Lab 5: Windows Internals

Jonathan Harijanto

February 15, 2017

CS 373: Defense Against the Dark Arts - Winter 2017

Abstract

The purpose of this blog is to describe my observation when I dive deep into the Windows kernel. This blog will cover the testing methodology and the knowledge that was gain from completing the lab.

I. Blog

A. What you looked at:

The topic for this week is Windows memory manipulation. Aditya (the instructor) hoped that we could understand the concepts of stealth through memory manipulation and the basic knowledge of Windows kernel debug technique. There were supposedly five (5) different labs; however, he decided to cancel the fifth ones because he was running out of time.

In these four labs, I looked at various malware called Agony, Zbot, and TDSS. The first three lab reports are about Agony & Zbot analysis, and the fourth one is about TDSS. The tools that I used for these experiments were:

- Tuluka – software to detect and remove rootkits.
- Cuckoo – software to analyze any malicious file inside a sandbox.
- LiveKD – a command tool that runs Kd and WinDbg.
- ProcessHacker – software to monitor system resources and detect malware.
- WinDbg – software to debug Windows kernel.
- FakeNet – a program to monitor network traffic.

B. How you looked at it & What you found:

1) Lab 1:

The first lab was more like an introduction to the Agony malware. Thus, to obtain more information about this new malware, I need to run it on Cuckoo. The steps were similar to the previous lab; I had to extract a zip file called Agony.zip from WindowsInternals folder with infected as the password, rename the file into bad, open FakeNet and run the command analyzer in Command Prompt. The results were surprising, I received 9 different bin files and a file called `e2r355.ren`, while the live demo video had only 3 bin files. Fortunately, these differences didn't affect this lab at all.

I read the report produced by Cuckoo and concluded that this malware is a trojan. The next step was to run another software called Tuluka. I clicked on the `SST` tab (near menu bar) and sorted the list by Suspicion. From the results, I found three suspicious functions which are: `NtEnumerateValueKey`, `NtQueryDirectoryFile`, `NtQuerySystemInformation`. Using this software, I could see that these functions were hooked because each current address wasn't the pointing to the original address.

To investigate this malware further, I launched LiveKD to explore inside kernel memory. I typed `u <current address>` for each of the functions and viewed it from the stack perspective. I could see that Agony pushed a new address that was referencing to `wininit+` instead of `nt!`'s original address.

Now that I knew that the memory addresses were being manipulated, I had to patch them back. Fortunately, Tuluka has the ability to do so! All I need to do was right-clicked on these infected functions such as `NtQueryDirectoryFile`, and chose Restore Service. Essentially, this functionality cut the hook and restored the pointer to its original address.

2) Lab 2:

In this second lab, I had to deal with a new malware called Zbot. This malware is interesting because after it was being extracted from the zip, renamed into an executable file and double-clicked, it decided to delete itself from the Desktop. Next, Aditya told us that we had to open Notepad and ProcessHacker, then check for Notepad memory property using Process Hacker. The way to check the property of a program in ProcessHacker is to right-click and choose Properties on the designated running programs.

The main reason why I had to open Notepad because some malware/ rootkit likes to hook Notepad's address and capture keystrokes. Anyhow, in the Properties table, I sorted the list based on file permissions and observed several processes only named as Private. Among these Private processes, two of them has Read Write Execute permissions. I was suspicious these two because, based on the list, only a process with a name (example: Notepad.exe) that could have Read Execute permissions. In the end, the instructor finally revealed that Zbot is a malware that performed process injection. Essentially, Zbot automatically spawned a process called Private when I opened Notepad. These Private processes were the ones that are going to capture everything I typed and send it to a server. In conclusion, Aditya warned us that a no name (a.k.a Private) process means that there exists a sneaky or malicious program inside our machine.

