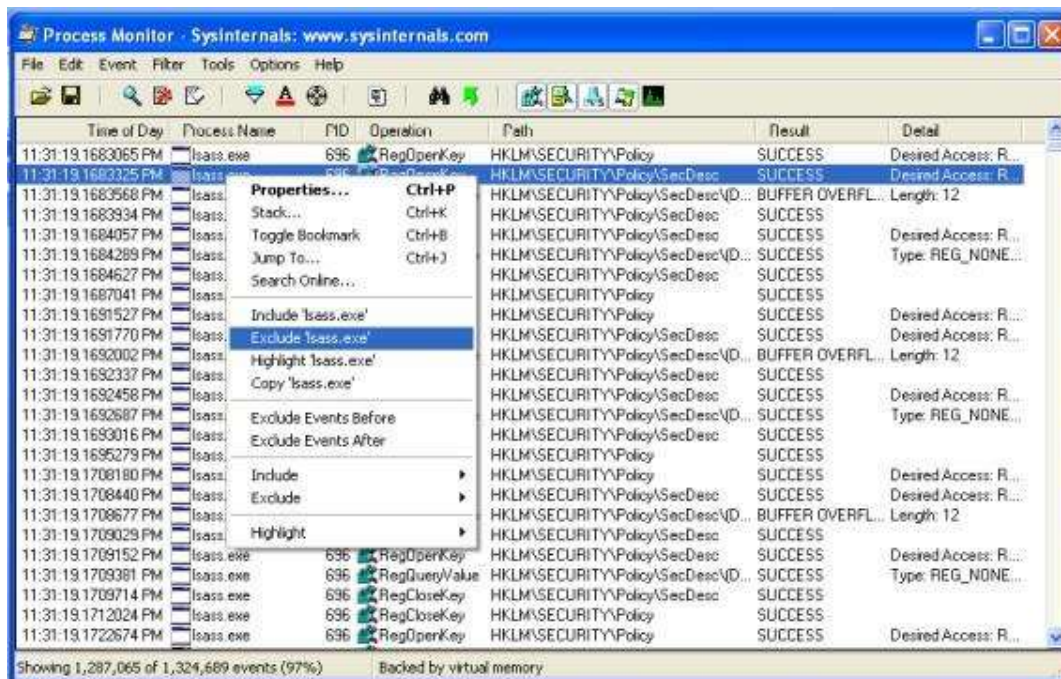




## Excluding Harmless Processes

To make the analysis easier, we will ignore all the processes that are already running before the malware starts.

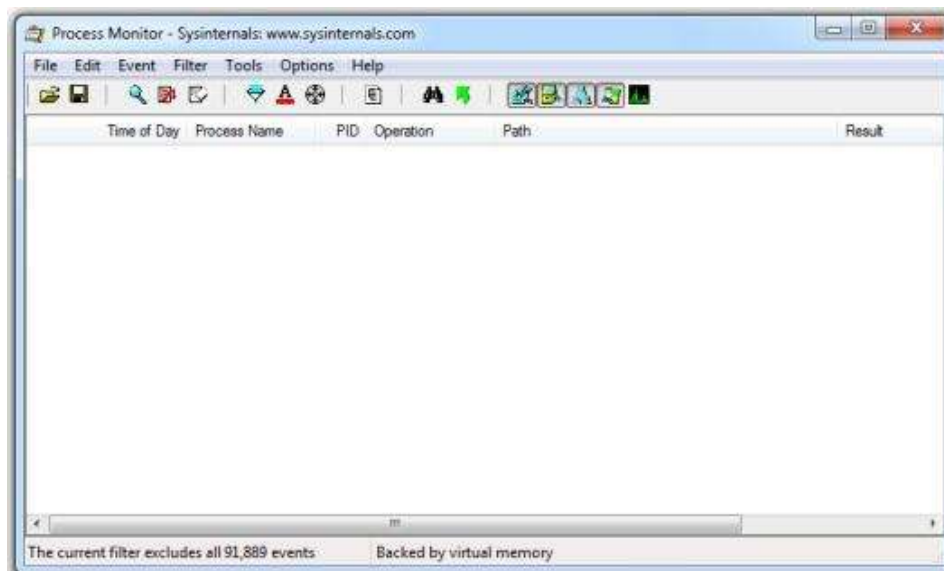
In Process Monitor, right-click the name of one of the visible processes, such as **lsass**, and click "**exclude 'lsass.exe'**", as shown below:



Wait while the event filter is applied.

Right-click a remaining process, such as **svchost.exe** and exclude it too.

Repeat the process until all current processes are hidden, as shown below. When I did it, the remaining processes to exclude were csrss.exe, explorer.exe, services.exe, vmtoolsd.exe, iexplore.exe, VMwareTray.exe, verclsid.exe, winlogon.exe, wmiprvse.exe, wuauclt.exe, regshot.exe, spoolsv.exe, alg.exe, rundll.exe, WMIADAP.EXE, GoogleUpdate.exe, GoogleCrashHandler.exe, chromeinstaller.exe, and setup.exe.



## Run the Lab03-01.exe File

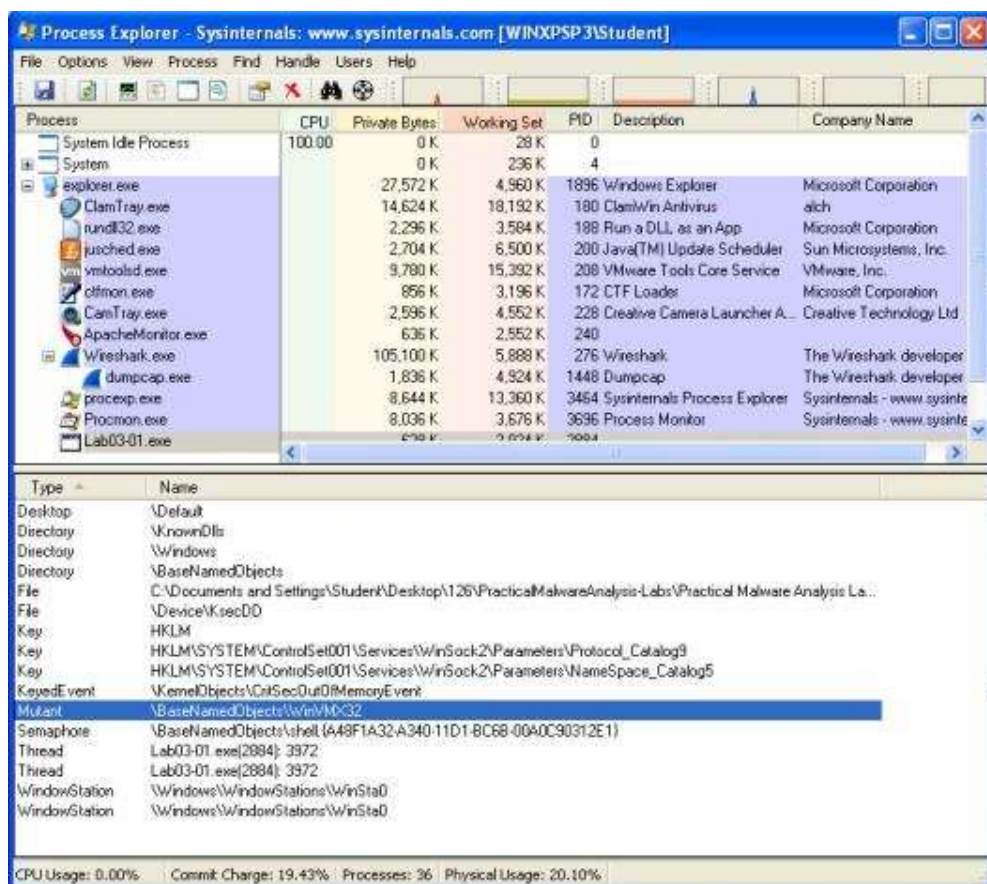
Now double-click the Lab03-01.exe File.

## Viewing the Running Malware in Process Explorer

In Process Explorer, in the top pane, find Lab03-01.exe and click it.

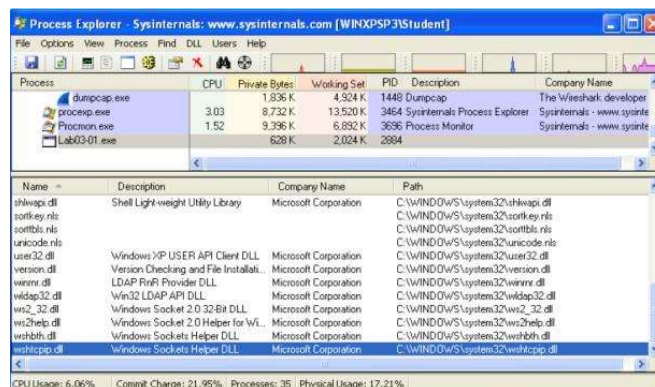
In Process Explorer, click View, "**Lower Pane View**", **Handles**.

You see the **WinVMX32** mutant, as highlighted below. A mutant, also called a mutex, is used for interprocess communication. A wonderful explanation of mutexes in terms of rubber chickens is here.



In Process Explorer, click View, "**Lower Pane View**", **DLLs**.

Scroll to the bottom to find **ws2\_32.dll** and **wshtcpip.dll**, as shown below. This shows that the malware has networking functionality.



Save this image with the filename "**Proj 4c from YOUR NAME**".

Make sure it contains the **ws2\_32.dll** and **wshtcpip.dll** items. (In Server 2008, the second item appears in capital letters like this: **WSHTCPIP.DLL**)

## Viewing the Malicious Process's Events in Process Monitor

In Process Monitor, click the magnifying glass icon on the toolbar to stop capturing events. In Process Monitor, click **Filter, Filter**. Enter a Filter for "**Process Name**" is **Lab03-01.exe**, **Include**, as shown below. Click Add to add the filter.



Add two more filters:

- **Operation of RegSetValue**
- **Operation of WriteFile**

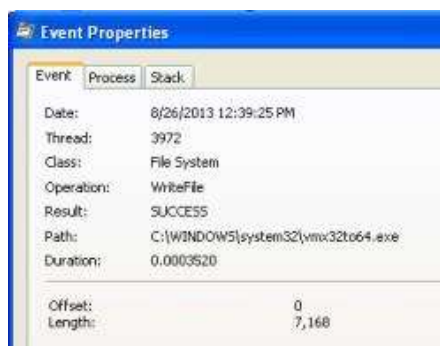
Click **OK**.

You end up the two events shown below. (Windows XP has an additional 8 events with Paths ending in "Cryptography\RNG\Seed" -- if you see those events, just ignore them.)



Double-click the event with a Path ending in vmx32to64.exe. The Properties sheet shows that this event creates a file named vmx32to64.exe, as shown below.

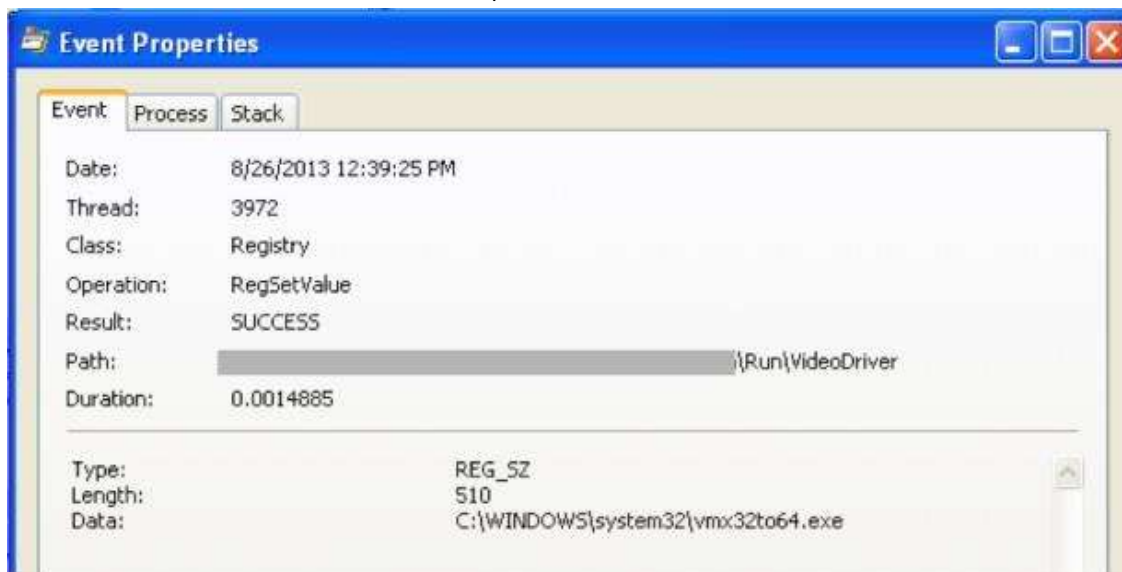
As explained in more detail in the book, this event has copied the malware itself to a file named vmx32to64.exe, so that filename is a useful indicator of infection.





Double-click the with a Path ending in **VideoDriver**.

This creates a new a Run key in the registry named "VideoDriver" with a value of "C:\WINDOWS\system32\vmx32to64.exe" -- this is a persistence mechanism, to relaunch the malware when the machine restarts.]



Save this image with the filename **"Proj 4d from YOUR NAME"**.

We will grade it based on the start of the Run registry key that is redacted above.

## Viewing INetSim Logs

On the Kali Linux machine, click in the window running inetsim.

