Lab 17 Solutions - The Case of Mader Rootkit

Lab 17: The Case of Mader Rootkit

From the memory image (*mader.vmem*)

- Identify the SSDT hooks?
- Which kernel functions are hooked?
- Which malicious driver is implementing the hooks?
- Based on the hooks, what type of activity the Rootkit is monitoring?
- Is the driver using any other functionality to monitor the system activity?
- Can you dump the malicious driver and check if the driver is malicious?

Bonus Question:

- Can you disassemble the hooking function and identify the code where it is calling the hooked function?

Answers

01. Identify the SSDT hooks?

Running the ***ssdt*** plugin and looking for any functions not owned by either ***ntoskrnl.exe*** or ***win32k.sys*** shows the ***ssdt*** hooks. In this case, the highlighted functions are owned by a driver "***core.sys***"

```
root@kratos:~/Volatility# python vol.py -f mader.vmem ssdt | egrep -v "ntoskrnl|win32k"
Volatility Foundation Volatility Framework 2.5
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 80501b8c with 284 entries
  Entry 0x0019: 0xf61cd74e (NtClose) owned by core.sys
  Entry 0x0029: 0xf61cd604 (NtCreateKey) owned by core.sys
  Entry 0x003f: 0xf61cd6a6 (NtDeleteKey) owned by core.sys
  Entry 0x0041: 0xf61cd6ce (NtDeleteValueKey) owned by core.sys
  Entry 0x0062: 0xf61cd748 (NtLoadKey) owned by core.sys
  Entry 0x0077: 0xf61cd4a7 (NtOpenKey) owned by core.sys
  Entry 0x00c1: 0xf61cd6f8 (NtReplaceKey) owned by core.sys
  Entry 0x00cc: 0xf61cd720 (NtRestoreKey) owned by core.sys
  Entry 0x00f7: 0xf61cd654 (NtSetValueKey) owned by core.sys
SSDT[1] at bf999b80 with 667 entries
```

02. Which kernel functions are hooked?
In this case, the registry related kernel functions (***NtClose, NtCreateKey, NtDeleteKey*** etc.) are hooked as shown in the above screenshot.

03. Which malicious driver is implementing the hooks?
The malicious driver implementing the hooks is ***core.sys***, normally all the kernel functions are exported by either ***ntoskrnl.exe*** or ***win32.sys***

04. Based on the hooks, what type of activity the Rootkit is monitoring?
Based on the hooks it can be seen that the Rootkit is intercepting the registry related functions.

05. Is the rootkit using any other functionality to monitor the system activity?
Yes, the Rootkit is using callback functions to monitor system activity. Running the callbacks plugin shows the Rootkit installed a callback with the ***PsSetCreateProcessNotifyRoutine***. This allows the Rootkit to receive notifications whenever a new process starts or exits

06. Can you dump the malicious driver and check if the driver is malicious?

The malicious driver can be dumped from the memory to disk using the ***moddump*** plugin and by providing the base address. To determine the base address modules plugin can be used as shown in the screenshot.

```
root@kratos:~/Volatility# python vol.py -f mader.vmem modules | grep -i "core.sys"
Volatility Foundation Volatility Framework 2.5
0x8135e850 core.sys                  0xf61cb000    0x12000 \SystemRoot\system32\drivers\core.sys
root@kratos:~/Volatility# python vol.py -f mader.vmem moddump -b 0xf61cb000 -D dump/
Volatility Foundation Volatility Framework 2.5
Module Base Module Name          Result
---------- -------------------- ------
0x0f61cb000 core.sys             OK: driver.f61cb000.sys
```

Submitting the dumped component to VirusTotal confirms it to be malicious as shown in the screenshot.

