Kernel Mode Rootkit

CSC 471 - 02

Tyler Prehl

4/8/2022

Introduction

The purpose of this lab is to gain insight into using rootkits as malware, but more specifically SSDT hooking. Throughout the lab, I will analyze the effects of SSDT hooking malware to better understand how it operates and how such malware can be identified after a machine is infected.

Analysis and Results

Q1) What is an SSDT?

An SSDT is a System Service Dispatch Table, and it is the source of SSDT hooking. The SSDT contains pointers to kernel functions, which are used upon system call invocations. Malware that uses SSDT hooking can be extremely effective.

Q2) Please use WinDBG and type the following command: “dds KiServiceTable L 12a”. Please take a screenshot of the output result. What’s the meaning of each column?

The total output is the SSDT table. The first column represents the row’s location in memory within the SSDT table (how the row is originally accessed). The second column represents the pointed location of the kernel function of that row, and the last column reveals the names of the functions.

```
kd> dds KiServiceTable L 12a
804e2620  8057a521 nt!NtAcceptConnectPort
804e2624  805760c9 nt!NtAccessCheck
804e2628  8058cabf nt!NtAccessCheckAndAuditAlarm
804e262c  8058e3f7 nt!NtAccessCheckByType
804e2630  8058d7c0 nt!NtAccessCheckByTypeAndAuditAlarm
804e2634  80638f82 nt!NtAccessCheckByTypeResultList
804e2638  8063b113 nt!NtAccessCheckByTypeResultListAndAuditAlarm
804e263c  8063b15c nt!NtAccessCheckByTypeResultListAndAuditAlarmByHandle
804e2640  805745ef nt!NtAddAtom
804e2644  80649b8b nt!NtSetBootEntryOrder
804e2648  8063873d nt!NtAdjustGroupsToken
804e264c  8058cdde nt!NtAdjustPrivilegesToken
804e2650  80630120 nt!NtAlertResumeThread
804e2654  80577310 nt!NtAlertThread
804e2658  80591595 nt!NtAllocateLocallyUniqueId
```

Q3) Comparing the two outputs (before hook and after hook), can you tell which functions have been hooked?

Yes, you can tell which functions have been hooked by the “SSDTHook” identifier. More importantly, the actual location of the function (the second column) has changed. On the left in the image is the notepad text file of the output after running the loader.exe malware file with the hooked function, and on the right is the notepad text file of the output before the malware hooked the !NtTerminateProcess function.

Q4) Can you take a guess of what is this malware trying to do (based on the hooked functions)?

Based on the hooks denoted in WinDBG - NtTerminateProcess, NtLoadDriver, NtOpenProcess, and NtDeleteValueKey - this malware is trying to hide the fact that it is opening/terminating processes and deleting values, while the NtLoadDriver was used to load and run the SSDT hook driver.

```
NtTerminateProcess address: 0x80585851
NtLoadDriver address: 0x805a29bd
NtOpenProcess address: 0x80574bc1
NtDeleteValueKey address: 0x805936da
NtTerminateProcess hooked.
NtLoadDriver hooked.
NtOpenProcess hooked.
NtDeleteValueKey hooked.
SSDT hook driver loaded.
```

Discussion and Conclusion

From successful completion of this lab, I have utilized WinDBG to analyze how an SSDT hooking malware executable hooked a WindowsXP system by debugging the Windows kernel, and also how to identify that the system was compromised. Understanding the fundamentals of rootkits, hooking, and debugging Windows kernels is essential for understanding, identifying, and purifying malware that masks its existence and accesses areas of a machine that are otherwise off-limits.