Lab5

**Issued**: July 2nd, 2021

**Due**: July 8th, 2021 11:59PM

Name: Kamalesh Ram Chandran Govindaraj

**Overview**:

In class we reviewed code injection techniques for Windows. In this lab we will use a piece of recent malware, first detected in May of 2020, that implements remote DLL injection. The file hash is:

f2ccaee6b295609f443f12272df3ef862d89f496046316bd9bd7823110a58645

And here is a link to its VirusTotal [description](https://www.virustotal.com/gui/file/f2ccaee6b295609f443f12272df3ef862d89f496046316bd9bd7823110a58645/detection). Here is a more detailed description from Microsoft. <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor%3aWin32%2fShtormec.A>

Open up a clean FlareVM, ensure it is set to HostOnly mode, and then copy the malware found within to the Flare desktop. The password for the file is “infected”

[lab5.zip](https://piazza.com/class_profile/get_resource/koko92l4m4u7ep/kqlowmj16je4n3)

**Step One**

First, let’s set breakpoints. These are breakpoints where you will need to stop and answer questions and in other places, stop to change the logic of the control flow. Go to each of the memory locations below and set a breakpoint. You will be able to rerun the analysis many times as this malware does not self-delete at the end. Be sure to save your solution after setting the breakpoints initially. That way, if Windows Defender should kick on and delete it you will still have the breakpoints in the IDA solution.

|  |  |
| --- | --- |
| **Memory Address** | **Reason for Stop** |
| 00401CCB | CONTROL FLOW CHANGE |
| 00401CFC | Question |
| 00401D5D | Question |
| 00401DA2 | CONTROL FLOW CHANGE |
| 00401E51 | CONTROL FLOW CHANGE |
| 00401EAF | Question |
| 004010C0 | Question |
| 0040116B | Question |
| 004011DB | CONTROL FLOW CHANGE |
| 004011E9 | Question |
| 0040122E | Point of clarification |

Each question following is worth 5 points. Run IDA Educational as admin and load in the malware sample. Be sure to enable **View > Toolbars > Debugger commands.** Then, run the malware and answer the question below.

**Part 1**

**Location 00401CCB** - We have skipped through the preamble of the start function and made it here. There is a MessageBox coming up! We need to **carefully** step forward one step at a time, clicking ‘Ok’ for the message box ‘123’ that pops up. It may be minimized in the task bar. Click until you get to memory location 00401CD7. Change the **ZF** flag to 1 as we want the control flow to go left. You can click the resume button to go to the next break point.

**Location 00401CFC** - What reg key is about to be opened and what will it let it do?

<Answer Here>

You can click the resume button to go to the next break point.

**Location 00401D5D** - What is the malware likely constructing with the string it is building at this point? That is, this string will be used for something, what is it? Two items. One, the ‘operator+’ appends string information. Second, the link to the article at top of this assignment should be helpful in determining why the string is constructed.

<Answer Here>

You can click the resume button to go to the next break point.

**Location 00401DA2** - Change the **ZF** register to be 0.

You can click the resume button to go to the next break point.

**Location 00401E51** - Change **ZF** register to be 0.

You can click the resume button to go to the next break point.

**Location 00401EAF** - The location of a DLL is about to be shown when it is placed on the stack. What is the precise location it expects to find the DLL and does it exist? Remember what we went over in lecture on July 1st and how to examine values pushed on the stack by following the location in the hex dump.

<Answer Here>

You can click the resume button to go to the next break point.

**Location 004010C0** - Go to this point. Open up in graph view and pan down. What three applications is it interested in, likely targeting these for process injection?

|  |  |
| --- | --- |
| App1 | <Answer Here> |
| App2 |  |
| App3 |  |

You can click the resume button to go to the next break point.

**Location 0040116B** - What app gets opened by the malware after this call? Here, you will need to ***single step*** through all the push arguments and determine which one holds the path to the executable that will be launched (and later process injected into) by CreateProcessA at memory location 0040119D. You will need to examine arguments you believe may be the executable by following the hex dump for stack arguments as they are pushed.

<Answer Here>

Keep stepping one step at a time. Do this until a window opens - which will happen after the CreateProcessA call. What app opens? (Do not close this Window - just minimize it!)

<Answer Here>

**Location 004011DB** - Make certain the zf flag is 0. Just double check this one - it probably is 0 but just be sure.

You can click the resume button to go to the next break point.

**Location 004011E9** - A call to WriteProcessMemory is about to happen. What string will be written into memory? Again, this will be another exercise where you need to step through the items pushed on the stack until you find your target and after it is pushed on the stack, dump the hex content to view it.

<Answer Here>

You can click the resume button to go to the next break point.

**Location 0040122E** - At this point you will see CreateRemoteThread in the process started. For some reason (I did not have the patience to track down why) the process ID provided where to create the RemoteThread does not match the process ID for the application opened by the malware. Had it all worked out, I would have had you open the application in ProcessHacker, look at the strings, and verify the string we wrote to the memory was there.

At this point you may stop debugging. One last question below.

Take a look at **Location 00401F48**. If you look at it in graph view and pan left and right, you see some interesting text being posted to message boxes. Write a couple sentence description of what you believe the app is doing at this point.

<Answer Here>