Press Ctrl+C. A message appears telling you where the Report file is, as shown below:

```
* daytime_13_udp - stopped (PID 3404)
* daytime_13_tcp - stopped (PID 3403)
* time_37_udp - stopped (PID 3402)
* time_37_tcp - stopped (PID 3401)
* pop3s_995_tcp - stopped (PID 3392)
* syslog_514_udp - stopped (PID 3400)
* ident_113_tcp - stopped (PID 3399)
* finger_79_tcp - stopped (PID 3398)
* ntp_123_udp - stopped (PID 3397)
* ftps_990_tcp - stopped (PID 3394)
* ftp_21_tcp - stopped (PID 3393)
* pop3_110_tcp - stopped (PID 3391)
* smtps_465_tcp - stopped (PID 3390)
* smtp_25_tcp - stopped (PID 3389)
* https_443_tcp - stopped (PID 3388)
* http_80_tcp - stopped (PID 3387)
* dns_53_tcp_udp - stopped (PID 3386)
* tftp_69_udp - stopped (PID 3395)
* irc_6667_tcp - stopped (PID 3396)
Simulation stopped.
Report written to '/var/log/inetsim/report/report.3384.txt' (45 lines)
=== INetSim main process stopped (PID 3384) ===
```

In the Linux machine, execute this command, replacing "report.3384.txt" with the correct name of your report file.

```
nano /var/log/inetsim/report/report.3384.txt
```

Scroll to the bottom and you should see DNS connections to **www.practicalmalwareanalysis.com**, as shown below:



```

GNU nano 2.2.6 File: /var/log/inetsim/report/report.3384.txt
2013-08-26 15:39:05 DNS connection, type: AAAA, class: IN, requested name: tools.google.com.localdomain
2013-08-26 15:39:05 DNS connection, type: A, class: IN, requested name: tools.google.com
2013-08-26 15:39:05 HTTP connection, method: POST, URL: http://tools.google.com/service/update2?w=6:SLuBTQoZakw$
2013-08-26 15:39:05 DNS connection, type: A, class: IN, requested name: wpad.localdomain
2013-08-26 15:39:05 DNS connection, type: PTR, class: IN, requested name: 255.255.255.255.in-addr.arpa
2013-08-26 15:39:05 DNS connection, type: PTR, class: IN, requested name: 254.119.168.192.in-addr.arpa
2013-08-26 15:39:10 HTTP connection, method: POST, URL: http://tools.google.com/service/update2?w=6:Jzw2lJwGf-d$
2013-08-26 15:39:25 DNS connection, type: A, class: IN, requested name: www.practicalmalwareanalysis.com
2013-08-26 15:41:20 DNS connection, type: PTR, class: IN, requested name: 255.119.168.192.in-addr.arpa
2013-08-26 15:42:32 DNS connection, type: PTR, class: IN, requested name: 1.119.168.192.in-addr.arpa
2013-08-26 15:51:55 DNS connection, type: A, class: IN, requested name: www.practicalmalwareanalysis.com
2013-08-26 16:06:55 DNS connection, type: A, class: IN, requested name: www.practicalmalwareanalysis.com
2013-08-26 16:06:55 Last simulated date in log file

KALI LINUX
The network you simulated. The host you are on is 192.168.1.101

Get Help WriteOut Read File Prev Page Cut Text Cur Pos
Exit Justify Where Is Next Page UnCut Text To Spell

```

Save this image with the filename **"Proj 4e from YOUR NAME"**.

Make sure **"www.practicalmalwareanalysis.com"** is visible.

## Viewing the Network Request in Wireshark

In the Windows machine, in Wireshark, click **Capture, Stop**.

At the top left of the Wireshark window, in the Filter bar, type a filter of **frame contains practicalmalwareanalysis**

Press Enter to see the filtered packets, as shown below.

No.	Time	Source	Destination	Protocol	Length	Info
61	67.9901570	192.168.119.167	192.168.119.207	DNS	92	Standard query 0x175f A www.practicalmalwareanalysis.com
62	68.0074860	192.168.119.207	192.168.119.167	DNS	108	Standard query response 0x175f A 192.168.119.207
355	818.063155	192.168.119.167	192.168.119.207	DNS	92	Standard query 0xcb5b A www.practicalmalwareanalysis.com
356	818.074154	192.168.119.207	192.168.119.167	DNS	108	Standard query response 0xcb5b A 192.168.119.207
713	1718.07823	192.168.119.167	192.168.119.207	DNS	92	Standard query 0x5cab A www.practicalmalwareanalysis.com
714	1718.09298	192.168.119.207	192.168.119.167	DNS	108	Standard query response 0x5cab A 192.168.119.207

Click the line showing the first DNS request for www.practicalmalwareanalysis.com -- in the example above, it is packet 61.

In the top center of Wireshark, click the Clear button to clear the filter. The packets following the DNS request appear, as shown below.

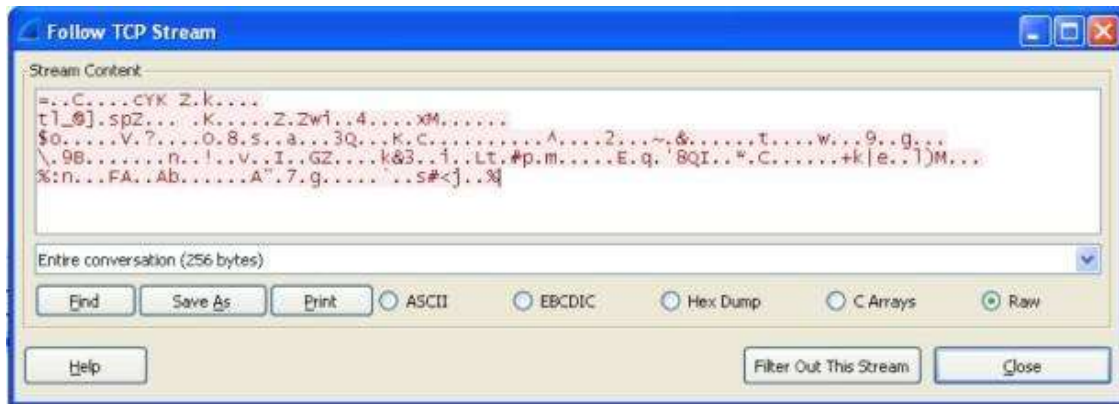
No.	Time	Source	Destination	Protocol	Length	Info
61	67.9901570	192.168.119.167	192.168.119.207	DNS	92	Standard query 0x175f A www.practicalmalwareanalysis.com
62	68.0074860	192.168.119.207	192.168.119.167	DNS	108	Standard query response 0x175f A 192.168.119.207
63	68.0121800	192.168.119.167	192.168.119.207	TCP	62	hp-webadmin > https [SYN] Seq=0 win=64240 Len=0 MSS=1460 SACK_PERM
64	68.0177040	192.168.119.207	192.168.119.167	TCP	62	https > hp-webadmin [SYN, ACK] Seq=0 Ack=1 win=14600 Len=0 MSS=1460
65	68.0177350	192.168.119.167	192.168.119.207	TCP	54	hp-webadmin > https [ACK] Seq=1 Ack=1 win=64240 Len=0
66	68.0178320	192.168.119.167	192.168.119.207	SSL	310	Continuation Data
67	68.0222380	192.168.119.207	192.168.119.167	TCP	60	https > hp-webadmin [ACK] Seq=1 Ack=257 win=15544 Len=0
68	68.0277140	192.168.119.207	192.168.119.167	TCP	60	https > hp-webadmin [RST, ACK] Seq=1 Ack=257 win=15544 Len=0

There is a TCP handshake here, but no actual HTTPS connection. A real HTTPS connection contains many more packets, such as "Client Hello", "Server Hello", and "Change Cipher Spec".

Find the SYN packet sent to the https port, which may be marked "443". In the example above, it is packet 63. Right-click it and click **"Follow TCP Stream"**.

You see "Stream Content" containing 256 bytes of random packets, as shown below. These are **beacons** and are used by malware to notify the Command and Control server that the machine is infected and ready to use.

Since the data is random, your image will look different than the example below, but it should be 256 bytes in size.



Save this image with the filename **"Proj 4f from YOUR NAME"**.  
Make sure the **"(256 bytes)"** message is visible at the bottom.

# Practical 6

## Introduction to Hopper

### Purpose

Hopper is a disassembler and debugger that runs on Mac OS X or Linux, but not Windows. It has similar functionality to IDA Pro but costs 10x less. And the free version works on 64-bit executables.

### What You Need

A 64-bit Ubuntu 14.04 machine, real or virtual. I recommend using the Ubuntu 14.04.03 machine I put on Mega, which already has Hopper installed. Here's the link to get my VM:

#### Ubuntu 14.04.03 with Hopper

Size: 2,198,435,208 bytes

SHA256( Ubuntu64-14.04.3-Hopper.7z)=

11d4cf7d54aca0b2c9d559f0ce16cdcf2be4996491a2ddaab845d272fb2458e

#### Installing Hopper

If you use the VM I put on Mega, hopper is already installed. If you prefer to install it yourself, I did it this way. Note: the official Hopper repository for Ubuntu does not work (as of 11-23-15) due to an invalid signature.

I installed Ubuntu 14.04.03 x64 Desktop into a fresh VM. Then, in a Terminal window, I executed these commands:

```
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get dist-upgrade -y
sudo apt-get install build-essential -y
sudo apt-get install git subversion autoconf automake cmake libffi-dev libxml2-dev libgnutls-dev libicu-dev libblocksruntime-dev libkqueue-dev libpthread-workqueue-dev autoconf libtool clang -y
cd /tmp
curl http://www.hopperapp.com/HopperWeb/downloads/hopperv3-3.11.2.deb > hopperv3-3.11.2.deb
sudo dpkg -i hopperv3-3.11.2.deb
```

### Starting Ubuntu

Launch the Ubuntu VM in VMware Player or VMware Fusion.

Log in with these credentials:

- Username: student
- Password: student

### Creating a Program to Debug

This is a simple vulnerable program we've used before.

In a Terminal window, execute this command:

nano pwd.c

Enter this code:

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

```
int test_pw()
```

```
{
```

```
char pin[10];
```

```
int x=15, i;
```

```

printf("Enter password: ");
gets(pin);
for (i=0; i<10; i+=2) x = (x & pin[i]) | pin[i+1];
if (x == 48) return 0;
else return 1;
}
void main()
{
if (test_pw()) printf("Fail!\n");
else printf("You win!\n");
}

```

Your screen should look like this, without the explanatory boxes and arrows:

Save the file with **Ctrl+X, Y, Enter**.

Execute these commands to compile the code and run it:

```

gcc -g -o pwd pwd.c
./pwd

```

Enter a password of **password** and press Enter. The program exits normally, with the "Fail!" message, as shown below.

```

root@kali:~/127# gcc -g -o pwd pwd.c
root@kali:~/127#
root@kali:~/127# ./pwd
Enter password: password
Fail!
root@kali:~/127#

```

## Starting Hopper

From the Ubuntu desktop, at the top left, click the square reddish Search button.

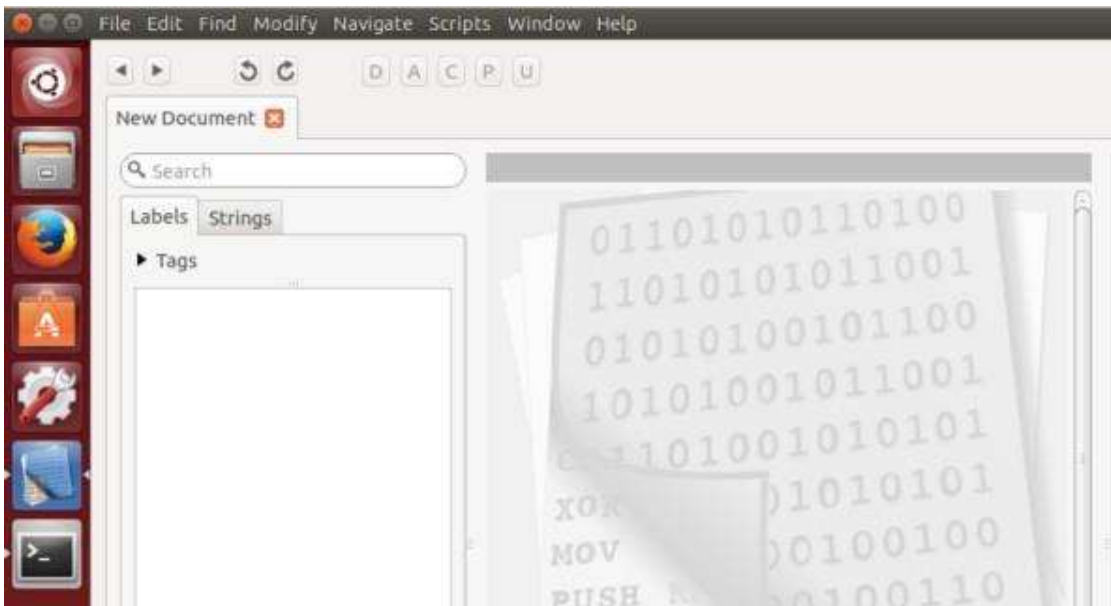
In the search field, type

**hopper**

In the search results, click "**Hopper Disassembler v3**", as shown below.