3) Lab 3 (Answer for TopHat - Agony additional questions included here!):

This lab is about kernel debugging; I'm required to run two different Windows machines (debuggee and debugger) for this debug process. In the debuggee machine, I had to execute Agony malware inside analyzer (Cuckoo) and opened Tuluka to see the hooking address. Next, in the debugger machine, I ran winDbg to use the Kernel Debug (press Ctrl + K) feature. The instructor, Aditya, told us to start debugging from NtEnumerateValueKey, then continued until NtQuerySystemInformation.

The first step was to set a breakpoint at the `NtEnumerateValueKey` hook address using a command `bp 9d108480`. Then, I entered g to let the process run and hit the breakpoint. When it reached the breakpoint, I went to WinDbg menu bar, selected View and chose Call Stack to see the changes in the stack. Apparently, I saw an additional entry `wininit+0x1480` being pushed into the stack. Next, I went to menu bar again, selected View and chose Disassembly to see the entire instruction code.

The question asked from TopHat was to find the offset of winnit instructions from the beginning of the function boundary. To do that, I had to step into the assembly instruction and locate the difference between initial and final hooked address. For `NtEnumerateValueKey` function, the initial addresses hooked by wininit was `9d108480` and the last addresses hooked by wininit was `9d1084d4`. Thus the offset was `9d1084d4 - 9d108480 = 84` (decimal).

I applied the same technique to find the offset instruction for the other two `Nt` functions. For `NtQueryDirectoryFile`, I set the breakpoint to `9d108050` because that's the initial hooked address shown by Tuluka. I kept on stepping into the assembly instruction and found out that the final address hooked by wininit was `9d108086`. Hence, the offset was `9d108086 - 9d108050 = 54` (decimal). Lastly, for `NtQuerySystemInformation`, the breakpoint was set to its intial hooked address which was `9d107f00`. The last addresses shown by WinDbg was `9d107f1a`. Thus, the offset for this API was `9d107f1a - 9d107f00 = 26` (decimal).

4) Lab 4:

This lab is a special one because Aditya didn't finish the lab. Thus, I decided to finish it on my own with the help of internet. For this lab, I had to analyze a new malware called TDSS. The only tools required for this lab section was WinDbg and Tuluka. First, I entered the command `!devstack \device\harddisk0\dr0`, in WinDbg, to what are the drivers that are serving disk (device stack). Before the machine got infected, there were three different drivers: `partmgr, Disk, LSI_SAS`.

The next step was to extract the TDSS from zip and execute the malware. Surprisingly, when I re-entered the `devstack` command, there were only two drivers left on the disk. I noticed that LSI_SAS was not a driver object anymore because TDSS infected it. According to the internet, the reason that could happen because the pointer that pointing to LSI_SAS driver was now pointing to a fake driver object created by the malware.

To investigate this issue, I need to test the non-infected LSI_SAS beforehand. I entered `dt _device_object 85402f08` command, where `85402f08` was the original address for LSI_SAS before infected, to examine the device object data type. The result showed that the `DriverObject` is pointing to `85429e90`. Thus, I typed another command `dt _driver_object 85429e90` to analyze that specific driver object data type further. As I expected, the result stated that the `DriverStartIO` is pointing to null, which is correct because it showed that non-infected LSI_SAS is not pointing to a fake driver.

Things got interesting when I tested the infected LSI_SAS. I entered `dt _device_object` command with the address given from the malware and, in the end, I could see that `DriverStartIO` was pointing to some addresses instead of null. Hence, this result concluded that TDSS hooked the LSI_SAS driver address to a fake driver so it could perform a malicious behavior inside the machine.

5) Programming Assignment:

My program was written in Python. This single python file is able to perform 5 different tasks assigned from TopHat. Also, I used an external library called `psutil` for retrieving process information and system utilization. Thus, in order to run my program smoothly, it is required to download and install `psutil` to your testing machine.

II. Conclusion

This week lab taught me a lot about Windows kernel internals. I didn't know that we could dive very deep into the kernel if using the right tools/ programs. Furthermore, I also learned using new software such as Tuluka, ProcessHacker, LiveKD and new features like Kernel Debug mode in WinDbg. Last but not least, I was exposed to new security concepts like the hook.