Analysis | File detail | Additional information | Comments 0 | Votes

Antivirus	Result	Update
Ad-Aware	Gen:Variant.Zusy.20036	20161215
AegisLab	Rootkit.W32.Agent!c	20161215
AhnLab-V3	Trojan/Win32.Agent.C26207	20161215
ALYac	Gen:Variant.Zusy.20036	20161215
Antiy-AVL	Trojan[Rootkit]/Win32.Agent	20161215
Arcabit	Trojan.Zusy.D4E44	20161215
AVG	Hider.ABCF	20161215
Avira (no cloud)	TR/Rootkit.Gen	20161215

07. Can you disassemble the hooking function and identify the code where it is calling the hooked function?

In this example lets disassemble a hooking function ***0xf61cd604 (NtCreateKey)*** owned by "***core.sys***", ***volshell*** plugin can be used to disassemble the function

```
root@kratos:~/Volatility# python vol.py -f mader.vmem ssdt | egrep -v "ntoskrnl|win32k"
Volatility Foundation Volatility Framework 2.5
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 80501b8c with 284 entries
  Entry 0x0019: 0xf61cd74e (NtClose) owned by core.sys
  Entry 0x0029: 0xf61cd604 (NtCreateKey) owned by core.sys
  Entry 0x003f: 0xf61cd6ab (NtDeleteKey) owned by core.sys
  Entry 0x0041: 0xf61cd6ce (NtDeleteValueKey) owned by core.sys
  Entry 0x0062: 0xf61cd748 (NtLoadKey) owned by core.sys
  Entry 0x0077: 0xf61cd4a7 (NtOpenKey) owned by core.sys

root@kratos:~/Volatility# python vol.py -f mader.vmem volshell
Volatility Foundation Volatility Framework 2.5
Current context: System @ 0x819cc830, pid=4, ppid=0 DTB=0x319000
Python 2.7.11+ (default, Apr 17 2016, 14:00:29)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: dis(0xf61cd604)
0xf61cd604 55                                PUSH EBP
0xf61cd605 8bec                              MOV EBP, ESP
0xf61cd607 56                                PUSH ESI
0xf61cd608 57                                PUSH EDI
```

Disassembling the functions shows the malware calling a function by reading a dword value from the address ***0xf61dbea0***

The address ***0xf61dbea0*** contains a dword value of ***0x8061a286***. This means when ***NtCreateKey*** function is called, it will execute the malicious function at ***0xf61dbea0*** (owned by ***core.sys***) which in turn performs some malicious activity and then calls the function at address ***0x8061a286***

```
0xf61cd635 eb17                 JMP 0xf61cd64e
0xf61cd637 ff7520               PUSH DWORD [EBP+0x20]
0xf61cd63a ff751c               PUSH DWORD [EBP+0x1c]
0xf61cd63d ff7518               PUSH DWORD [EBP+0x18]
0xf61cd640 ff7514               PUSH DWORD [EBP+0x14]
0xf61cd643 56                   PUSH ESI
0xf61cd644 ff750c               PUSH DWORD [EBP+0xc]
0xf61cd647 57                   PUSH EDI
0xf61cd648 ff15a0be1df6         CALL DWORD [0xf61dbea0]
0xf61cd64e 5f                   POP EDI
0xf61cd64f 5e                   POP ESI
0xf61cd650 5d                   POP EBP

In [4]: dd(0xf61dbea0, length=4)
f61dbea0  8061a286

In [5]:
```

Inspecting the SSDT table on a clean system (in this case WindowsXP SP3) shows that the address ***0x8061a286*** is associated with the ***NtCreateKey*** function implemented by the kernel (***ntoskrnl.exe***) as shown in the below screenshot, this shows how Rootkit performs hooking and then later redirect control to the actual function.

```
80501c08   805998e8 nt!NtConnectPort
80501c0c   80540e00 nt!NtContinue
80501c10   806389aa nt!NtCreateDebugObject
80501c14   805b3c6e nt!NtCreateDirectoryObject
80501c18   80605124 nt!NtCreateEvent
80501c1c   8060d3c6 nt!NtCreateEventPair
80501c20   8056e27c nt!NtCreateFile
80501c24   8056dc5a nt!NtCreateIoCompletion
80501c28   805cb888 nt!NtCreateJobObject
80501c2c   805cb5c0 nt!NtCreateJobSet
80501c30   8061a286 nt!NtCreateKey
80501c34   8056e38a nt!NtCreateMailslotFile
80501c38   8060d7be nt!NtCreateMutant
80501c3c   8056e2b6 nt!NtCreateNamedPipeFile
80501c40   805a0da8 nt!NtCreatePagingFile
80501c44   8059a404 nt!NtCreatePort
```