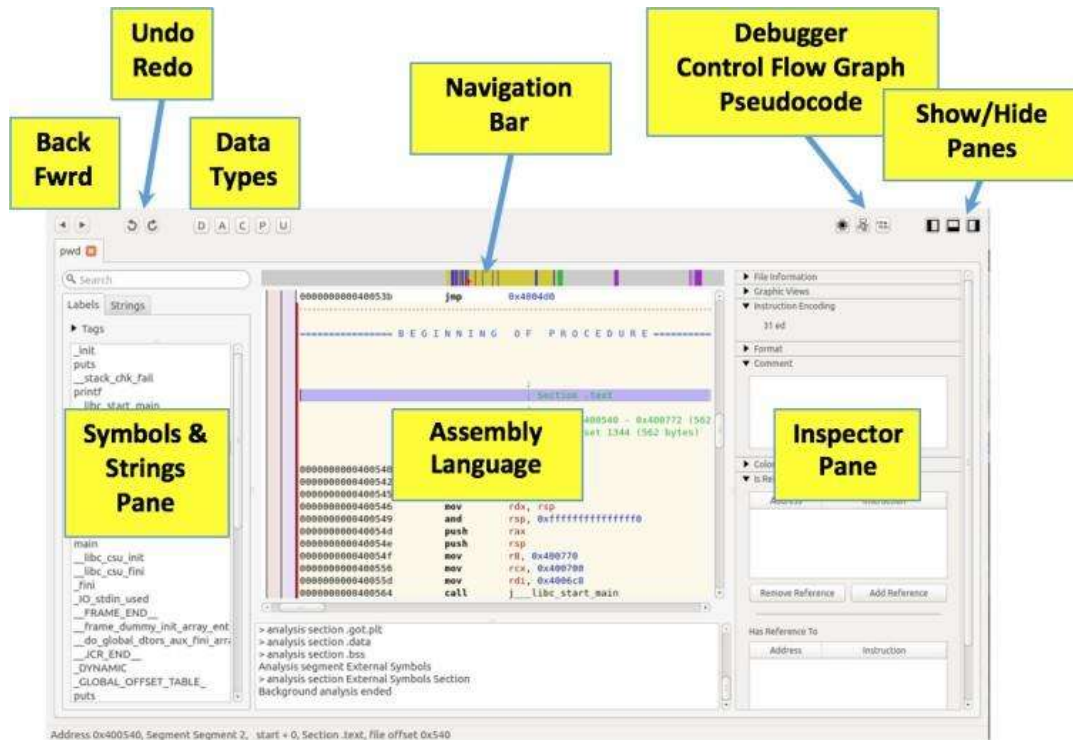


Hopper opens. Move the mouse into the dark gray bar at the top, and the menu items will appear, starting with "File", "Edit", and "Find", as shown below.




From the Hopper menu bar, click File, **"Read Executable to Disassemble..."**.  
 Navigate to the **pwd** file you created above and double-click it.  
 In the "Read Executable" box, click **OK**.  
 You see the Hopper main window, as shown below.





## Restarting Hopper

Every 30 minutes, Hopper will die, just to irritate you into paying \$90, with the message shown below.



When this happens, do these things:

- Click **OK**.
- From the Ubuntu desktop, at the top left, click the square reddish **Search** button.
- Click **"Hopper Disassembler v3"**.
- In the "Registration" box, click **"Try the Demo"**.
- From the Hopper menu bar, click **File, "Read Executable to Disassemble..."**.
- Navigate to the **pwd** file you created above and double-click it.
- In the "Read Executable" box, click **OK**.

## Pass 1: Assembly Code

We will make several passes through this program, seeing how Hopper helps us understand it.

The first pass looks at the code as a series of Assembly Language instructions. This is the most basic function of a disassembler.

## Hiding the Inspector Pane

At the top right, click the rightmost of the three "Show/Hide Panes" buttons. This gives you more room to see the Assembly Language pane.

## Hiding the "Symbols & Strings" Pane

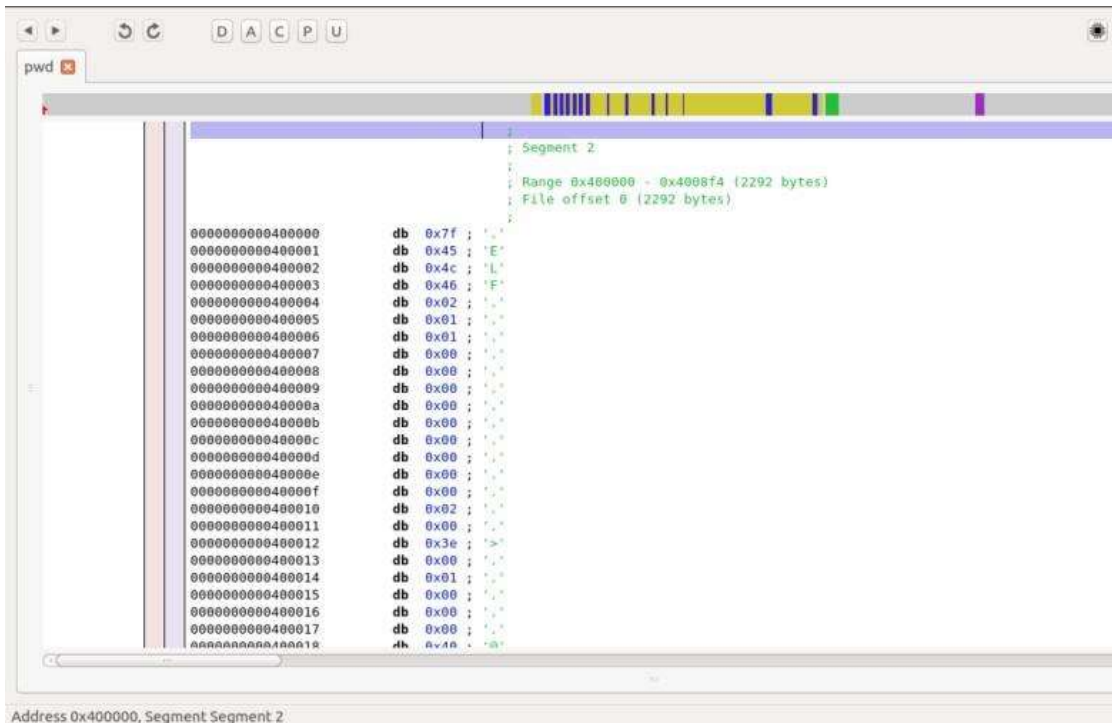
At the top right, click the leftmost of the three "Show/Hide Panes" buttons. This allows the Assembly Language pane to use the entire window width.

## Hiding the Message Pane

At the bottom, there is a pane containing messages from Hopper, like "analysis section .got.plt". Drag the top border of that section to the bottom. Now, finally, the whole window is available for assembly code, as shown below. On the right edge, drag the scroll bar to the top. Click the very first line to select it.

## The Navigation Bar

At the top center, there's a colored bar with a little red arrow. This is a linear graph of the entire file, and the red arrow indicates your current position. The red arrow in the Navigation Bar is now at the left edge, as shown below.

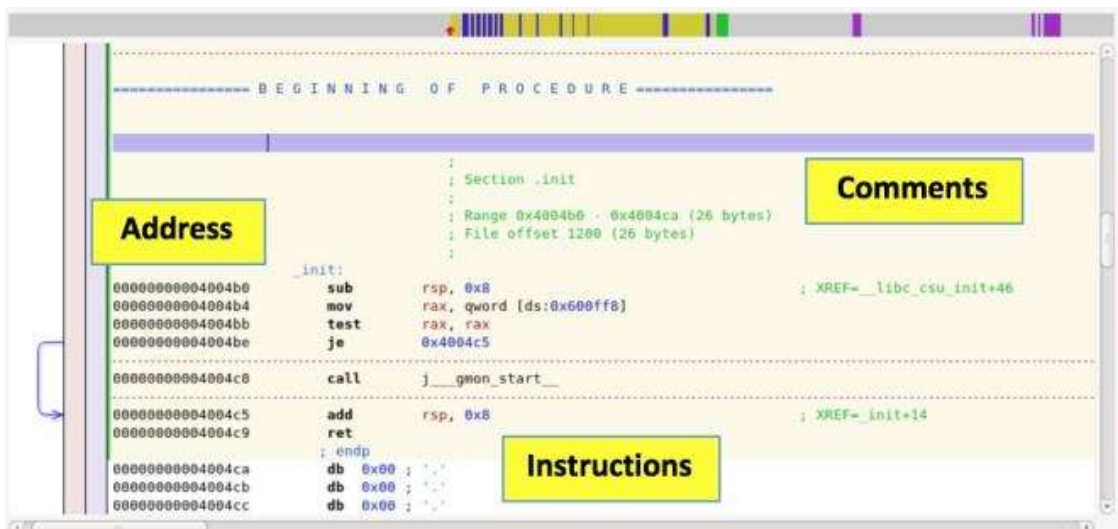


In the Navigation bar, drag the little red arrow to the first yellow stripe.

Code appears, with a yellow-shaded background behind it, as shown below.

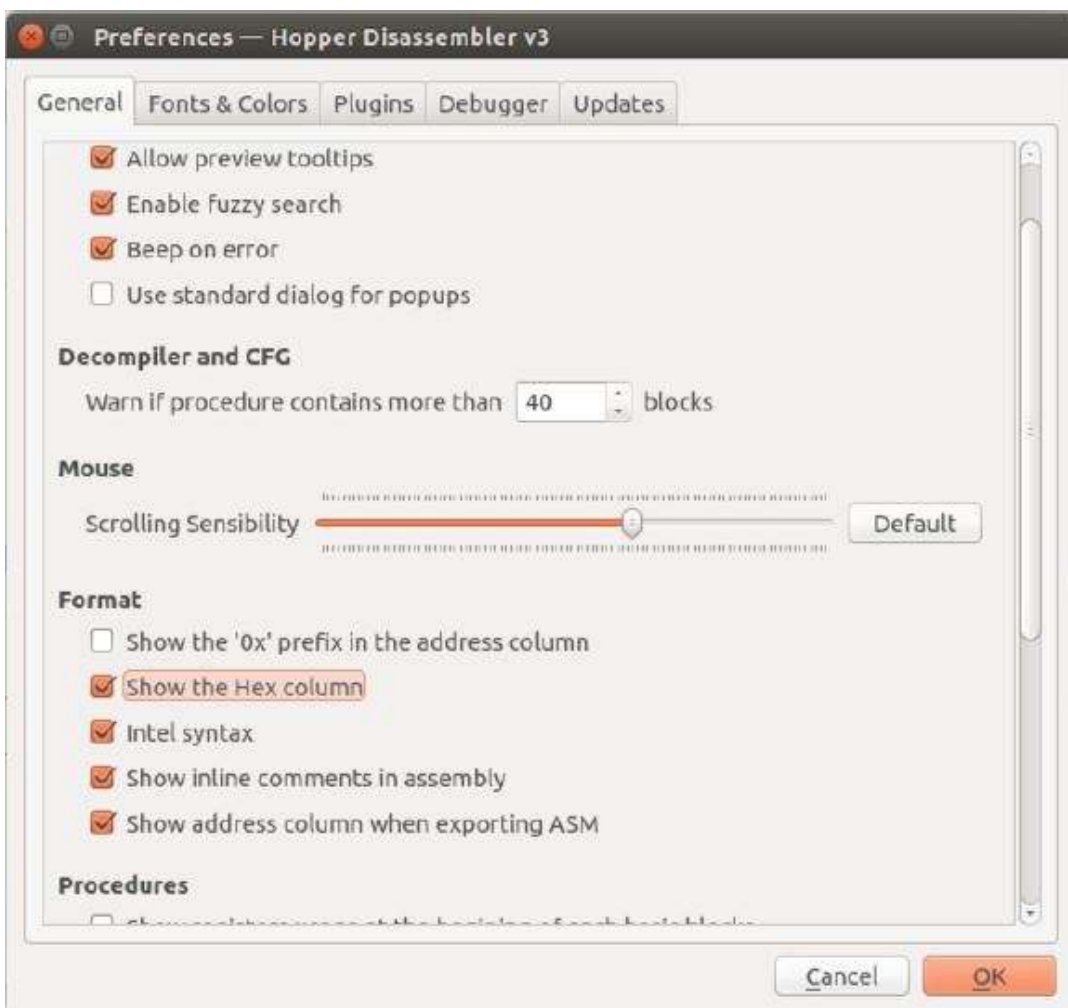
Notice these sections:

- **Address:** in hexadecimal, 64 bits long
- **Comments:** Added by Hopper to make the code easier to understand
- **Instructions:** in assembly language



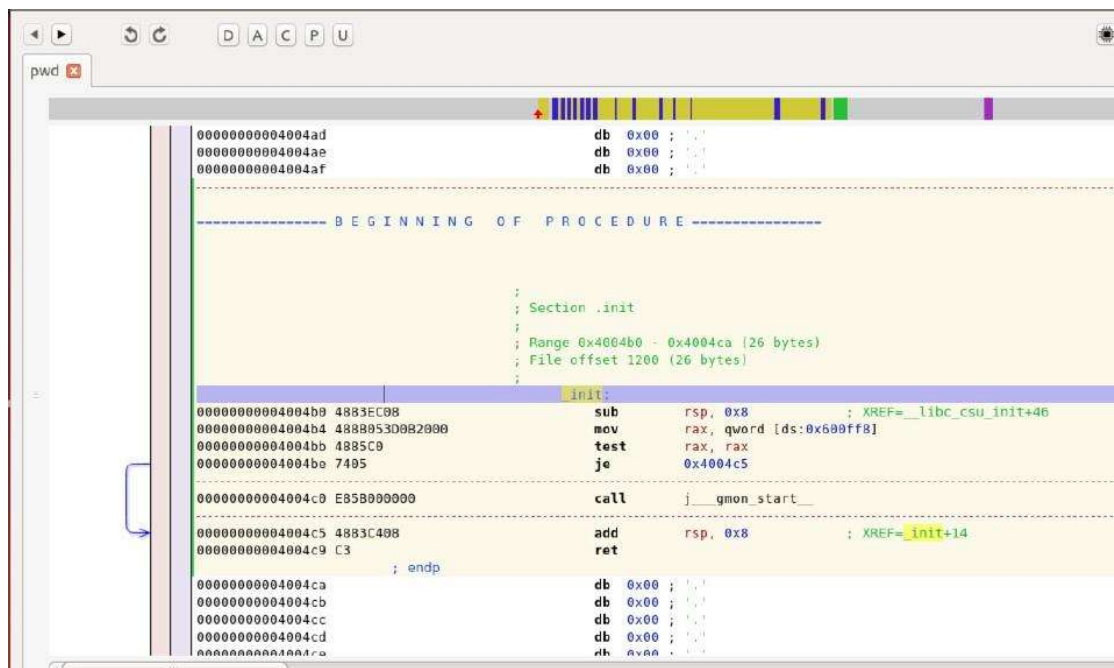
### Displaying the Hex Column

One important thing that is missing from the Hopper display is the instructions in their raw hexadecimal form. To show them, from the Hopper menu bar, click Window, Preferences. In the Preferences box, click "Show the hex column", as shown below. Click OK.





