Challenges in Kernel-Mode Memory Scanning

October 2, 2009

Aditya Kapoor
Research Scientist
McAfee

Rachit Mathur
Research Scientist
McAfee

Virus Bulletin Conference
23rd – 25th September, 2009
Geneva, Switzerland

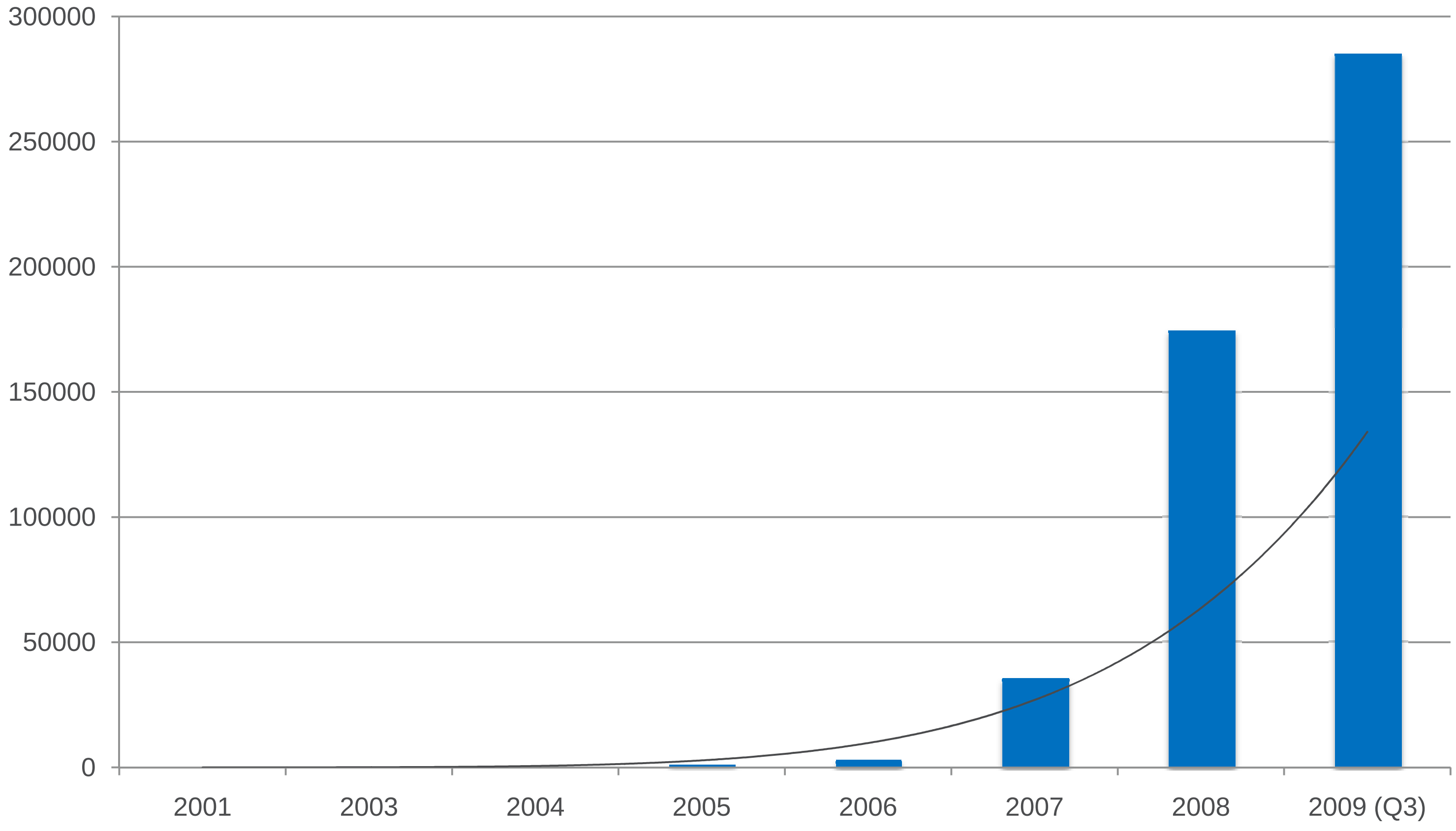
Agenda

McAfee

- Introduction
- Trends
- Techniques
- Concept
- Workings and discussion
- Real world examples
- Conclusion & Questions

