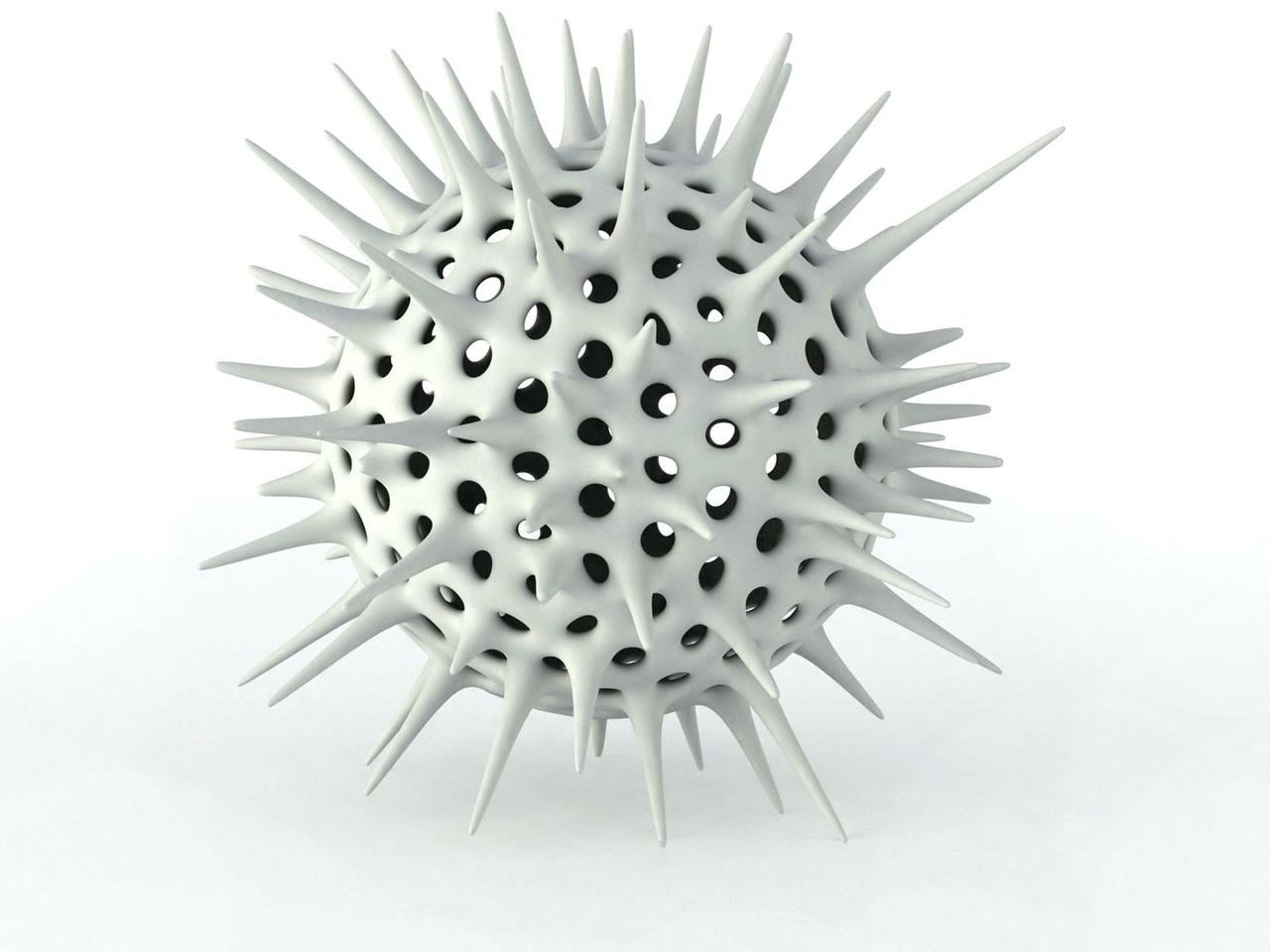
Audrey Long

Project 2: Simple Malware Analysis



|  |  |
| --- | --- |
| **Goal:** | |
|  | The purpose of this lab is to perform an in-depth analysis of a real-world malware sample. In the lab students will initially triage a unknown malware sample, revealing its likely actions and goals. Following this, static (disassembly) then dynamic (debugging) analysis will be performed in order to gain a deeper understanding of the sample. By the completion of the lab, armed with their findings, students will able to remotely task the malware sample, providing full control over the infected target. |

|  |  |
| --- | --- |
|  | **Warning**  This lab contains a live-malware sample. Although it contains no automated self-replication mechanisms, it is still a malicious program, and as such it should be treated with respect.  All analysis should be performed in a isolated Virtual Machine (VM) environment. |

**Lab Components**

|  |  |
| --- | --- |
| **Component** | **Description** |
| Webcam Shots.scr | The malware sample |
| hash.exe | Utility for generating SHA1/MD5 hashes |
| PEInsider | Utility for examining/parsing Portable Executable (PE) headers |
| PEID | Utility for identifying packed binaries |
| upx.exe | Utility for packing/unpacking binaries |
| strings.exe | Utility for exacting all ASCII/Unicode string in a binary |
| procmon.exe | Utility for closely monitoring a process |
| procexp.exe | Utility for listing running processes (and providing information about each) |
| server.py | Custom command & control server that the malware will connect to |
| Python | You should already have this installed from previous assignments, but if not please download Python **2.7.x** **NOT 3.X** |
| mirc.exe | mIRC client installer |

**Part 1 – Initial Triage**

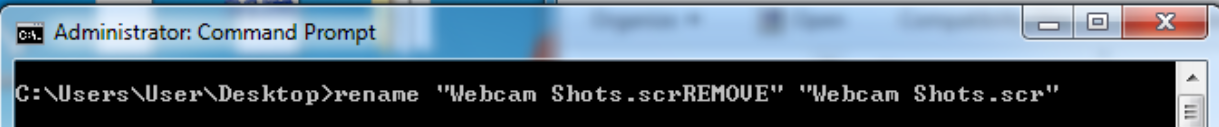
**IMPORTANT: Please follow these instructions VERY CAREFULLY.**

**YOU MUST FIRST DISABLE WINDOWS DEFENDER IN ORDER TO COMPLETE THIS EXERCISE. PLEASE FOLLOW THE INSTRUCTIONS IN THE DisableDefenderRegistry PDF. IF YOU DO NOT DISABLE THIS SERVICE, THE WebcamShots FILE WILL NOT BE ABLE TO BE COPIED TO THE VM.**

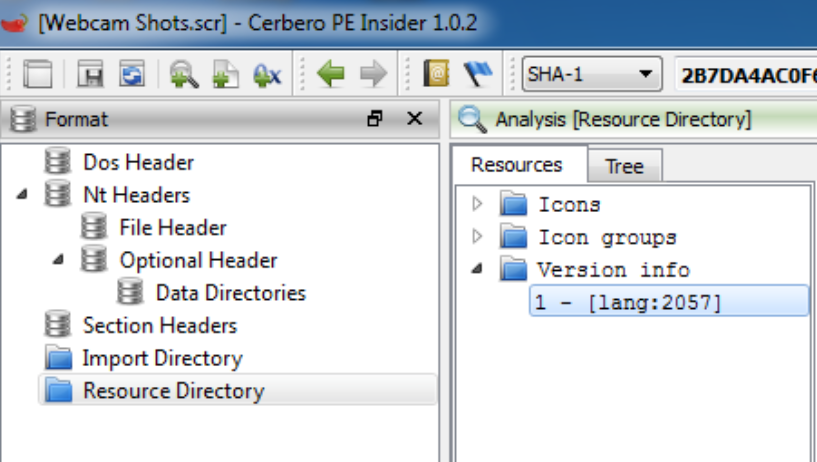
Whenever an unknown binary is obtained for analysis, the initial triage step is essential. This step can provide basic information about the binary, such as its type, its hash, and other indicators that may indicate its characteristics (e.g. maliciousness), or provide insight into likely actions.