Now the hexadecimal version of each assembly instruction is visible, as shown below.



## Color Codes in Hopper

Hopper assigns each byte of a file into one of these five categories, each coded with a color:

**Data** (purple): a constant, like an array of integers

**ASCII** (green): a NULL terminated C string

**Code** (blue): an instruction

**Procedure** (yellow): Part of a method that has been successfully reconstructed by Hopper

**Undefined** (grey): an area not yet explored by Hopper

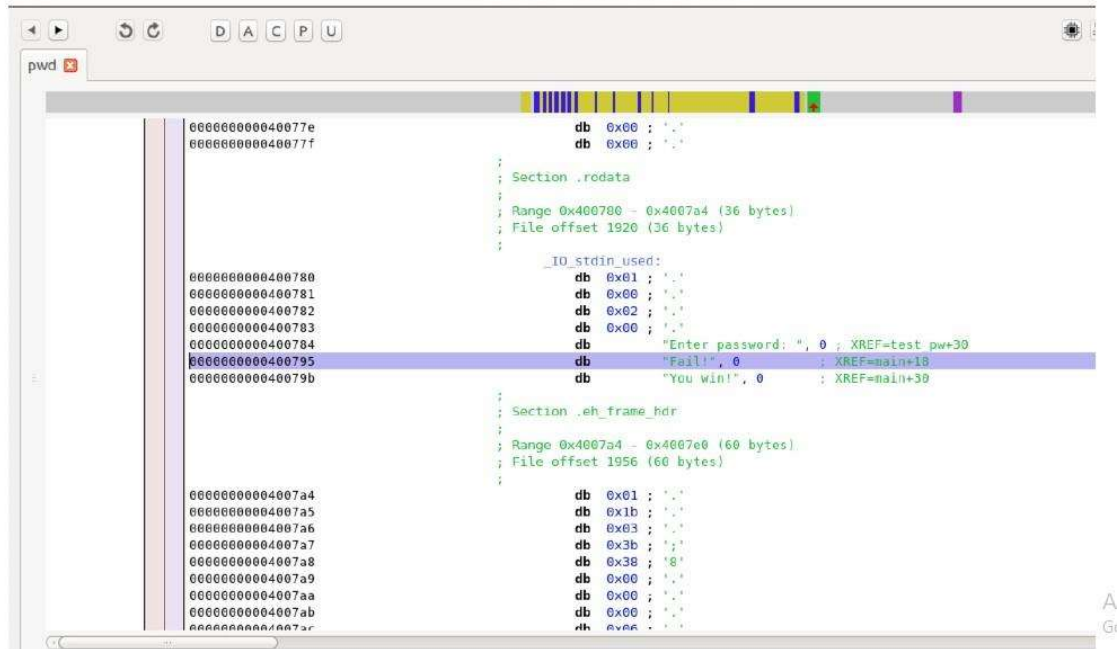
The yellow-shaded region of assembly language in the image above is a "Procedure" -- code Hopper was able to understand.

The regions with a white background above and below the yellow-shaded region are "Code" regions--they appear to be assembly instructions, but Hopper isn't sure what they are for.

## Viewing ASCII Strings

In the Navigation bar, drag the little red arrow into the green bar.

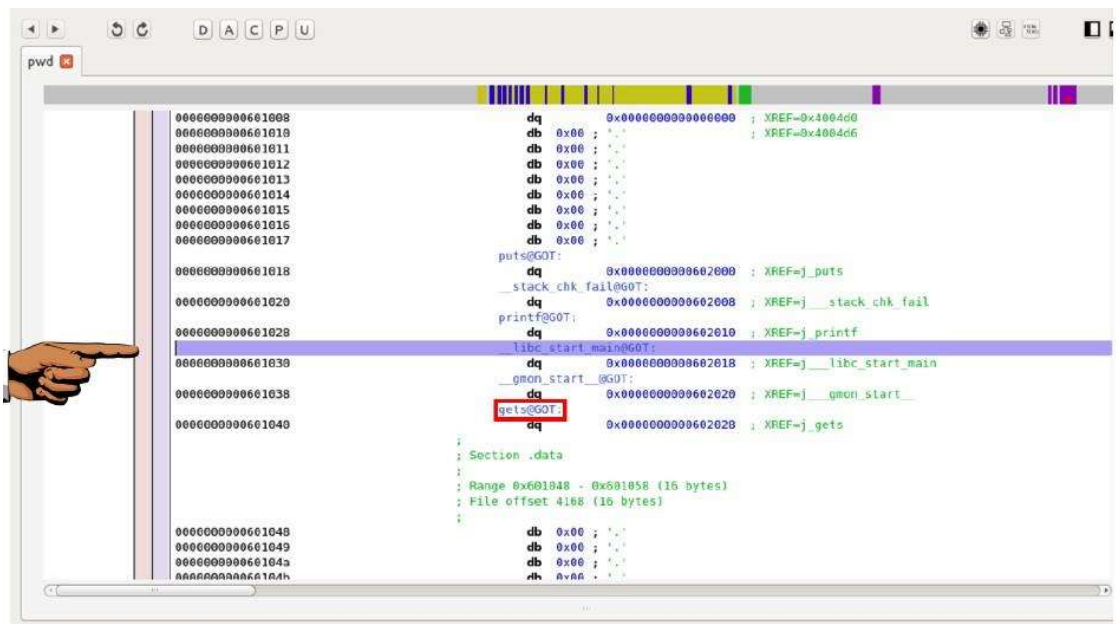
This section contains string constants, such as "Enter password" and "Fail!", as shown below.



## Viewing Data

In the Navigation bar, drag the little red arrow into the purple bar at the far right, as shown below.

This section contains the Global Offset Table, which we have used before in exploitation, as shown below.



## Saving a Screen Image

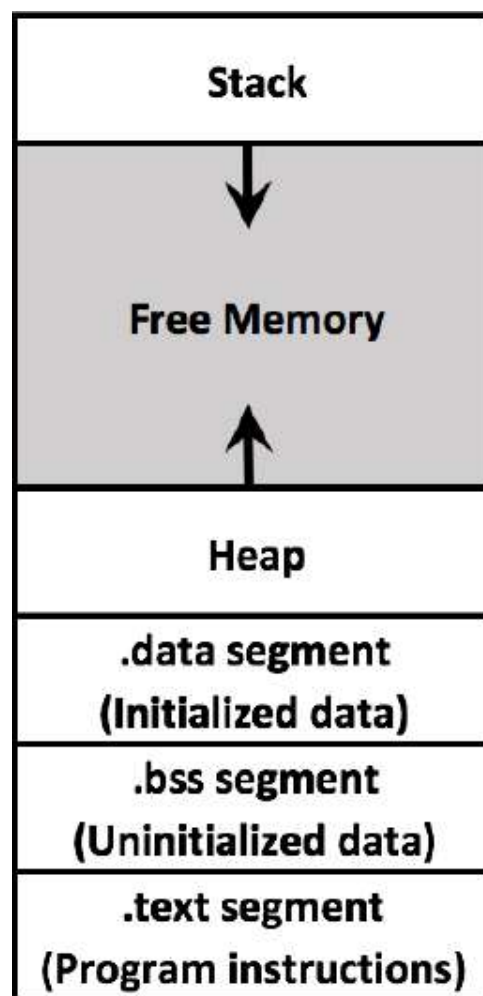
Make sure puts@GOT is visible, as shown above.

Press the PrintScr key to copy the whole desktop to the clipboard.  
YOU MUST SUBMIT A FULL-SCREEN IMAGE FOR FULL CREDIT!  
Paste the image into Paint.  
Save the document with the filename "YOUR NAME Proj 4xa", replacing "YOUR NAME"  
with your real name.

## Pass 2: Segments, Sections, Symbols, & Strings

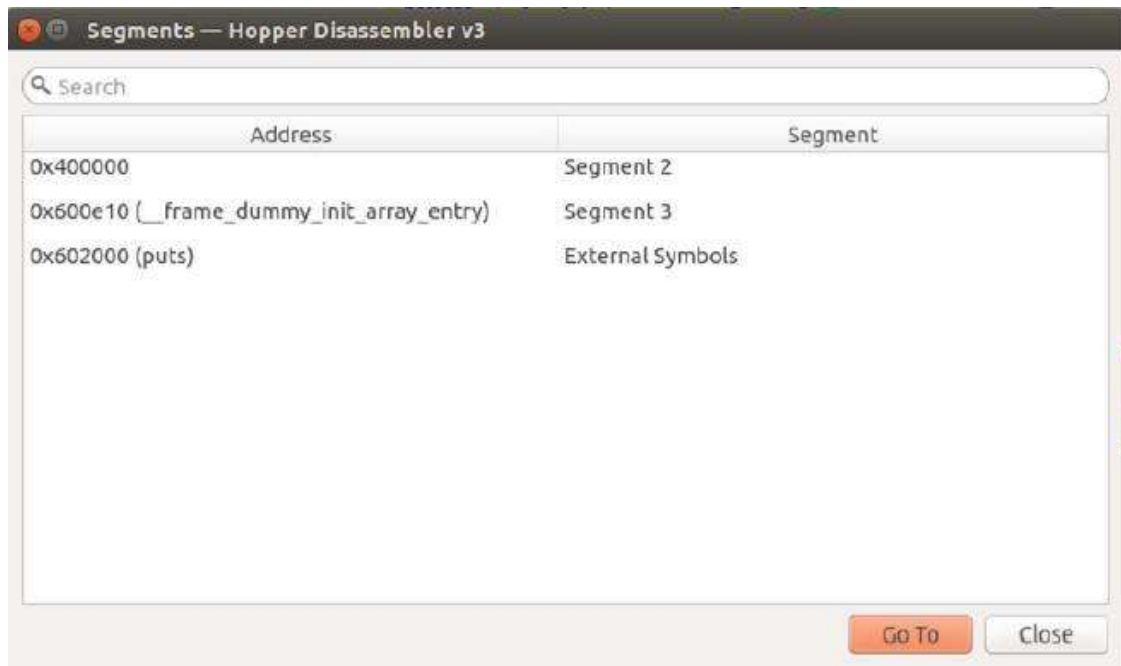
### Memory Layout

Here are the main sections of memory used by a Linux executable:



### Navigating by Segments

From the Hopper menu bar, click Navigate, "Show Segment List".  
A "Segments" window appears, as shown below.

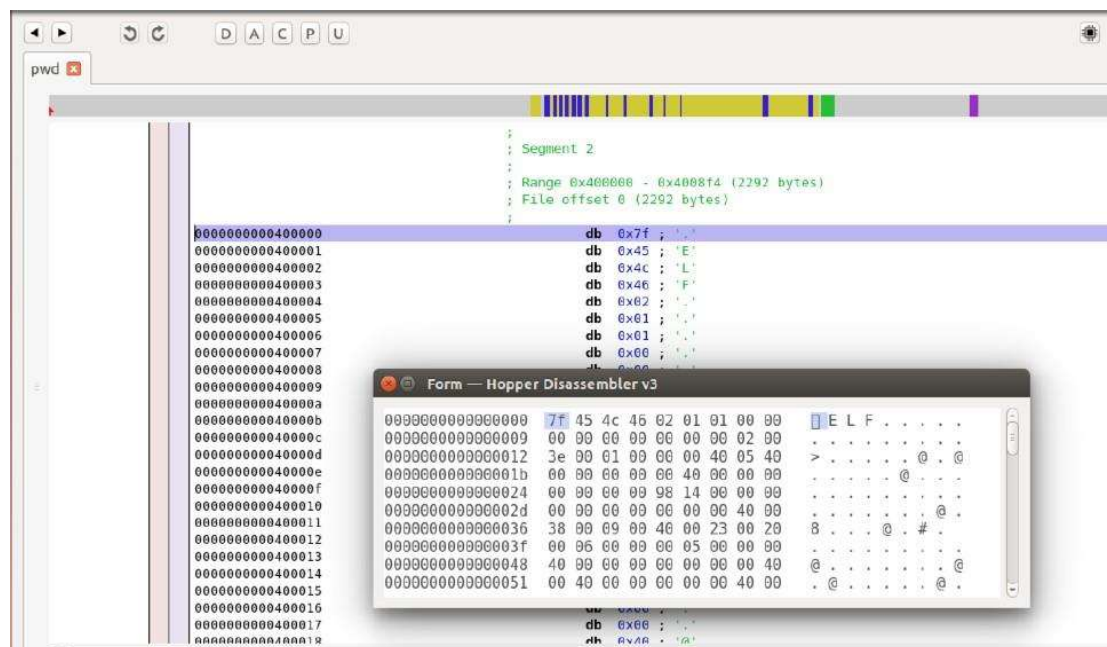


This isn't much use for understanding program flow, but you can see the start of the program in memory.

