West Chester University

CSC 471

Dr. Si Chen

Spring 2024 Lab 4

Submitted by

Sean Berlin

4/23/24

1. Introduction:

The purpose of this lab is to understand Kernel Mode Rootkits and use WinDBG with VirtualBox for debugging Windows kernels. Setting up kernel-mode debugging on a virtual machine manually involves employing a Virtual COM Port for communication. By delving into system-level operations, we aim to detect and analyze rootkit behavior within the Windows OS. Through the experiment setup, we establish a debug environment to observe the Windows XP virtual machine kernel under WinDBG scrutiny. This investigation deepens our understanding of rootkit operations and equips us with skills to mitigate such security risks.

2. **Analysis and Results:**

Q1. What is a SSDT?

SSDT stands for System Service Descriptor Table. It is a core component of the Windows kernel. It maintains a list of system service call addresses, providing essential functions for user-mode applications. Malware often targets the SSDT to intercept system calls, allowing it to manipulate system behavior and evade detection

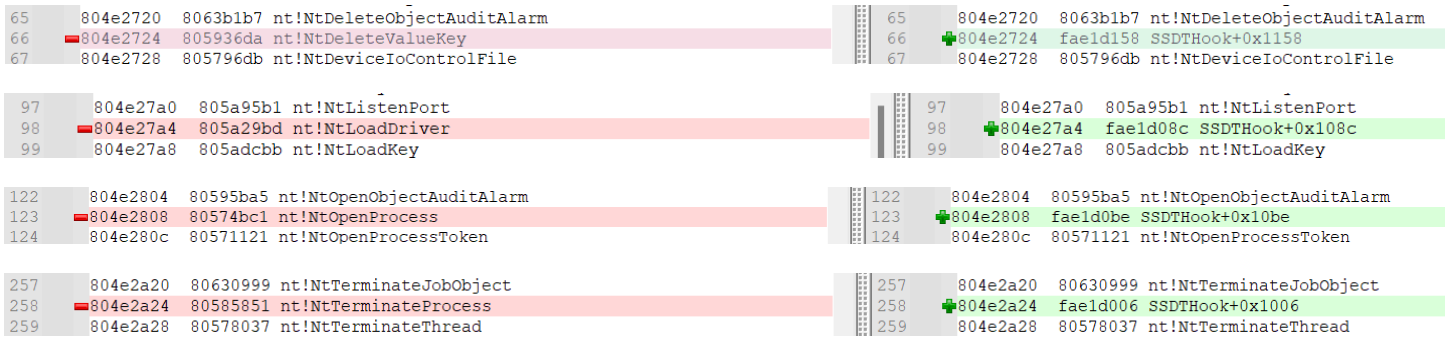
Q2. Please use WinDBG and type the following command:
dds KiServiceTable L 12a
Please take a screenshot of the output result. What's the meaning of each column?

```
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
804e265c  8062702f nt!NtAllocateUserPhysicalPages
804e2660  8059df59 nt!NtAllocateUuids
804e2664  80569302 nt!NtAllocateVirtualMemory
804e2668  805d7867 nt!NtAreMappedFilesTheSame
804e266c  805a1387 nt!NtAssignProcessToJobObject
804e2670  804e2c2c nt!NtCallbackReturn
804e2674  80649b77 nt!NtDeleteBootEntry
804e2678  805c99ad nt!NtCancelIoFile
804e267c  804ec61c nt!NtCancelTimer
804e2680  805699ae nt!NtClearEvent
804e2684  80567c07 nt!NtClose
804e2688  8058d24c nt!NtCloseObjectAuditAlarm
804e268c  8064f9c0 nt!NtCompactKeys
804e2690  8058b776 nt!NtCompareTokens
```

Each line displays the address of a system service function along with the corresponding function name prefixed by nt!. The first column represents the memory address of the function within the kernel's address space. The second column displays the relative address of the function within the nt module, noted

by the module name followed by an exclamation mark (nt!). The third column shows the name of the system service function, such as NtAcceptConnectPort, NtAccessCheck, NtAccessCheckAndAuditAlarm, etc. These functions represent some of the system operations that user-mode applications can invoke through system calls to interact with the operating system kernel.

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



Above are screenshots of the four functions that have been hooked. On the left are “before hook” and on the right are “after hook”.

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

The malware is pursuing a variety of objectives, including control, stealth, and possibly even remote access or data exfiltration. By hooking critical system functions related to security and process management, as shown in Question 3, the malware demonstrates an intention to manipulate system operations and evade detection. These hooks could facilitate activities like hiding its presence in the registry, loading malicious drivers stealthily, controlling processes to avoid detection, and preventing termination to maintain persistence. Additionally, hooks in functions related to network communications or security could suggest attempts to intercept or manipulate network traffic, suppress logging mechanisms, or enable unauthorized remote access. To summarize, the malware behaves in a way consistent with rootkit tactics. Specifically, compromises system security and integrity while remaining undetected and retaining control over the compromised system.

Analysis:

Generated by ChatGPT- To analyze, students explored the System Service Descriptor Table (SSDT) using WinDBG, gaining insights into the structure and

function of this critical component of the Windows kernel. The output of the command dds KiServiceTable L 12a provided a detailed view of system service functions and their corresponding memory addresses within the kernel's address space. Through this examination, we identified the significance of each column in the output result, highlighting the role of the SSDT in facilitating essential system operations for user-mode applications. Furthermore, we speculated on the potential implications of hooked functions identified in the SSDT, suggesting the malware's intentions to manipulate system operations, evade detection, and potentially enable unauthorized access or data exfiltration. This analysis underscores the importance of understanding SSDT and its relevance in detecting and analyzing rootkit behavior within the Windows operating system.

3. Discussion and Conclusion:

Generated by ChatGPT- The lab effectively fulfilled its objective of delving into Kernel Mode Rootkits and employing WinDBG alongside VirtualBox for Windows kernel debugging. Through hands-on experimentation, we gained a practical understanding of setting up kernel-mode debugging and scrutinizing the behavior of the Windows XP virtual machine kernel. Analysis of the System Service Descriptor Table (SSDT) using WinDBG confirmed its pivotal role in facilitating system functions and vulnerability to malware interception. Examination of hooked functions revealed the malware's likely intentions to manipulate system operations, evade detection, and potentially enable unauthorized access or data exfiltration. While the exact functionality of the malware remains speculative, the lab provided valuable insights into the complexities of kernel-level security threats and equipped us with essential skills for detecting and analyzing rootkit behavior. Moving forward, continued exploration and research in this domain are crucial for developing robust cybersecurity measures to mitigate the risks associated with kernel-level vulnerabilities and exploits.