In this section, you will perform initial triage on an unknown binary: Webcam Shots.scr

**IMPORTANT: The file is currently named “Webcam Shots.scrREMOVE”. Be sure you rename it to “Webcam Shots.scr” prior to performing this project. The “REMOVE” has been added to prevent accidental installation on other systems.**

****

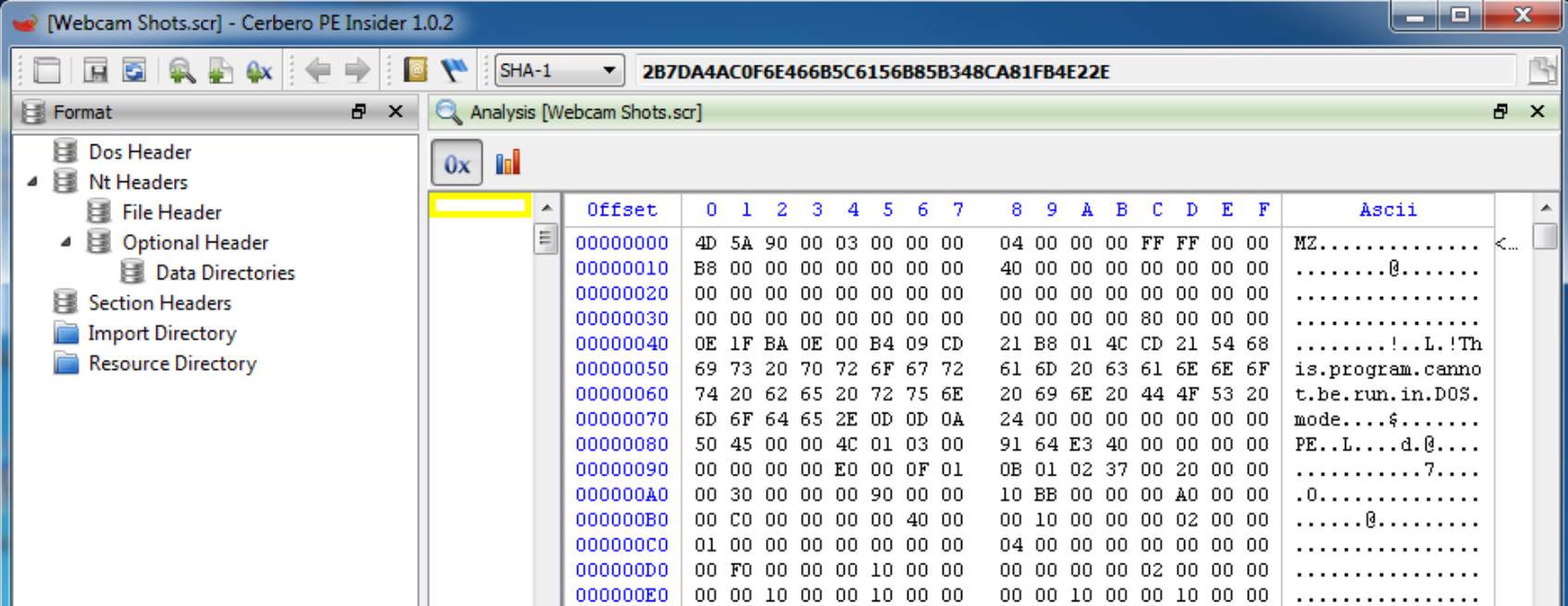
|  |
| --- |
| **Question 0x1** |
| Using tools such as the commandline (cmd.exe), the Windows shell (Explorer.exe), PE Viewer (PEInsider) and the provided hash utility (hash.exe), complete the following table. See the figure below as an example of where to find some of this information from PEInsider. **PEInsider will install to “C:\Program Files (x86)\Cerbero\PEInsider”**  **Answer:**   |  |  | | --- | --- | | **Characteristic** | **Value** | | File Name | Webcam Shots.scr | | File Size | 18,976 bytes | | File Hashes (MD5/SHA1) | MD5 : 38874b8a35cb88d77f37d7beb8ca41bb  SHA1: 2b7da4ac0f6e466b5c6156b85b348ca81fb4e22e | | File Description\* | Generic Host Process for Win32 Services | | Internal Name\* | Winservices.exe | | Company\* | Microsoft Corporation |   \*from resource/version information within the PEInsider utility (See below). |



|  |
| --- |
| **Question 0x2** |
| Do any of the components that make up your answer to question #1 strike you as unusual or perhaps suggestion the file is malicious (or trying to hide its true purpose)?  **Answer:** I think it is unusual that the filename on disk does not match the internal file name. Another strange discovery is the file extension, after doing some research it appears the SRC file extension is used for windows screensavers and the internal name suggests the true file extension is an exe. |

Examining the data found within the Portable Executable (PE) header can often reveal a wealth of information about a binary, such as its type, or API functions that it imports. To answer the next few questions, various PE tools will be used to exact highly informational data from the malware’s PE header.

Open the malware (Webcam Shots.scr) in the PE-viewer tool (PEInsider). Use this utility to answer the next four questions.