In the Segments box, click 0x400000 and click the "Go To" button.

A series of bytes appears, beginning with ".ELF", as shown below.

To see the code in a more readable view, click Windows, "Show Hexadecimal Editor".



## Navigating by Sections

From the Hopper menu bar, click Navigate, "Show Section List".

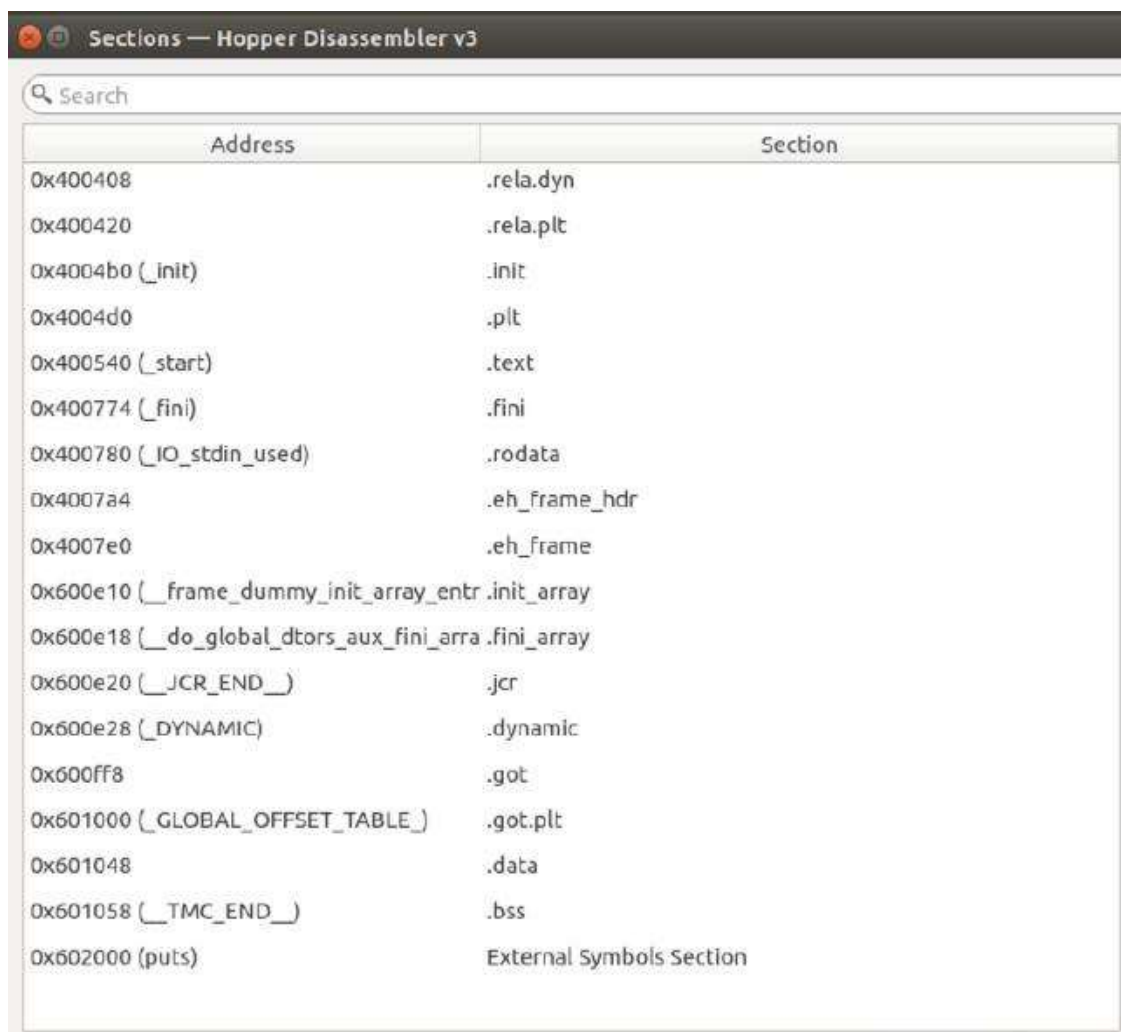
A "Sections" window appears, as shown below.

This has several familiar items in it:

**\_init** : prepares to start the program

**\_start** : (.text section) More preparation to launch the program

**\_GLOBAL\_OFFSET\_TABLE** : (.got.plt section) pointers to library functions, which we have exploited in the past



Address	Section
0x400408	.rela.dyn
0x400420	.rela.plt
0x4004b0 (_init)	.init
0x4004d0	.plt
0x400540 (_start)	.text
0x400774 (_fini)	.fini
0x400780 (_IO_stdin_used)	.rodata
0x4007a4	.eh_frame_hdr
0x4007e0	.eh_frame
0x600e10 (__frame_dummy_init_array_init_array)	
0x600e18 (__do_global_dtors_aux_fini_array)	
0x600e20 (__JCR_END__)	.jcr
0x600e28 (__DYNAMIC)	.dynamic
0x600ff8	.got
0x601000 (_GLOBAL_OFFSET_TABLE_)	.got.plt
0x601048	.data
0x601058 (__TMC_END__)	.bss
0x602000 (puts)	External Symbols Section



## Using Symbols

At the top right, click the leftmost of the three "Show/Hide Panes" buttons. This shows the "Symbols & Strings" pane, as shown below.

On the left side, a list of friendly Labels appears.

In the left pane, click main. The assembly code for the main() routine appears, as shown below.

## Using Strings

In the left pane, click the Strings tab.

Click Fail!.

The right pane shows the ASCII string "Fail!", as shown below.

Suppose we want to trick the code into accepting any password, and not printing "Fail!".

We'd like to see the code that uses the string "Fail!".

In the right pane, to the right of the word "Fail!", there is a comment beginning with XREF=.

Double-click the "main+18 text. The code appear in main() that prints

"Fail!", as shown below.

## Control Flow Graph

At the top right of Hopper, click the "Control Flow Graph" button, as shown below.

The Control Flow Graph appears, as shown below.

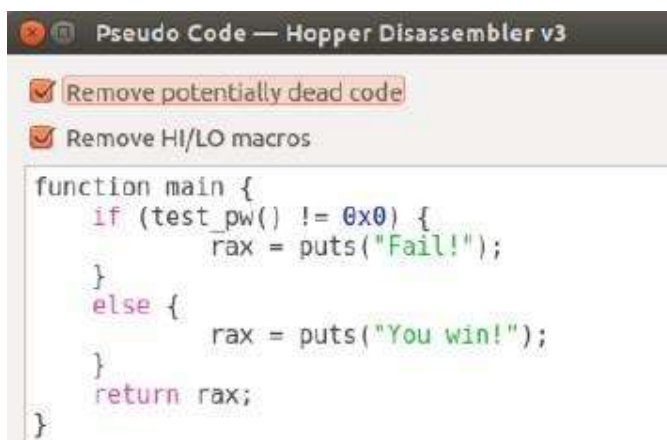
It shows that main calls a subroutine named test\_pw, then tests the return value, and jumps to one of two puts calls to print messages.

The Control

The Control Flow Graph is pretty, but it doesn't show which choice prints "Win!" and which prints "Fail!".

Close the Control Flow Graph.

## Pseudocode



```
function main {  
    if (test_pw() != 0x0) {  
        rax = puts("Fail!");  
    }  
    else {  
        rax = puts("You win!");  
    }  
    return rax;  
}
```

## Understanding the Assembly Code

Now the assembly code is easier to understand. Starting at address 4006d1, the code calls test\_pw. It then uses test to see if the return value is zero. If it is, it jumps to location 4006e6 and prints "You Win!". Otherwise it doesn't jump, and prints "Fail!". So a simple way to make the program accept any password is to fill the bytes outlined in green below with NOPs.

## Saving a Screen Image

Make sure the hex codes starting with BF95 and ending with EB0A are visible, as shown above.

Press the PrintScrn key to copy the whole desktop to the clipboard.

YOU MUST SUBMIT A FULL-SCREEN IMAGE FOR FULL CREDIT!

Paste the image into Paint.

Save the document with the filename "YOUR NAME Proj 4xb", replacing "YOUR NAME" with your real name.

## Exiting Hopper

The paid version of Hopper lets you modify code and save the modified version, but not the free demo version.

Close Hopper.

## Modifying the Executable

In a Terminal window, execute this command to install hexedit:

```
apt-get install hexedit -y
```

Enter the password student when you are prompted to.

In a Terminal window, execute these commands to copy the pwd executable to a new file named pwd2, and open it in the hexedit hex editor:

```
cp pwd pwd2
hexedit pwd2
```

The binary opens in hexedit, as shown below.

The first four characters are .ELF, shown on the right side in ASCII.

In hexedit, press Ctrl+S.

Enter this "hexa string":

BF95

Press Enter.

The cursor moves to location 6DA, as shown below.

Carefully type 90 over twelve bytes, as shown below.

To save the file, press Ctrl+X, Y, Enter.

In a Terminal window, execute this command to run the file:

```
./pwd2
```

## Pass 3: Debugging

Hopper provides a graphical interface to the gdb debugger. We'll use that next.

### Starting the Hopper Debugger

From the Ubuntu desktop, at the top left, click the square reddish Search button. In the search field, type



### Starting Hopper

From the Ubuntu desktop, at the top left, click the square reddish Search button. In the search results, click "Hopper Disassembler v3". In the "Registration" box, click "Try the Demo". Hopper opens. From the Hopper menu bar, click File, "Read Executable to Disassemble...". Navigate to the pwd file you created above and double-click it.

### Setting a Breakpoint

Before opening the debugger, we'll set a breakpoint in Hopper. In the "Symbols & Strings" pane, on the Tags tab, click main. In the right pane, in the reddish vertical stripe, next to the first instruction in main, click the mouse. A red dot appears, showing that this is now a breakpoint, as shown below.

### Starting the Debugger

At the top right of Hopper, click the Debugger button, as shown below. A GDB window appears, as shown below. The most useful buttons are labelled below.

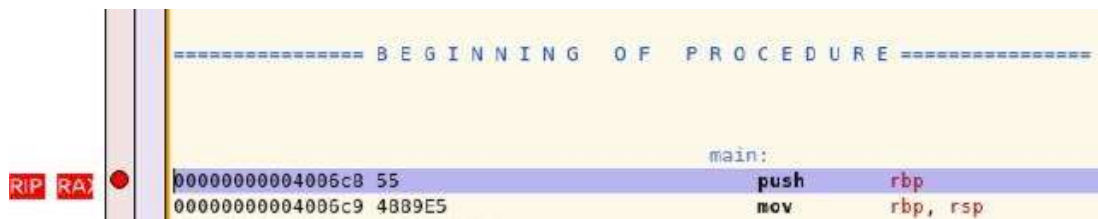
## Starting the Program

In the GDB window, click the Continue button.

The program launches, and stops at your breakpoint.

The disassembler window labels the breakpoint with a red RIP marker, indicating that the Instruction Pointer is here, as shown below.

It also labels this line with a red RAX.



The GDB window shows the current contents of the processor's registers.

As shown below, both RIP and RAX have the same value--the address of the breakpoint.

### Troubleshooting

If you make any sort of mistake using the debugger, you cannot easily close it and restart it.

Instead, you must use command-line utilities to kill all hopper processes, as explained in the "Exiting the Debugger" section at the end of these instructions.

## Using "Step into"

In the GDB window, click the "Step into" button.

In the lower left of the window, the "Callstack" pane now shows a location of }main + 0x1", as shown below.

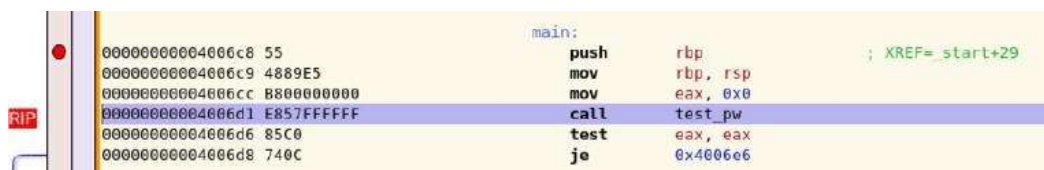
The Disassembler window also shows that the program has moved forward one instruction, as shown below.

The red RIP has moved down one line, and the red RAX still points to the first command in main(), as shown below.

In the GDB window, click the "Step into" button twice more.

In the lower left of the window, the "Callstack" pane now shows a location of }main + 0x9", as shown below.

The Disassembler window now shows that we are at the "call test\_pw" instruction, as shown below.



In the GDB window, click the "Step into" button once more.

In the lower left of the window, the "Callstack" pane now shows two lines: we are still in main, but also in the subroutine test\_pw, called by main, as shown below.

This is the meaning of "Step into" -- it executes only a single instruction, even if that instruction is a call and enters a new subroutine.

The Disassembler window now shows that we are inside the "call test\_pw" function, as shown below.

In the GDB window, click the "Step into" button nine more times.

The Disassembler window shows that we about to call the j\_printf function, as shown below.

In the GDB window, click the "Step into" button once more.