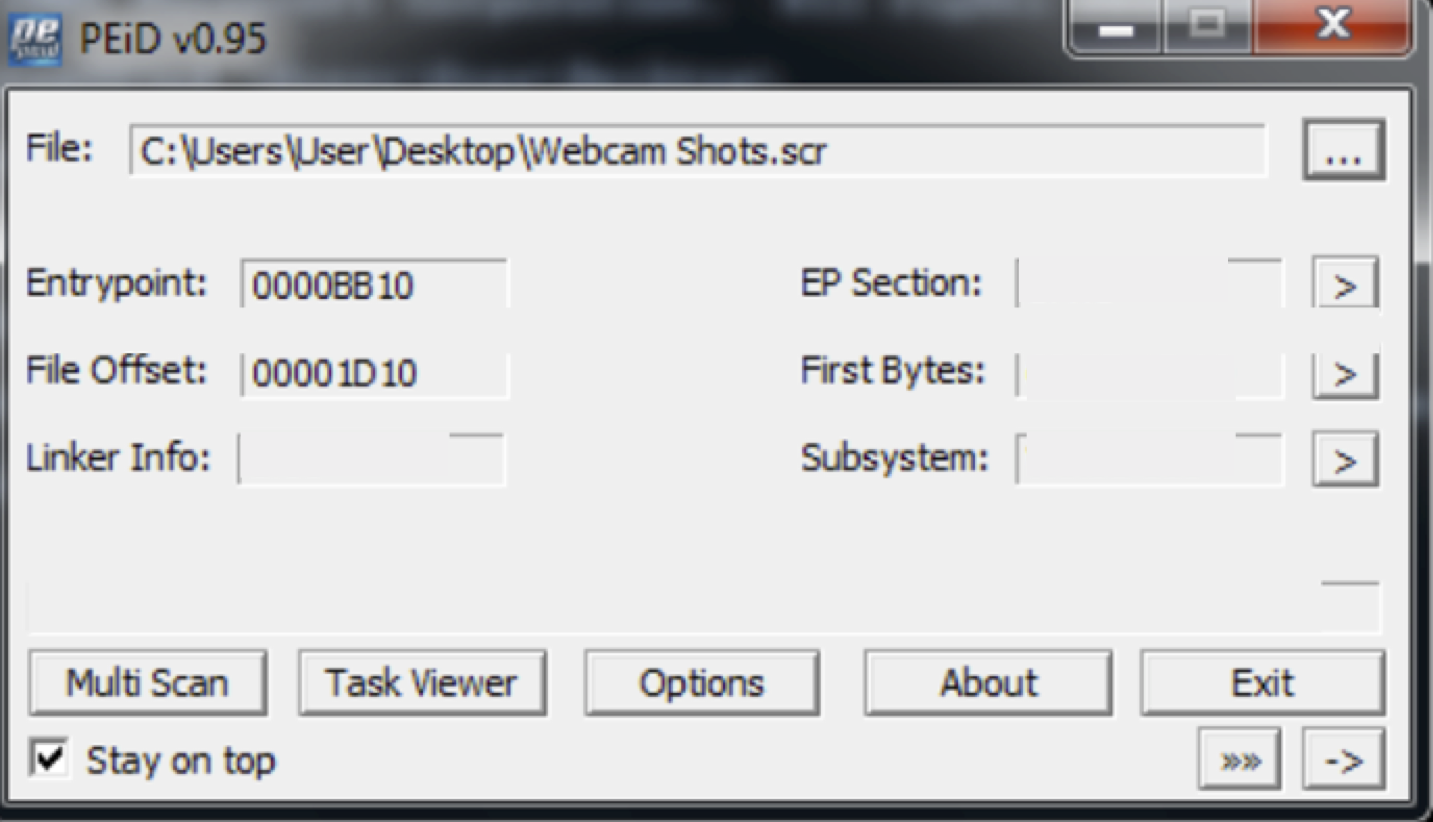
|  |
| --- |
| **Question 0x3** |
| What are the first two bytes in the binary? What do they indicate?  **Answer:** The first 2 bytes are “MZ” which indicates that this file is an MS DOS compatible file type. |

|  |
| --- |
| **Question 0x4** |
| What type of PE file is this (.exe, .dll, or .sys)? How can you tell? Hint: You need to use a combination of the following PE fields: FILE\_HEADER->Characteristics and OPTIONAL\_HEADER->Subsystem. PEInsider can give you more details about what the different flags mean by double clicking on the appropriate spot.  **Answer:** Looking at FILE\_HEADER->Characteristics clicking on the “characteristics” table it looks like the “file is executable” box is checked, and the “file is DLL” is not checked it also looked like nothing is checked within the “DLL characteristics” box. Blow shows the screen shots. While looking in the Optional Headers the ImageBase seems to align to an exe based on the assigned “400000” value.  OPTIONAL\_HEADER->Subsystem it looks to have the description “Windows GUI” |

|  |
| --- |
| **Question 0x5** |
| List the all PE **sections.**  **Answer:**  UPX0  UPX1  .rsrc |

|  |
| --- |
| **Question 0x6** |
| What can you (likely) infer from your answer to the previous question?  **Answer:**  It looks suspicious that the information above does not contain the sections typically seen within a section header table, such as: .data, .text, .bss, etc.. but instead the section headers above which indicate they have been packed with UPX. |

You should confirm your answer to question 0x6, as well as answer the next question with PEID:



|  |
| --- |
| **Question 0x7** |
| What packer version (name/version) does PEID identify?  **Answer:**  According to the PEID interface it looks like this malware was packed with UPX 0.89.6 – 1.02/1.05 – 2.90 by Markus and Laszlo |

|  |
| --- |
| **Question 0x8** |
| List two reasons why a piece of malware would be packed  **Answer:**  In general, it looks like malware is packed to obfuscate the malicious intent of the malware’s payload and makes the malware difficult to analyze. Another reason to pack the malware is to reduce the size of the payload on disk for easier distribution. |

As packed binaries hinder analysis, we will now unpack the malware. Luckily, binaries packed with UPX are trivial to unpack.

|  |
| --- |
| **Question 0x9** |
| How can the malware be (trivially) unpacked? (Hint: use the provided upx.exe utility).  **Answer:**  Upx.exe -d <filename> |

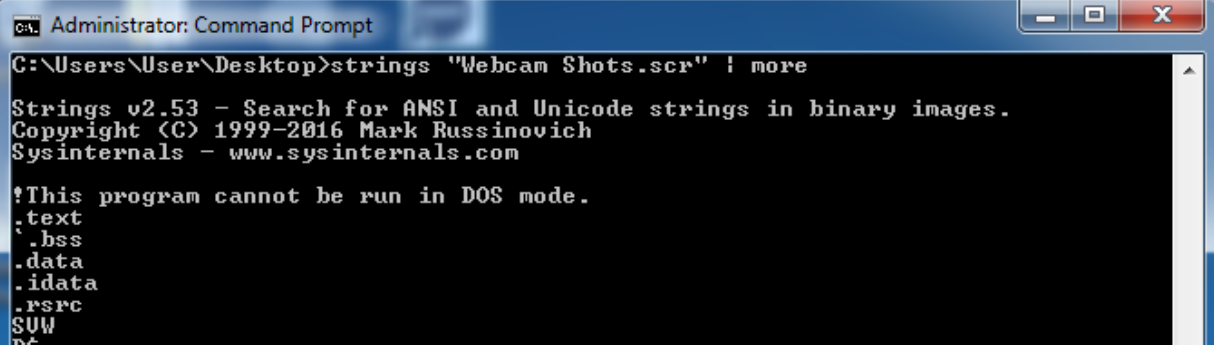
|  |
| --- |
| **Question 0xA** |
| Using the **unpacked** malware sample, complete the following. What do the new values indicate happened to the Webcam Shots.scr file? [Hint: Don’t overthink this question)  **Answer:**   |  |  | | --- | --- | | **Characteristic** | **Value** | | File Size | 27,680 kb | | File Hashes (MD5/SHA1) | md5 - 37a0497364dbb3c5e576322e8b243fc8  sha1 - cb4cbfbfea924a4d76a00b73059a392207a659a8 | |

Once a binary has been unpacked, triaging may continue. Open the **newly unpacked** malware in the PE-viewer utility (PEInsider).

|  |
| --- |
| **Question 0xB** |
| What dynamic-link libraries (dlls) does the malware import? What is the general purpose of each of these dlls?  **Answer:**  KERNEL32.DLL: Windows NT base API Client that handles memory management, basic i/o and interrupts for Windows machines [7]  ADVAPI32.DLL: Advances Windows 32 base API which provides advanced functionality such as registry and security calls [8]  CRTDLL.DLL: A Microsoft C Runtime module containing C library functions [9]  SHELL32.DLL: Microsoft Operating System module which contains windows shell API functions. [9]  USER32.DLL: Microsoft Operating System module which contains the Windows user component which can create, manage standard windows user interface elements. [9]  WININET.DLL: Microsoft Operating System module which runs an Internet extension for Win32 [9]  WS2\_32.DLL: Microsoft Operating System module which give you access to Windows socket functions [9] |

|  |
| --- |
| **Question 0xC** |
| List several imported functions, from any of dlls that the malware imports, that may lend insight into the malware’s actions?  **Answer:**  With the inclusion of the WS2\_32 and WININET DLLs it looks to me that the malware is trying to reach out to some location on the internet with the functions InternetOpenA, InternetOpenUrlA, and InternetReadFile which seem to open and read internet resources and perhaps try to achieve a remote code execution with the SHELL32 and CRTDLL DLL. Shell32 contains functions such as commandLineToArgvW, and shellExecuteEXW to perhaps open a shell and execute some C code. I would assume with the USER32 DLL it will suppress some Windows interfaces from opening up such as error boxes, a shell terminal, among other suppression by using the imported functions trackpopupmenu,showWindow, transilatemessage. |

ASCII or Unicode strings within a piece of malware can often reveal its intended actions or goals. Run the provided strings utility (strings.exe) on the unpacked malware in order to answer the next three questions. **You can run strings.exe once first and accept the licensing agreement or you can use the “/accepteula” flag.**



|  |
| --- |
| **Question 0xD** |
| List any string(s) that reference to other executable binaries.  **Answer:**  winservices.exe  ZoneLockup.exe  %s%s.exe |

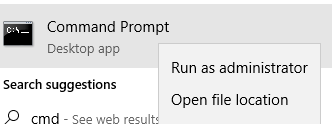
|  |
| --- |
| **Question 0xE** |
| List any string(s) that reference the registry.  **Answer:**  RegDeleteValueA RegCreateKeyExA RegCloseKey RegQueryValueExA RegSetValueExA |

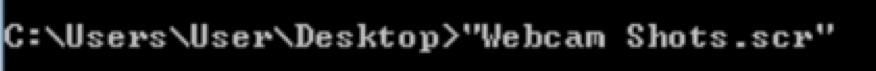
|  |
| --- |
| **Question 0xF** |
| List any string(s) related to networking/malware communications.  **Answer:**  Windows %s. uptime: %dd %dh %dm. cpu %iMHz. online: %dd %dh %dm. Current user: %s. IP:%s Hostname:%s. Processor %s. %PROCESSOR\_IDENTIFIER%  %s %s "[x.com](http://x.com/)" "x" :x  couldn't resolve host  @File downloaded to: %s size: %i Speed %.1fkb\sec Error InternetReadFile Error InternetOpenUrl Error InternetOpen Mozilla/4.0 (compatible)  Socks4 SOCKET\_ERROR Socks4 server waiting for connections could not open port  ioctlsocket inet\_addr  5.1.2700.0 (NT client.010817-1148) |

**Part 2 – Runtime Monitoring**

Once the initial triage stage has been completed, the malware may be executed under a closely monitored environment.

**IMPORTANT: Launch an ‘elevated’ command prompt. From this command prompt, execute the previously UNPACKED “Webcam Shots.scr” binary (From Question 0x09). This malware was designed to run with administrator privileges. Before running the malware, start the provided monitoring utilities AS ADMINISTRATOR: Process Monitor (procmon.exe) and Process Explorer (procexp.exe). These utilities will provide significant insight into the malware’s actions**





|  |
| --- |
| **Question 0x10** |
| Explain why malware may copy itself to a different directory and delete the original file.  **Answer:**  Attackers are aware that whenever something malicious happens on a user’s computer it’s a basic reaction to power down the machine to kill any unwanted processes so most malware will try to replicate itself into many temp file locations to avoid being detected and exterminated. |

Use Process Explorer and/or any other necessary utilities (e.g. hash.exe) to answer the following question.

|  |
| --- |
| **Question 0x11** |
| Were any new processes and/or files created? If so, fill out the following table.  **Answer:**   |  |  | | --- | --- | | **Characteristic** | **Value** | | File Name | ZoneLockup.exe | | File Size | 27,680 | | File Hashes (MD5/SHA1) | md5 - 37a0497364dbb3c5e576322e8b243fc8  sha1 - cb4cbfbfea924a4d76a00b73059a392207a659a8 | | Internal Description | Generic Host Process for Win32 Services | | Internal File Name | Winservices.exe | | Company | Microsoft Corporation | | Command Line\* | “C:\WINDOWS\SYSTEM32\ZoneLockup.exe” qwerC:\WINDOWS\SYSWOW64\ZONELOCKUP.EXE  Path: C:\WINDOWS\SysWOW64\ZoneLockup.exe |   \*Process Explorer can provide this information. **Be sure to run Process Explorer as administrator!** |

Notice how the name of this new process corresponds to the answer for question 0xD.

|  |
| --- |
| **Question 0x12** |
| Can you infer any relationships between the original malware binary and the newly created one?  **Answer:**  The newly created binary and the original malware binary seem to be the same in regards to having the same MD5, SHA1 and file size. |

Switch to Process Monitor (procmon.exe) in order to gain a more comprehensive insight into the malware’s actions. First, hit “CTRL + E” to stop capturing events. In order to reduce the 1000’s of events reported by Process Monitor, create a filter for both the name of original malware and the newly created process.



Once the output from Process Monitor has been filtered, answer the following questions.

|  |
| --- |
| **Question 0x13** |
| Can you find the event that illustrates the original malware (Webcam Shots.scr) creating the copy of itself? (Hint: look the first few references to the name of the new copy, ZoneLockup.exe).  **Answer:**  I believe I found with the images below the where the webcam shots copies over itself into zonelockup.exe with the first image. Notice the file write+ |

|  |
| --- |
| **Question 0x14** |
| What are the names of several Win32 API functions that are can be used to create a process?  **Answer:**  CreateProcess  CreateProcessAsUser  CreateProcessWithLogonW  AdjustTokenPrivileges  CreateRemoteThread  EnumProcesses  EnumProcessModules  GetModuleFilename  GetStartupInfo  ControlService  ProcessStart  ThreadCreate |

|  |
| --- |
| **Question 0x15** |
| Can you find the event that shows the original malware launching the copy of itself?  **Answer:**    The image below demonstrates webcam shots starting the zones process by calling the process create function. |

Most malware contains some sort of persistence mechanism to ensure that, on reboot, it will be automatically restarted.