In the lower left of the window, the "Callstack" pane now shows three lines: we are still in main, and in the subroutine test\_pw, but we are also in the subroutine j\_printf, as shown below.

The Disassembler window now shows that we are inside the "j\_printf" function, as shown below.

## Using "Step out"

We are about to enter the Global Offset Table and then enter the C library.

We aren't interested in debugging the C library--we want to debug the C code we wrote. So it doesn't make sense to keep on using "Step in".

What we really want to do is "Step out", to get back to test\_pw. In the GDB window, click the "Step out" button once.

In the lower left of the window, the "Callstack" pane now shows only two lines again: main, and test\_pw, as shown below.

## Using "Step over"

This is the level we want to stay at, inside the code we wrote.

To stay at this level, we use the "Step over" button.

In the GDB window, click the "Step over" button three times.

In the lower left of the window, the "Callstack" pane now shows two lines, but they are grayed out, as shown below.

The Disassembler window no longer shows which instruction we are on, as shown below.

What has happened? We have stepped over the j\_gets function, and the program is waiting for user input from inside that function.

## Interacting with the Console

In the GDB window, click the "Application Output" tab.

Here you see a warning message, followed by the "Enter password:", prompt, as shown below.

In the bar at the bottom, type in your name (not the literal text "YOURNAME", as shown below) and press Enter.

Your name appears inside the window, as shown below.





## Saving a Screen Image

Make sure YOURNAME and "Application Output" are visible, as shown above.

Press the PrintScrn key to copy the whole desktop to the clipboard.

YOU MUST SUBMIT A FULL-SCREEN IMAGE FOR FULL CREDIT!

Paste the image into Paint.

Save the document with the filename "YOUR NAME Proj 4xc", replacing "YOUR NAME" with your real name.

## Exiting the Debugger

The clean way to exit is to first click the square Stop button in the debugger, then click the red X in the top left to close the debugger window, and then close the Hopper disassembler.

If that process fails (and it often does), do this:

In a Terminal window, execute this command:

```
ps aux | grep hopper
```

Now use the kill command to kill each process, one by one, by process ID number, as shown below.

# Practical 7

## Using Jasmin to run x86 Assembly CodeWhat You Need for This Project

Any computer, running any OS

### Purpose

To practice writing and running basic x86 assembly code, using the Jasmin interpreter.

### Install Java

Open a Web browser and go to

<http://java.com>

Click the "Do I have Java?" link. Follow the instructions on your screen to download and run a Java applet. If you don't have Java, download and install it.

Note: If you are using the Mac, you should use Safari, not Chrome.

### Download Jasmin

In a Web browser, go to

<http://sourceforge.net/projects/tum-jasmin/files/>

On the "Looking for the latest version?" line, click the link to download the latest version of Jasmin. When I did it, it was Jasmin-1.5.8.jar.

If you don't want the drawing of a partially undressed woman on the splash screen, use this version:

Download politically correct Jasmin without the cheesecake

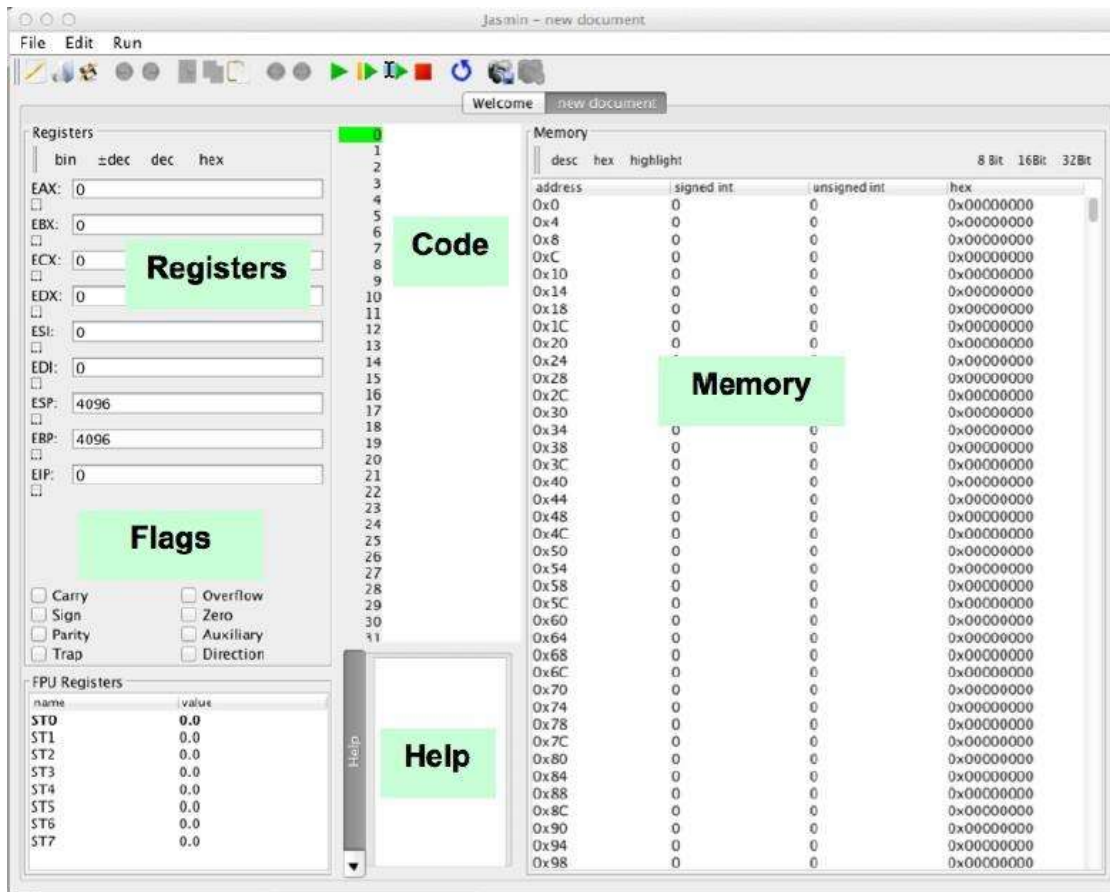
### Understanding the Jasmin Window

Double-click the Jasmin-1.5.8.jar file you downloaded.

Jasmin launches, with a cringe-worthy pinup on it.

Click the "New File" button.

Look over the window, referring to the diagram below:



## Registers

Data used during processing is stored in the registers EAX, EBX, ECX, and EDX.

The ESP (Extended Stack Pointer) contains the address of the top of the Stack.

The EIP (Extended Instruction Pointer) contains the address of the the next instruction to be processed.

## Flags

These one-bit values that are used for branching. For example the JZ instruction will jump if the Zero flag is 1 (set), and the JNZ instruction will jump if the Zero flag is 0 (cleared).

## Code

This is where you type in commands, such as `mov eax,4`

## Help

Help messages appear here.

## Memory

This processor has  $0x1000 = 4096$  bytes of RAM, which is not enough to run complete modern programs, but plenty for running little assembly programs for learning purposes.

With the Memory pane scrolled to the top, as shown in the image above, you see memory that the program will use to store data during processing.

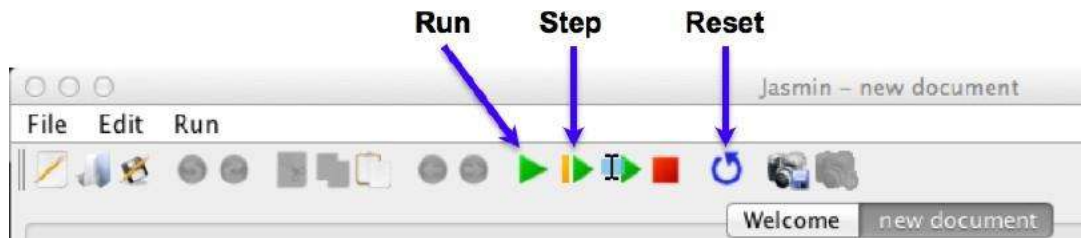
Scroll this pane to the bottom to see the Stack, which starts at address `0x1000` and grows downward.

## Using mov Instructions

In the Code section, type in these instructions.

These instructions move the number 4 into eax, and the number 6 into ebx.

At the top of the Jasmin window, click the green Run button, as shown below.



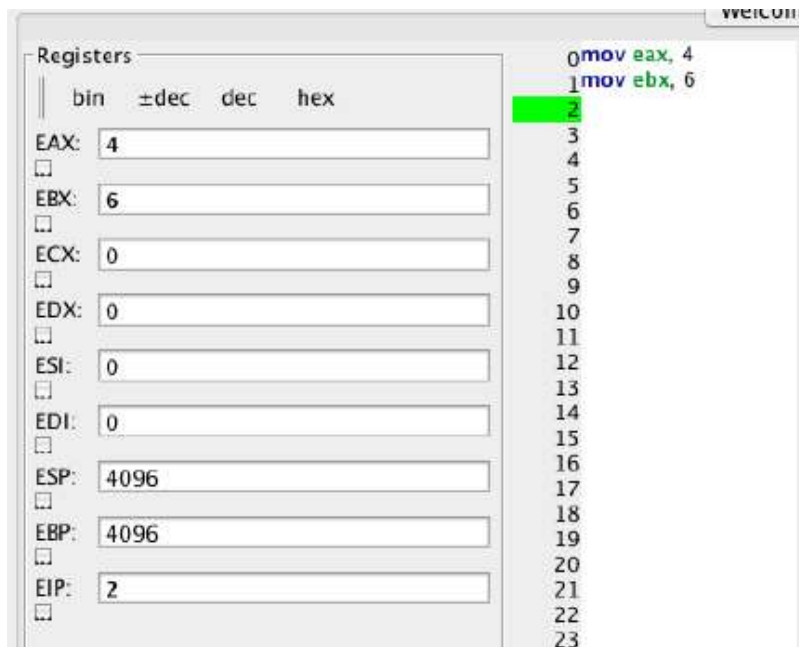
The program runs. When it stops, notice these things, as shown below:

EAX contains 4

EBX contains 6

EIP contains 2, because instructions 0 and 1 have been executed

In the Code area, instruction 2 is highlighted in green, indicating that it is the next instruction to be processed.



## Storing Results in Memory

Add more lines to your Code section to make your program look like this:

```
mov eax, 4
mov ebx, 6
mov [eax], ebx
mov ecx, eax
add ecx, ebx
mov [eax+4], ecx
```

Here's what these instructions do:

```
mov eax, 4
mov ebx, 6
mov [eax], ebx
mov ecx, eax
add ecx, ebx
mov [eax+4], ecx
```

Run the program. When it completes, you should see these results, as shown below:

```
EAX = 4
EBX = 6
ECX = 10
Memory location 0x4 contains 6
Memory location 0x8 contains 10
```

## Saving a Screen Image

Make sure the five items listed above are visible.

Press the PrintScrn key to copy the whole desktop to the clipboard.

**YOU MUST SUBMIT A FULL-SCREEN IMAGE FOR FULL CREDIT!**

Paste the image into Paint.

Save the document with the filename "YOUR NAME Proj 5a", replacing "YOUR NAME" with your real name.

## Using the Stack

In Jasmin, click File, New.

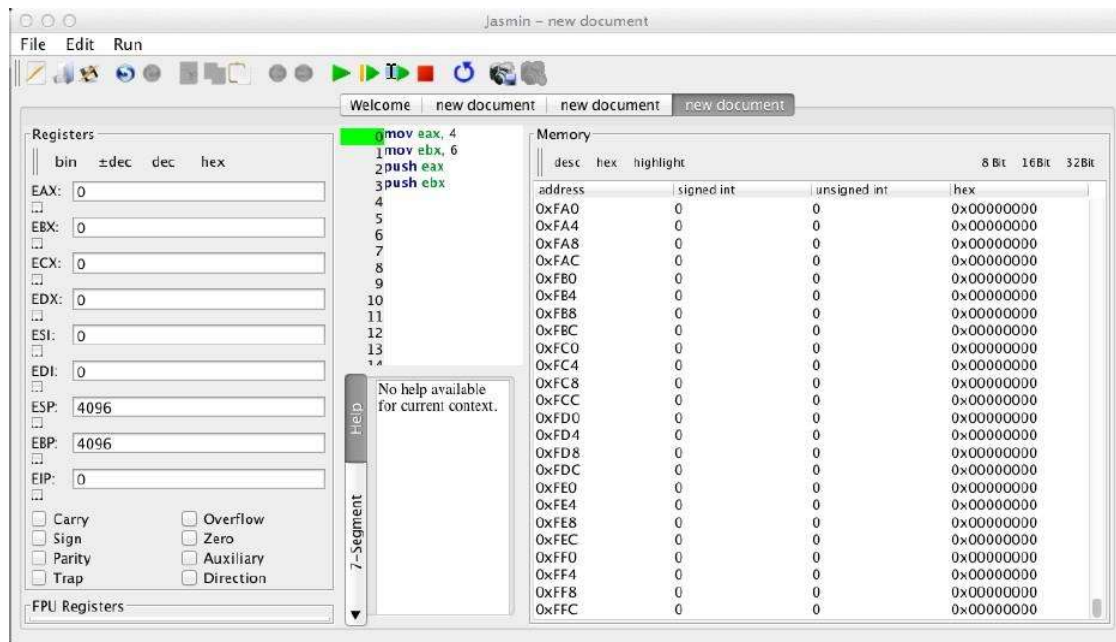
In the Code section, type in these instructions.

```
mov eax, 4
mov ebx, 6
push eax
push ebx
```

Before running the program, notice the ESP: it contains 4096, as shown below.