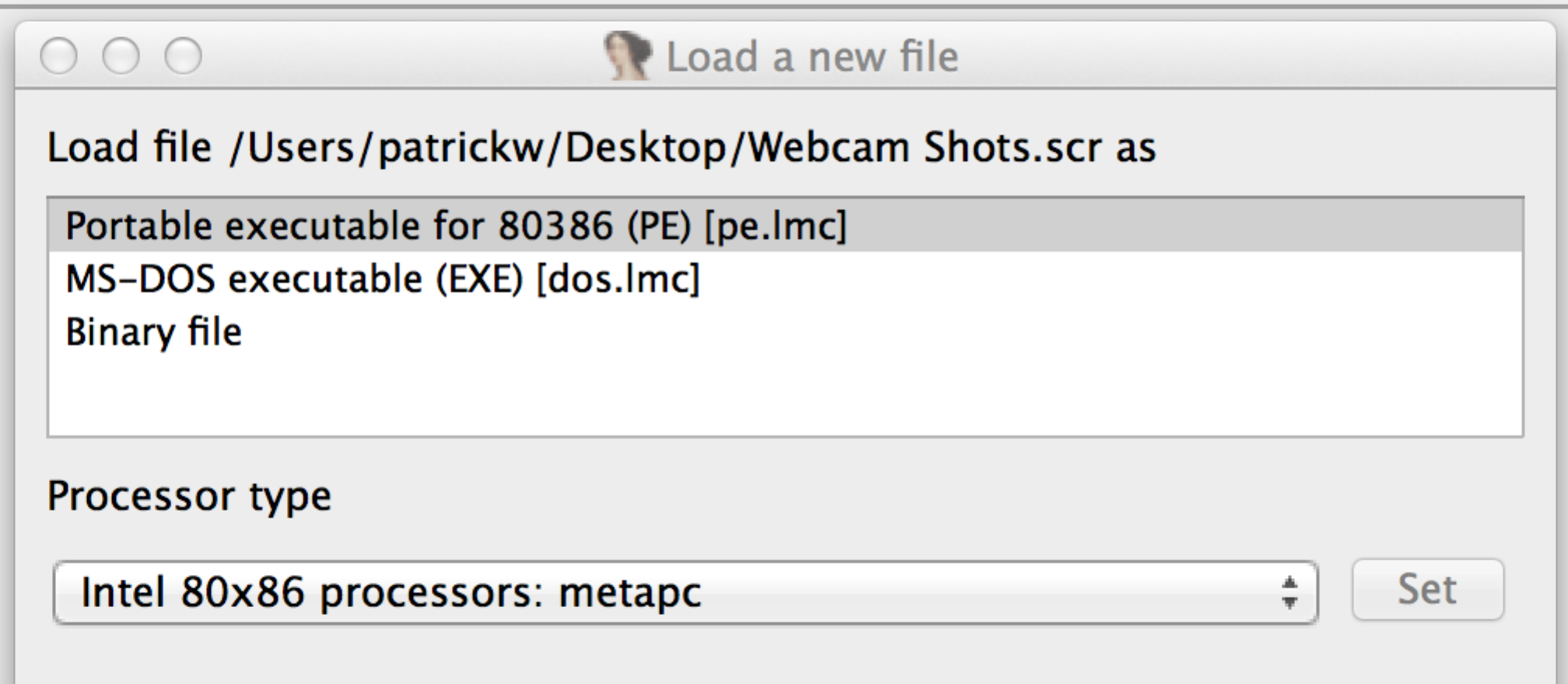
|  |
| --- |
| **Question 0x16** |
| Can you find the events that show the malware ensuring its persistence? (Hint: look for setting of the registry key that references the newly created copy of the malware**.)**  **Answer:** |

|  |
| --- |
| **Question 0x17** |
| Other persistence techniques/locations exist. Describe at least three other ways malware can persist on an infected system.  **Answer:** |

**Part 3 – Static Analysis with Ida Pro**

Once a malicious piece of software has been triaged and run in an instrumented environment, it is often quite helpful to disassemble it, using a tool such as IDA Pro. This static analysis can often confirm assumptions (made during the triage steps) or more generally, to gain a deeper insight into the malware.

Open the unpacked binary in IDA Pro in order to answer the following questions.



Recall that during the previous analysis steps, the malware was seen making a copy of itself (ZoneLockup.exe). This action will now be investigated. In IDA Pro, navigate to the function (press the ‘G’ key to bring up the ‘jump to address’ dialog) at offset 0x00402621 to answer the following question.

|  |
| --- |
| **Question 0x18** |
| From the disassembly, at the start of the function at offset 0x00402621, describe the steps (and API functions used) that malware performs in order to create the full path/name for its copy. Recreate these steps in C.    **Answer:** |

|  |
| --- |
| **Question 0x19** |
| Briefly describe what actions the malware is performing between addresses 0x004026E0 and 0x00402731.    **Answer:** |

|  |
| --- |
| **Question 0x1A** |
| At offset 0x00402775, the disassembly shows a mutex being created. What is the name of this mutex?    **Answer:** |

|  |
| --- |
| **Question 0x1B** |
| Once the CreateMutex() function has been invoked to create the mutex, the GetLastError() function is invoked (at offset 0x00402788). What is the error code it is checking for? What is the description of this error code? Why would the malware terminate itself is this error is encountered?    **Answer:** |

|  |
| --- |
| **Question 0x1C** |
| The malware at address 0x0040293F calls the InternetGetConnectedState() function. Using MSDN as a guide, describe this function and explain why the malware would invoke this function? **NOTE: If this function fails, you will need to enable your networking and restart analysis.**  **Answer:** |

Static disassembly can often reveal important values or constants. For example, at address 0x0040296E, the disassembly shows the malware invoking a function (via the call instruction). This function takes two parameters that are both hardcoded values.

|  |
| --- |
| **Question 0x1D** |
| What are the two hardcoded values passed to the function invoked at address 0x0040296E? What are the purposes of these parameters? (Hint see code at offset: 0x004012D9 and 0x004012F7 that references these values, as well as possibly, your answer to question 0xF).  **Answer:** |

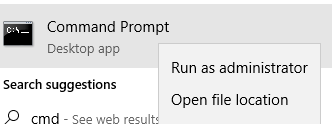
**Part 4 – Dynamic Analysis with Immunity Debugger (or Windbg)**

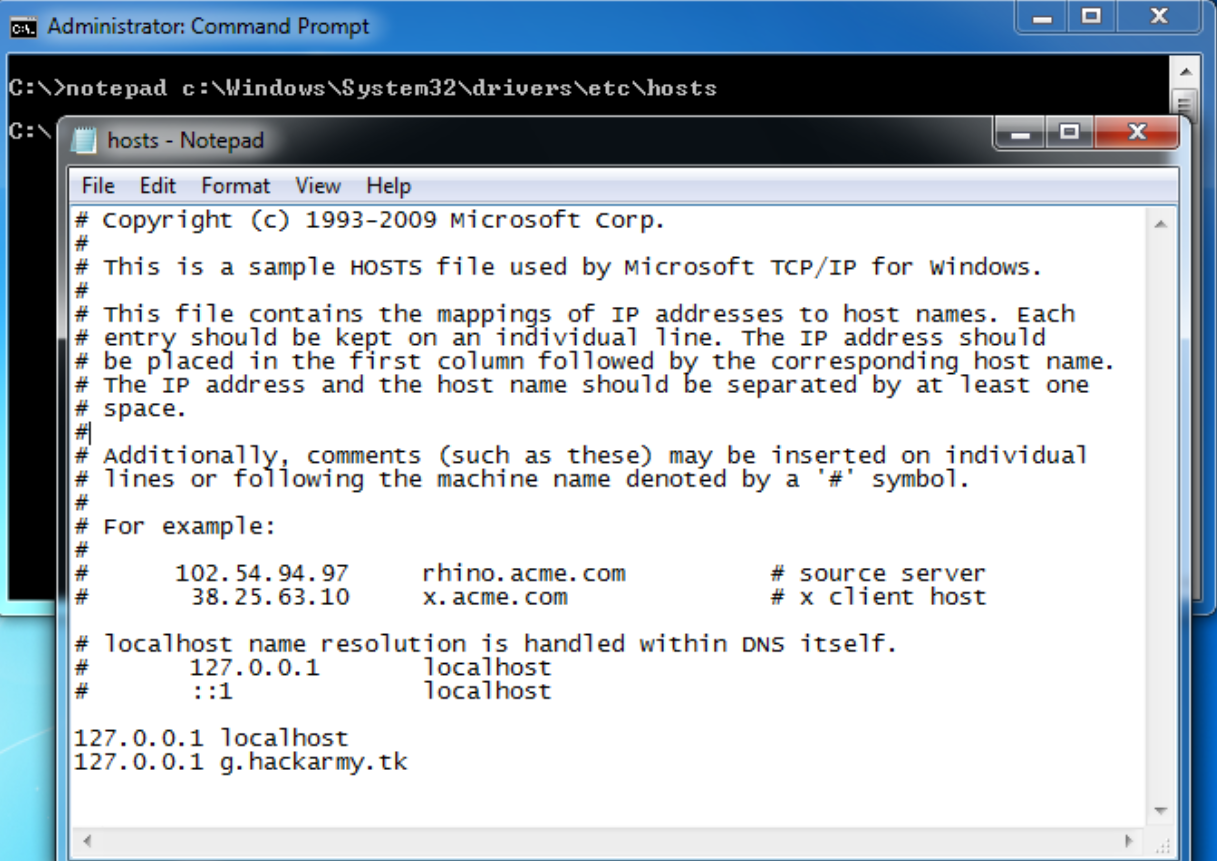
While static analysis can often provide significant insight into the inner workings of piece of malware, dynamic analysis often complements this analysis. Moreover, in many cases, dynamic analysis may prove to be a simpler and more powerful method of analysis.

In this section of the lab, dynamic analysis will be used to observe the malware’s network communications. Specifically, the malware will be executed within the debugger, allowing its inner most secrets to be revealed. This in turn, will provide the information needed to remotely command and control the malware, thus providing complete control over the infected system.

Before the malware is executed within the debugger (in order to observe its network communications), several prerequisites must be performed. First, a new entry (127.0.0.1 -> g.hackarmy.tk) should be added to the host file (C:\WINDOWS\system32\drivers\etc\hosts), to ensure that the malware can resolve an IP address of its hardcoded URL, g.hackarmy.tk. Once the host file has been edited and saved, and request for the URL ‘g.hackarmy.tk’ will resolve to 127.0.0.1, the localhost. (This can be confirmed via the ping command; ‘ping g.hackarmy.tk’ )

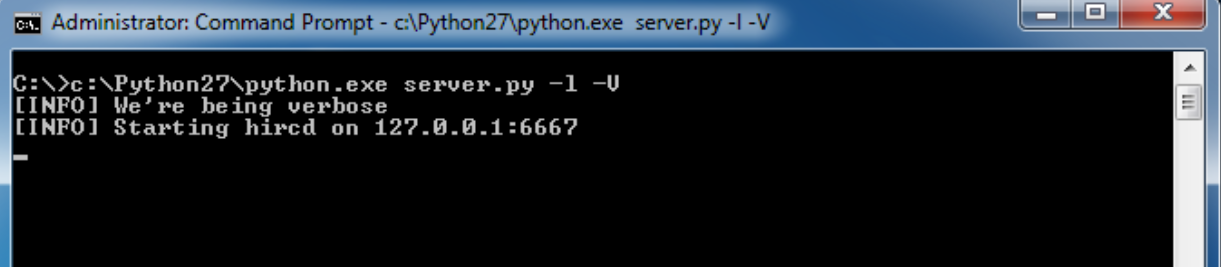
**IMPORTANT: Be sure to start an ‘elevated’ command prompt first. Otherwise, you won’t be able to save this file to the proper location.**





Once the host file has been modified, the provided command and control server (server.py) should be started. This server will listen on the localhost (127.0.0.1), port 6667, for any network connections. In order to start this server, ensure that python is installed (run the provided python installer, python.msi, if necessary) then simply execute the following commands from a command prompt (cmd.exe): \path\to\python \path\to\server.py -l –V

This should produce the following output:



Finally, kill any running instances of that malware (as question 0x1B illustrated, the malware first ensures that no other instances of itself are currently executing).

**Important:**

**On 64-bit systems, the "ZoneLockup.exe" gets copied to "c:\windows\syswow64" instead. This is due to the fact that the malware is a 32-bit executable running on a 64-bit Windows system. This is referred to as WOW64 [Windows on Windows64]. To continue dynamic analysis, you MUST do the following:**

1. **KILL ALL ZONELOCKUP.EXE TASKS AND MOVE IT TO SYSTEM32:**

**From an *elevated* command prompt, execute the following commands:**

**taskkill /f /im "zonelockup.exe"**

**copy c:\windows\syswow64\zonelockup.exe c:\windows\system32\zonelockup.exe**

**del c:\windows\syswow64\zonelockup.exe**

1. **Start Immunity Debugger as *administrator.* If you do not start it as administrator, the malware may not behave properly.**

**On 64-bit Windows, you will need to open C:\windows\sysnative\ZoneLockup.exe instead from the Immunity Open Dialog box (see below). You may get an error popup (see below), but click OK to continue past that error.** Once all the prerequisites have been met, open the persistent copy of the malware (C:\windows\sysnative\ZoneLockup.exe) into the debugger:





Once the persistent copy of the malware has been opened in the debugger, set a breakpoint at 0x004012F7 (call gethostbyname), then start execution (Debug->Run). When the breakpoint has been hit, answer the following question. **NOTE: If the address 0x004012F7 is not valid, go back to step 0x10 and make sure you ran the UNPACKED version of “Webcam Shots.scr”.**

**Note: If you are not hitting this breakpoint, it is likely InternetGetConnectedState() is failing. Please ensure your computer has Internet connectivity. Alternatively, you can break at this InternetGetConnectedState() and change the return value to TRUE (i.e. change EAX to 1).**

|  |
| --- |
| **Question 0x1E** |
| What does the gethostbyname function do? What does it return?  **Answer:** |

|  |
| --- |
| **Question 0x1F** |
| What is the parameter (host name) that is being passed to the gethostbyname function? (Hint: this should match your answer to question 0x1D…see how static and dynamic analysis are complementary?)  **Answer:** |

The malware should be stopped at your breakpoint (address 0x004012F7). Step over the gethostbyname function.

|  |
| --- |
| **Question 0x20** |
| In general, once a function has executed, what does the EAX register contain?  **Answer:** |

Your answer to question 0x1E specified what the gethostbyname function returns. Since you stepped over the call to this function (at address 0x004012F7), you can now examine this structure (which will be pointed to by the EAX register) to determine the IP address that corresponds to the host name, “g.hackarmy.tk.”