4096 is 0x1000 in hexadecimal--this is where the Stack ends.



Scroll down in the Memory pane to see the last values. As show above, the last location is at 0xFFC. This value is 32 bits long, so it contains four bytes, at locations 0xFFC, 0xFFD, 0xFFE, and 0xFFF. The ESP points to the next byte, 0x1000.

## Understanding Push

At the top of the Jasmin window, click the green Run button.

These instructions move the number 4 into eax, and the number 6 into ebx. Then both values are pushed onto the stack.

Notice these things, as shown below:

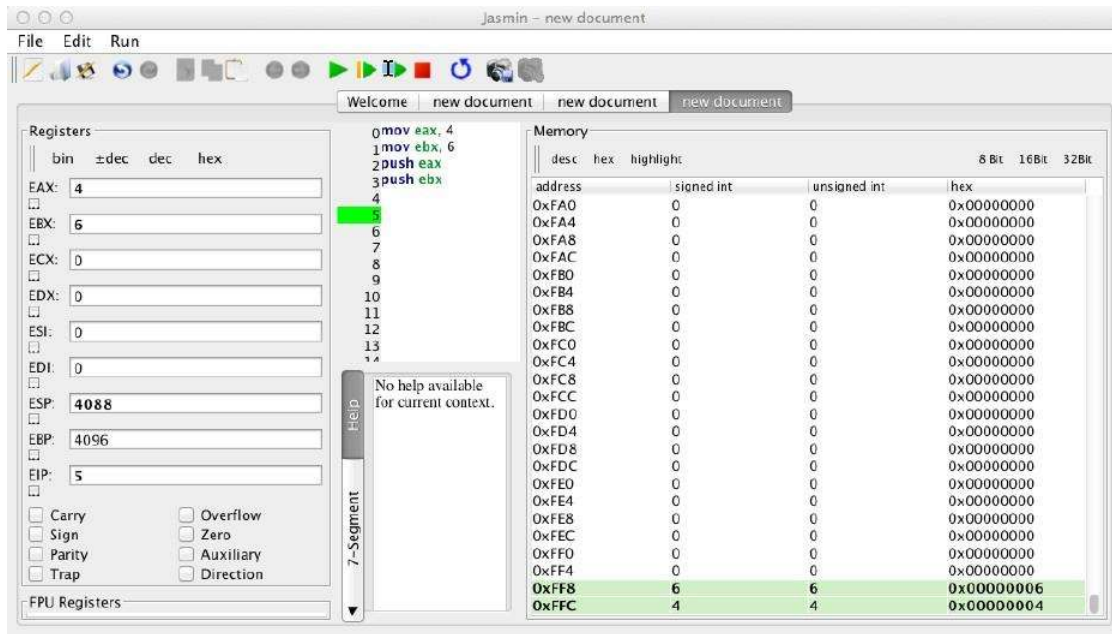
EAX contains 4

EBX contains 6

ESP contains 4088, which is 0xFF8, the new top of the stack.

Memory location 0xFFC contains 4, the first value pushed onto the stack.

Memory location 0xFF8 contains 6, the second value pushed onto the stack.



## Understanding Pop

Add a pop instruction to your code, so it now looks like this:

```
mov eax, 4
```

```
mov ebx, 6
```

```
push eax
```

```
push ebx
```

```
pop ecx
```

Run the code.

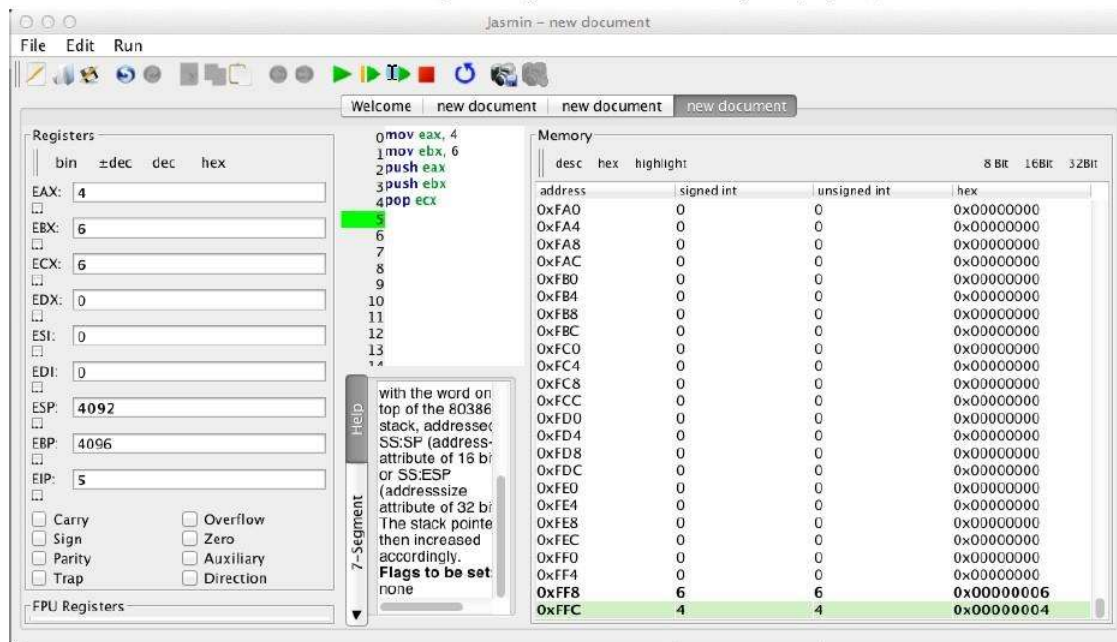
Notice these things, as shown below:

ECX contains 6, the value popped off the top of the stack.

ESP contains 4092, which is 0xFFC, the new top of the stack.

Memory location 0xFFC contains 4, the first value pushed onto the stack.

Memory location 0xFF8 contains 6, which is now the top value on the stack.



## Reversing a Sequence

In Jasmin, click File, New.

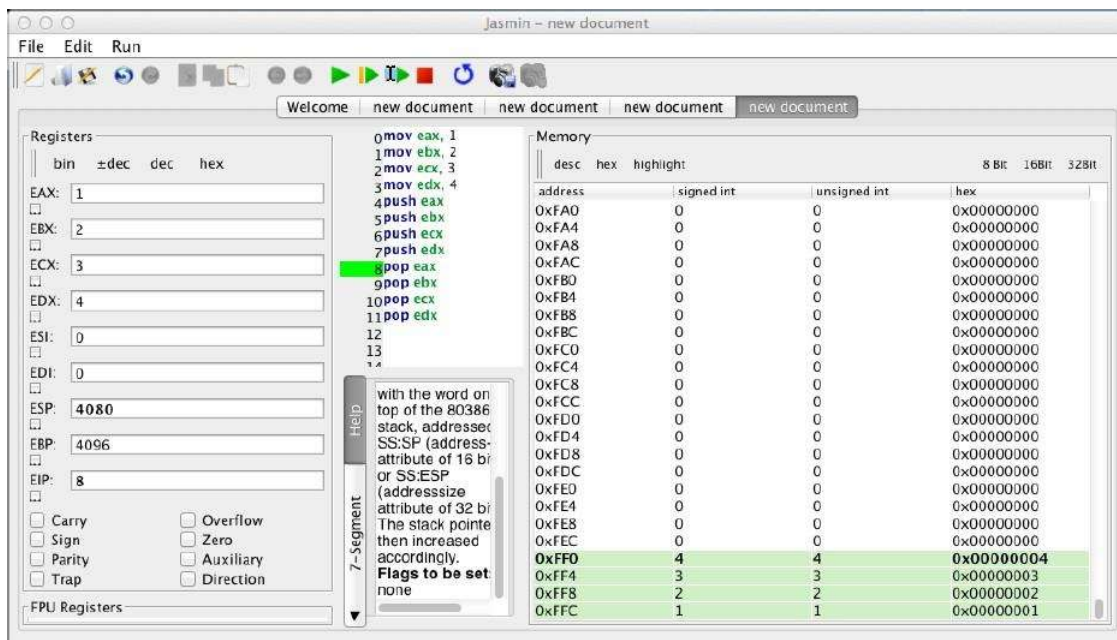
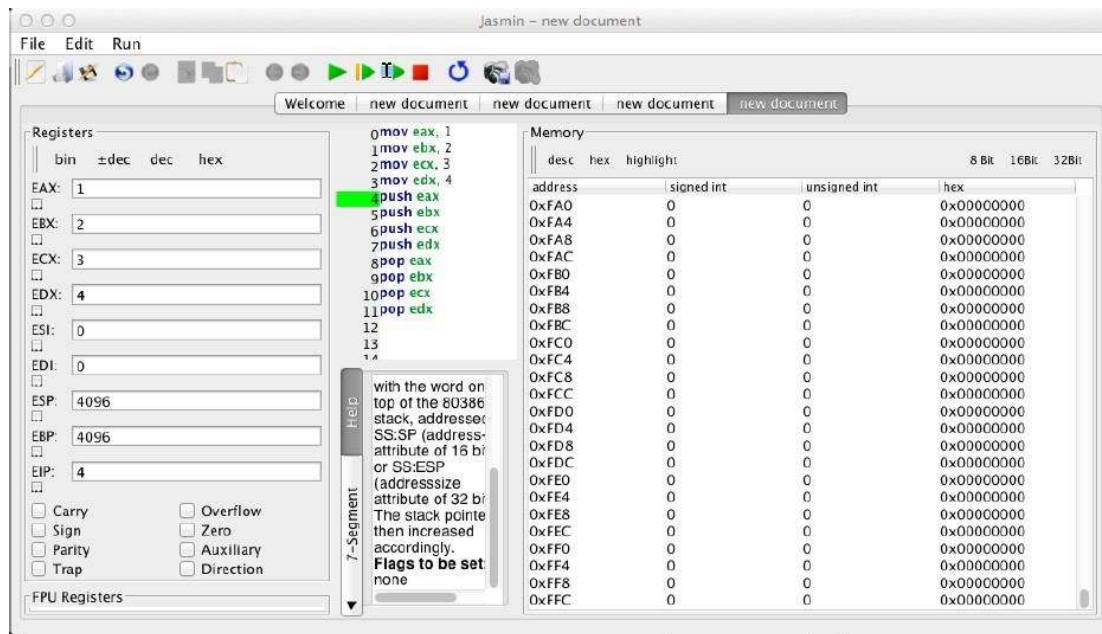
In the Code section, type in these instructions.

```
mov eax, 1
mov ebx, 2
mov ecx, 3
mov edx, 4
push eax
push ebx
push ecx
push edx
pop eax
pop ebx
pop ecx
pop edx
```

These instructions load values into the four registers, push them onto the stack in order, and pop them off the stack in order.

However, since the stack is a FILO (First In, Last Out) structure, this reverses the order of the values.

Push the Step four times to execute only the first four instructions, as shown below:



## Saving a Screen Image

Make sure the four items listed above are visible.

Press the PrintScr key to copy the whole desktop to the clipboard.

**YOU MUST SUBMIT A FULL-SCREEN IMAGE FOR FULL CREDIT!**

Paste the image into Paint.

Save the document with the filename "YOUR NAME Proj 5b", replacing "YOUR NAME" with your real name.

# Practical 8

## Assembly Code Challenges

### Challenge 1: Secret Message

Download this code and run it in Jasmin.

p5xa.asm

The code loads encrypted values onto the stack, decrypts them to place a secret message in ASCII into memory,

and then erases the memory.

Use Jasmin to run the code until it has decrypted the message, but has not erased it yet, and capture a whole desktop

image of the decrypted RAM contents. Save the image as "Proj 5xa from YOUR NAME".

Convert the message to ASCII and type it in the body of your email message.

### Challenge 2: Fibonacci Numbers

Write an assembly language program to display the first 10 Fibonacci numbers, as shown below:

55
34
21
13
8
5
3
2
1
1
0

Each Fibonacci number is the sum of the first two previous ones, and the first two are:

$F_0 = 0$

$F_1 = 1$

$F_2 = F_1 + F_0 = 1$

$F_3 = F_2 + F_1 = 1 + 1 = 2$

$F_4 = F_3 + F_2 = 2 + 1 = 3$

Tip: You can learn how to use loops here:

[http://wwwi10.lrr.in.tum.de/~jasmin/tutorials\\_basic.html](http://wwwi10.lrr.in.tum.de/~jasmin/tutorials_basic.html)

Turn in a screen shot showing your results, and attach an ASM file of your code, saved from Jasmin.



# Practical 9

## IDA Pro (Lab 5-1)

### Purpose

You will practice using IDA Pro.

You should already have the lab files, but if you don't, do this:

### Downloading the Lab Files

In a Web browser, go here:

<http://practicalmalwareanalysis.com/labs/>

Download and unzip the lab files.

### Downloading and Installing IDA Pro

In your Windows machine, open a Web browser and go to

[https://www.hex-rays.com/products/ida/support/download\\_freeware.shtml](https://www.hex-rays.com/products/ida/support/download_freeware.shtml)

Download "IDA Freeware" and install it.

If that link is down, use this alternate download link:

<https://samsclass.info/126/proj/idafree50.exe>

### Opening Lab05-01.dll in IDA Pro

Launch IDA Pro. Click OK. Click New. Click the "PE Dynamic Library" icon and click OK.

Navigate to Lab05-01.dll and open it.

### Q 1: Finding the Address of DLLMain

In IDA Pro, click Windows, "Functions window".

Click the "Function name" header to sort by name and scroll to the top.

Your image should show the location of DLLMain, as shown below:

Press the PrntScrn key to capture an image of the whole desktop.