|  |
| --- |
| **Question 0x21** |
| What is the IP address that corresponds to the host name (g.hackarmy.tk) ?  Hint(s): First go to the address specified in the EAX register, this is the hostent struct returned by the gethostbyname function. Once at this structure, go to offset 0xC; this is the pointer to host array (in case the host entry has more than one IP associated with it..in our case it should only have the one IP). Follow this pointer. At this new location, there is another pointer. Follow this pointer to identify the IP address as a network-byte ordered DWORD.  To help out, below is C-code equivalent of the above description, that loops through all potential IPs (not just the one).  remoteHost = gethostbyaddr((char \*) &addr, 4, AF\_INET);  if (remoteHost->h\_addrtype == AF\_INET) {  while (remoteHost->h\_addr\_list[i] != 0) {  addr.s\_addr = \*(u\_long \*) remoteHost->h\_addr\_list[i++];  }  }  **Answer:** |

Delete this breakpoint (at address 0x004012F7), and set new one at a call to the connect() function (address 0x0040132C). This is where the malware will attempt to connect to our local server (server.py). Once the breakpoint has been set, continue execution (Debug->Run). The debugger should stop at this new breakpoint. Single step over this function.

|  |
| --- |
| **Question 0x22** |
| What appears in the console window of our local server (server.py)?  **Answer:** |

Delete this breakpoint, and set a new on at a call to the send() function (address 0x00402B01). As a connection has been established with the server, the malware will invoke this function to send data to the server. By placing a breakpoint at this location, then continuing execution (Debug->Run), we will be able to examine this data.

|  |
| --- |
| **Question 0x23** |
| Once the breakpoint at the call to the send() function (address 0x00402B01) as been hit, what data is about to be sent to the server?  **Answer:** |

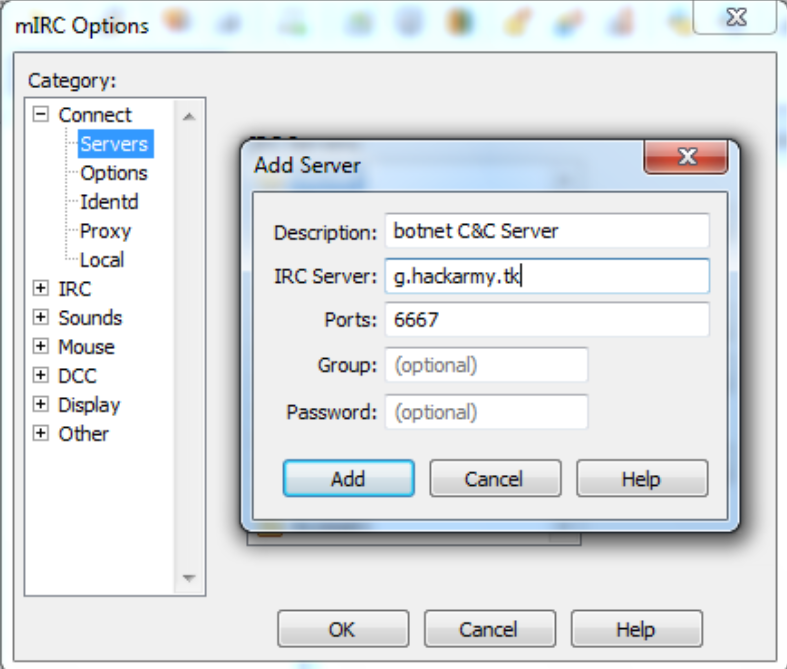
Again, remove this breakpoint, and set another one at address 0x00402DE5. This is another location in the malware’s code, where data is being sent to the server. Once the breakpoint has been set, allow execution to continue. When the debugger breaks (at address 0x00402DE5), again examine the data the malware is about to send to the server.

|  |
| --- |
| **Question 0x24** |
| Once the breakpoint at the call to the send() function (address 0x00402DE5) as been hit, what data is about to be sent to the server?  **Answer:** |

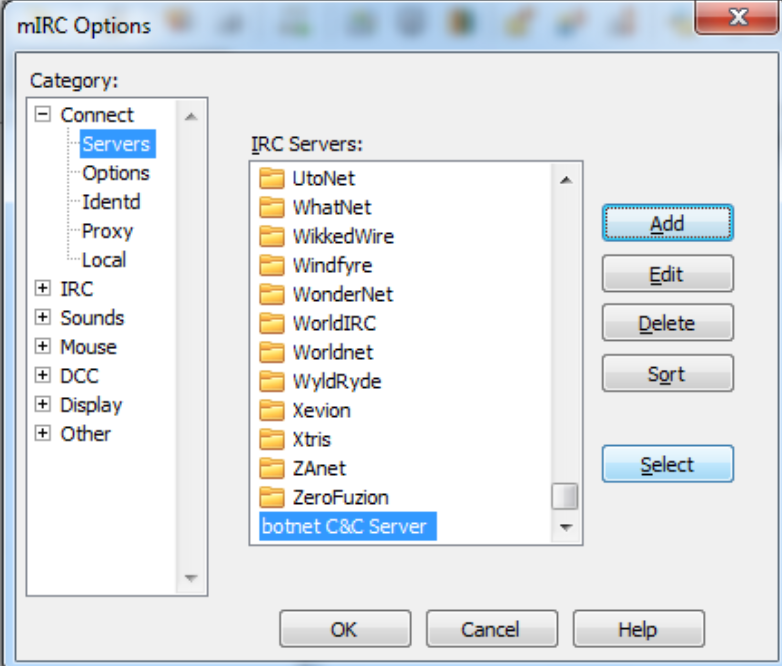
At this point, it should be fairly clear the malware sample is an IRC bot, that’s currently attempting to connect out to an IRC server (located at g.hackarmy.tk) for remote tasking. IRC bots communicate via the IRC protocol, to a server which is listening on the standard IRC port (6667). Your answer to last two questions should have provided you with the randomly generated nick-name of the bot (that will appear on the IRC server), as well as the name of the IRC channel (##g##).

Since the malware has now both connected and registered with our local (IRC) server, (server.py), we should be join the same IRC server/channel and see it there. To accomplish this, first delete the most recent (and all other breakpoints), and allow the malware to continue execution. Following this, setup an IRC client to connect to our local server and join the same IRC channel as the malware (described below).

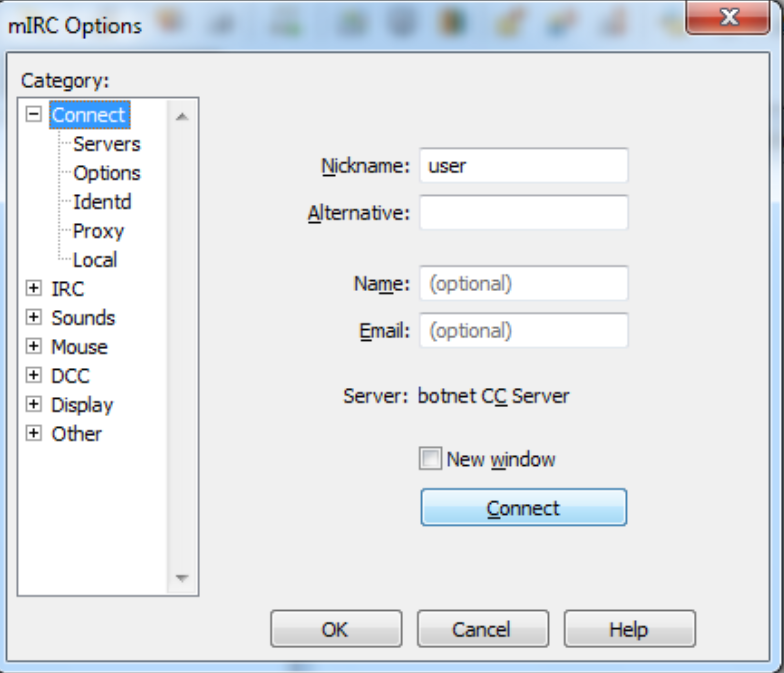
To setup an IRC client, run the provided mIRC installer. Once installed, launch mIRC. First, add the following information about the local server which we’ll be connecting to.