Open Paint and paste the image in with Ctrl+V.

Save this image with the filename "Proj 6a from YOUR NAME".

## Q 2: Find the import for gethostbyname

In IDA Pro, click Windows, Imports. Click the Name header to sort by name. Find "gethostbyname" -- note that capital letters and lowercase letters sort into separate groups.

Widen the Address column to make the entire address visible.

Your image should show the location of gethostbyname, as shown below:



Save a full-desktop image with the filename "Proj 6b from YOUR NAME".

## Q 5: Count Local Variables for the Subroutine at 0x10001656

In IDA Pro, click Windows, "IDA View-A". Press the SPACEBAR to get to text view.

Press g to Go. Enter the address 0x10001656 and click OK.

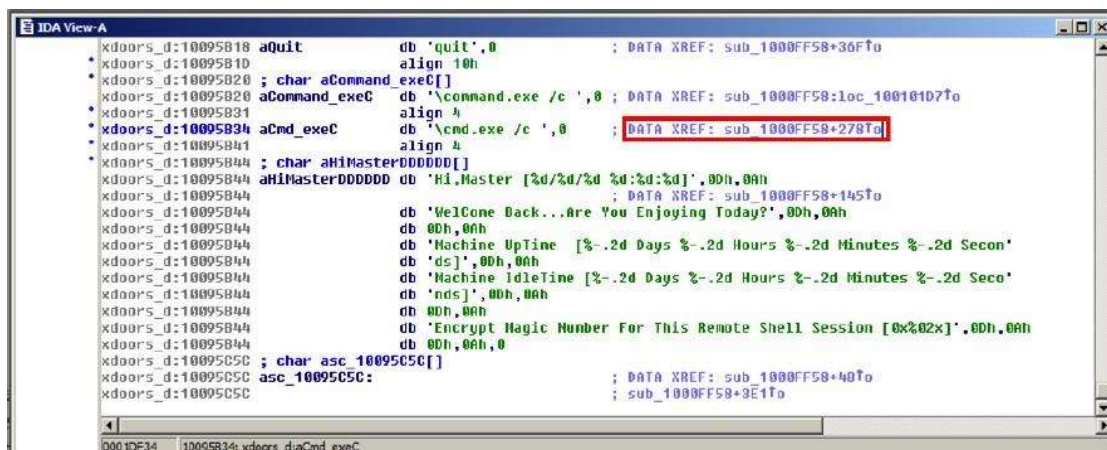
Scroll up to show the comments IDA added to the start of the function, listing its local variables, as shown below:

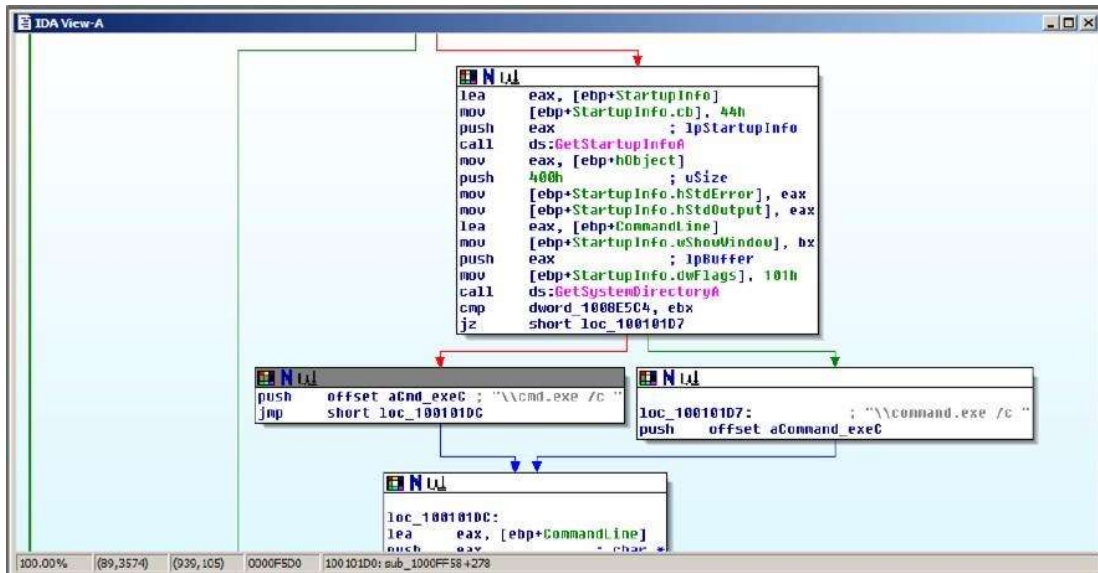


## Q 8: Finding the Purpose of the Code that References \cmd.exe /c

In IDA Pro, click Windows, Strings. Make the window larger. Sort by String. Find the String "\\cmd.exe /c" and double-click it. The function opens in text view, as shown below.

In the line containing "\\cmd.exe /c", double-click the address to the right of "XREF", as indicated by the red outline in the image below.





# Practical 10

## Disassembling C on Windows Part 3What You Need

A Windows machine, real or virtual. I used Windows 7.

Visual Studio Express, which you installed in a previous project.

IDA Pro Free, which you installed in a previous project.

### Purpose

You will write a small C programs using if statements, compile it, and examine it in the IDA Pro disassembler to learn what it looks like in assembly language.

### Starting Visual Studio Express

Click Start, "All Programs", "Microsoft Visual Studio Express", "VS Express for Desktop".

At the "Product key" screen, click Cancel.

### Making a New C Program

From the "Visual Studio Express 2012" menu, click FILE, "New Project...".

In the "New Project" window, on the left, expand the "Visual C++" container.

Click Win32.

In the center pane, accept the default selection of "Win32 Console Application".

At the bottom of the "New Project" window, type a Name of YOURNAME-6xa, replacing "YOURNAME" with your own name. Do not use any spaces in the name.

In the "Location" line, click the Browse button and navigate to a folder you have permission to save files in, such as your desktop.

Click the "Select folder" button.

In the "New Project" window, click OK.

A box opens, titled "Welcome to the Win32 Application Wizard".

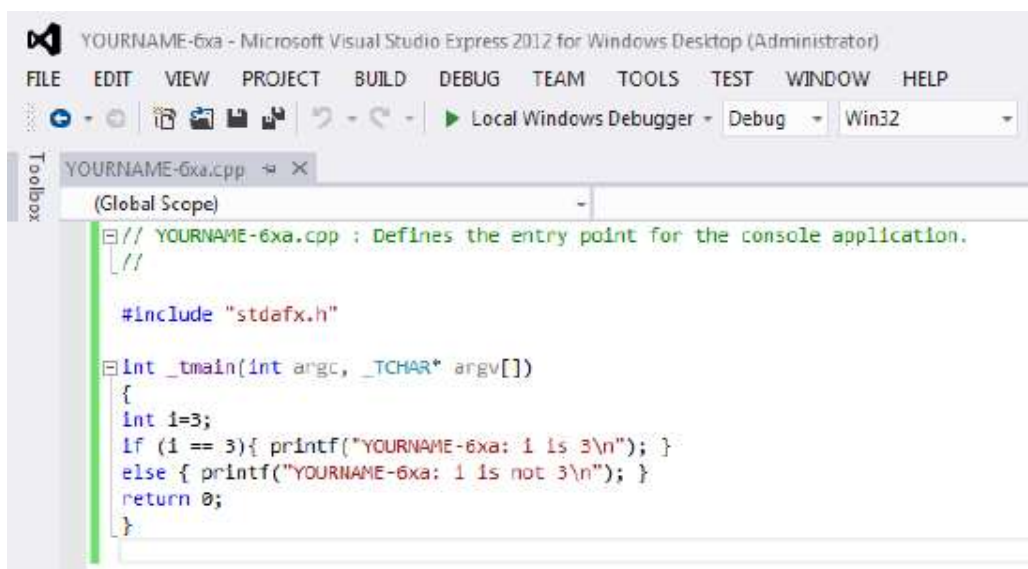
Click Next. In the next screen, accept the default settings and click Finish.

A window opens, showing a simple C program.

Modify this program to match the code shown in text and the image below.

Do not use the literal string "YOURNAME"--replace it with your own name.

```
// YOURNAME-6xa.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{ int i=3;
  if (i == 3){ printf("YOURNAME-6xa: i is 3\n"); }
  else { printf("YOURNAME-6xa: i is not 3\n"); }
  return 0;
}
```



## Compiling your Program

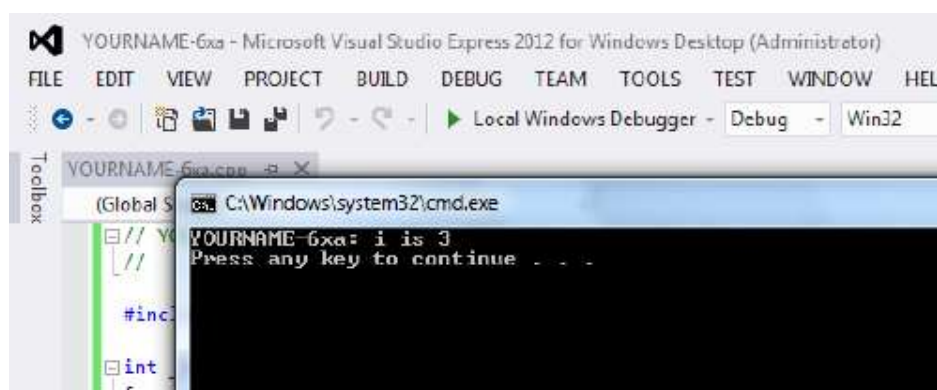
Click BUILD, "Build Solution".

You should see the message "Build: 1 succeeded" at the bottom of the window. If you see errors, you need to correct them and re-compile the program.

## Running your Program

Click DEBUG, "Start Without Debugging".

A Command Prompt window opens, showing the output of "YOURNAME-6xa: i is 3", as shown below:



## Disassembling the EXE

Click in the Command Prompt window, and press Enter to close it.

Minimize the Visual Studio Express window.

Start IDA Pro Free.

In the "About" box, click OK.

Agree to the license.



Close the Help window.

In the "Welcome to IDA!" box, click the New button.

In the "New disassembly database" box, double-click "PE Executable".

In the "Select PE Executable to disassemble" box, navigate to the folder you used to save your program in Visual Studio Express, probably your desktop.

Double-click the "YOURNAME-6xa" folder.

Double-click the Debug folder.

Double-click the YOURNAME-6xa.exe file.

In the "PE Executable file loading Wizard", click Next, Next, Finish.

A box appears, saying this file was linked with debug information.

Click Yes

IDA Pro loads the file. As before, the graph mode doesn't show the interesting part of the program.

Expand the Strings. Double-click one of the strings containing "YOURNAME-6xa".

The location containing the string appears.

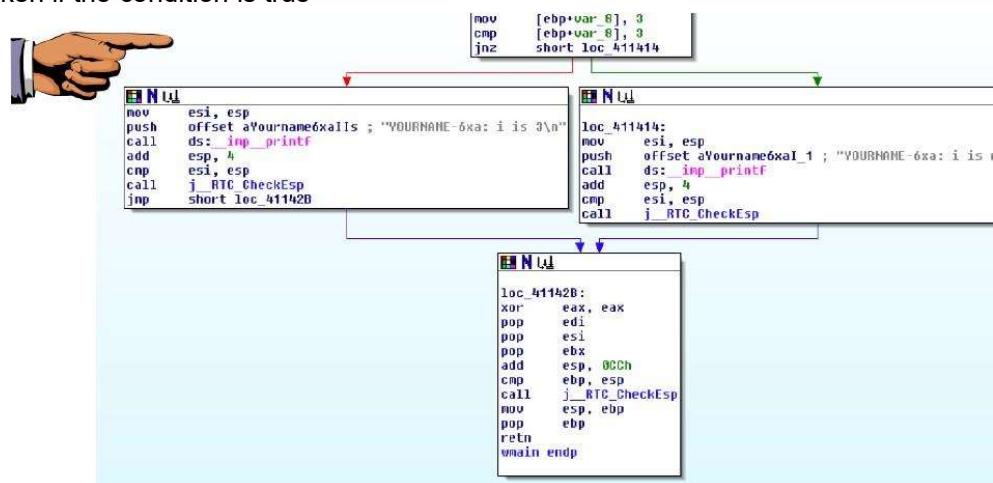
To the right of "YOURNAME-6xa" there is a "DATA XREF" comment. To the right of the "XREF", double-click "wmain".

Now the assembly code that performs the task you wrote in C appears, as shown below.

IDA Pro's graph mode makes if statements easy to understand!

The top box ends with a cmp operation (compare two numbers), and a jnz operation (Jump if Not Zero).

The red arrow shows the path taken if the condition is false, and the green arrow shows the path taken if the condition is true



## Saving the Screen Image

Make sure your image contains these items:

A box at the top with two arrows coming out of it, one red and one green

Both the boxes the arrows point to should contain YOURNAME

On your keyboard, press the PrntScrn key.

Click Start, type in PAINT, and open Paint.

Press Ctrl+V to paste in the image of your desktop.

YOU MUST SUBMIT WHOLE-DESKTOP IMAGES TO GET FULL CREDIT.

Save the image with a filename of "Proj 6xa from YOUR NAME".

## CHALLENGE: 10 Pts. Extra Credit

Modify the C program to perform a three-way test, to see if a variable is less than zero, equal to zero, or greater than zero.

Print appropriate messages containing your name in all three cases.

Compile it and disassemble it, producing assembly code similar to that shown below.