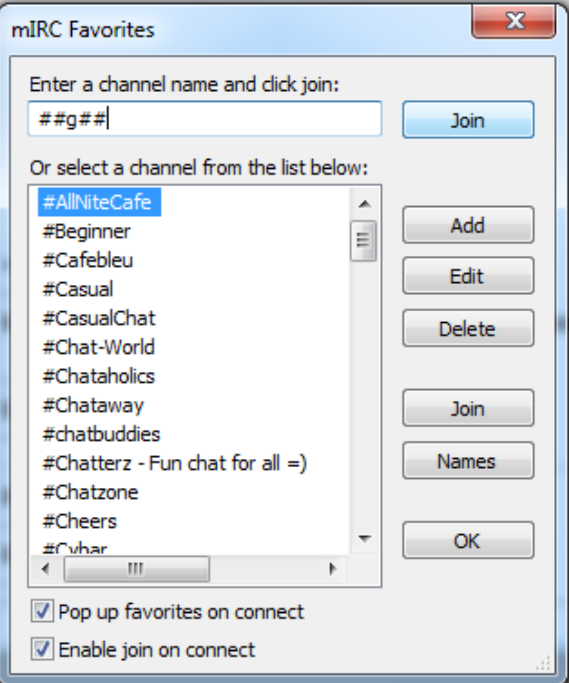
Following this, select this as the server to connect to:



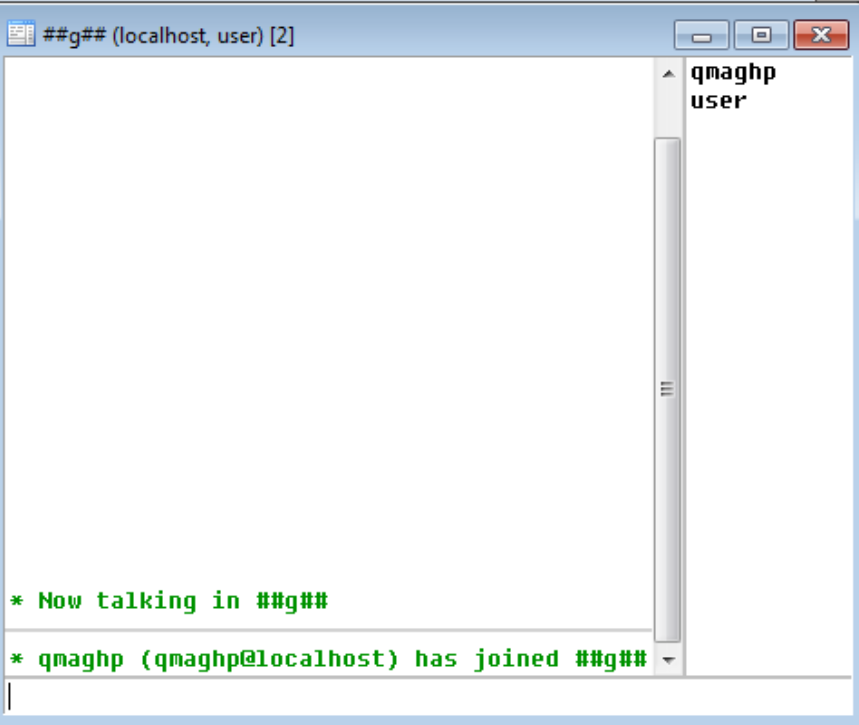
Then, specify a nickname, and click ‘Connect’ to connect to the local IRC server:



Finally, instruct the IRC server which channel you want to connect to (##g##):



You should now be connected to the IRC server and have joined the ##g## channel. The other ‘user’ in the channel should be the (randomly named) malware/bot:



In order to directly communicate with the malware via the IRC client, we must first authenticate to it as the ‘bot master.’ In order to do this, we’ll have to uncover the authentication password. Return to the debugger and set a breakpoint on a call to the strcmp() function at address 0x00402F4C. Back in the IRC client, select the bot (double-click the name) in order to ‘private message’ it.

In the private message window type the following: !password. Then press enter to send it to the malware. The debugger should immediately break at the strcmp() function. Here, you should see the malware comparing the password you entered, along with the true authentication password.

|  |
| --- |
| **Question 0x25** |
| What is the **correct password** that the malware is expecting?  **Answer:** |

Remove this breakpoint (and any other remaining breakpoints), and continue execution. Return to the IRC client and enter this password (preceded by a !) in the ‘private message’ window. Press enter, to send it. Congratulations, you now have authenticated as the ‘bot master’ thus possessing complete control over the bot!

|  |
| --- |
| **Question 0x26** |
| Locate **all of the commands** this malware supports or claims to support and detail their actions in the chart below. |

The malware supports various commands that allow an authenticated ‘bot master’ complete command and control over the infected target computer. Using any method (static analysis, dynamic debugging, etc.), uncover these commands and explain how they work. [Note not all commands may be implemented, but you still must identify all potential commands]

|  |  |
| --- | --- |
| **Command** | **Action** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

References

<https://resources.infosecinstitute.com/topic/2-malware-researchers-handbook-demystifying-pe-file/> [1]

[https://docs.microsoft.com/en-us/windows/win32/debug/pe-format [2](https://docs.microsoft.com/en-us/windows/win32/debug/pe-format%20%20%5b2)]

<https://labs.detectify.com/2016/04/12/using-reverse-engineering-techniques-to-see-how-a-common-malware-packer-works/> [3]

<https://www.oreilly.com/library/view/practical-malware-analysis/9781593272906/ch02s04.html#:~:text=Malware%20writers%20often%20use%20packing,compressed%20and%20cannot%20be%20analyzed>. [4]

[https://malware.news/t/the-basics-of-packed-malware-manually-unpacking-upx-executables/35961 [5](https://malware.news/t/the-basics-of-packed-malware-manually-unpacking-upx-executables/35961%20%5b5)]

<https://www.dll-files.com/search/?q=KERNEL32.DLL> [6]

<https://www.webopedia.com/TERM/K/kernel32_dll.html> [7]

[https://www.file.net/process/advapi32.dll.html [8](https://www.file.net/process/advapi32.dll.html%20%5b8)]

<https://www.fileinspect.com/fileinfo/crtdll-dll/> [9]

[https://medium.com/mrx-007/basic-static-analysis-of-malware-and-common-dll-ef9455d49968 [10](https://medium.com/mrx-007/basic-static-analysis-of-malware-and-common-dll-ef9455d49968%20%5b10)]

<https://resources.infosecinstitute.com/topic/windows-functions-in-malware-analysis-cheat-sheet-part-1/> [11]