Trends and Techniques

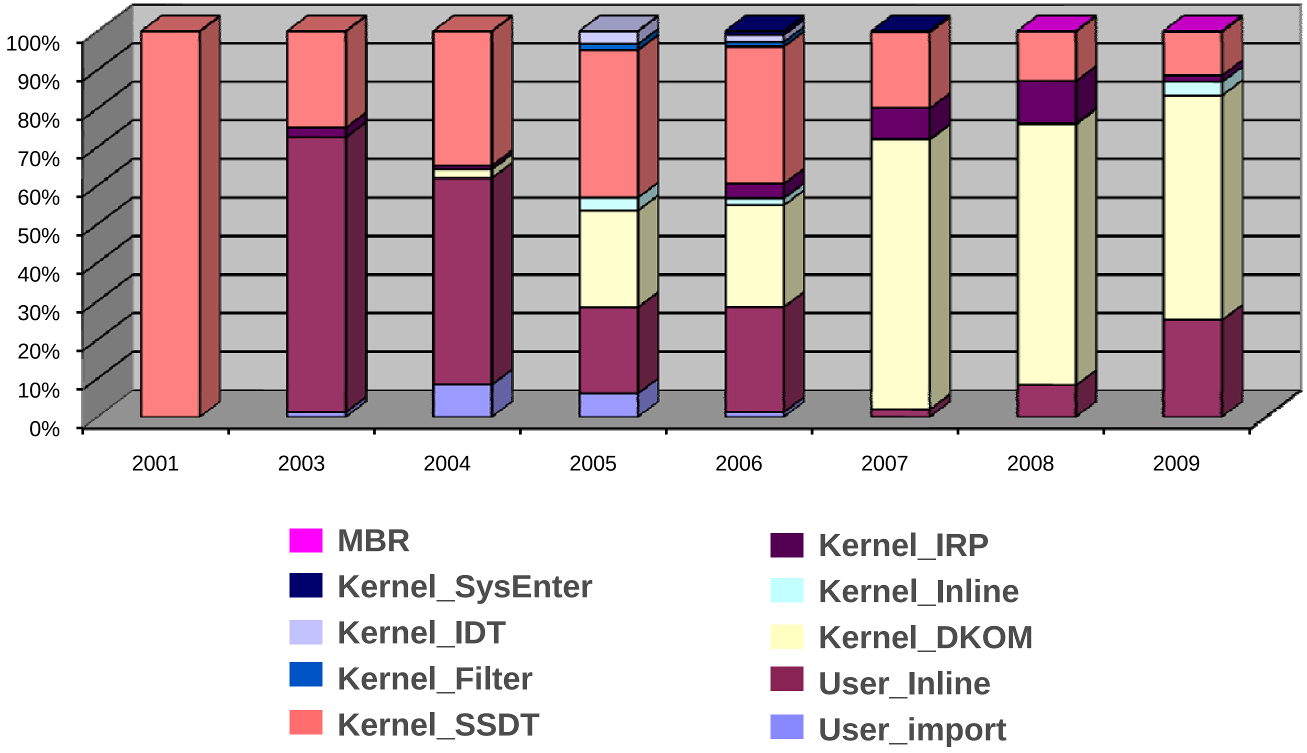
- Exponential growth of malware with rootkit capabilities.



Trends and Techniques

- Exponential growth of malware with rootkit capabilities.

- Popular kernel mode techniques growth.



Techniques employed by various rootkits

Inline hooks

- HackerDefender
- PWS-progent
- W32/feebs
- NTIllusion
- Vanquish

Import Table hooks

- Adcliker-BA
- Qoolaid

DKOM

- Backdoor-AWQ
- FuRootkit
- Vanti

Inline hook (Kernel)

- Apropos

IRP hook

- PWS-Gogo
- Spam-mailbot.c

SSDT hook

- Backdoor-CKB
- Backdoor-DKD

IDT hook

- Apropos

Sysenter hook

- Spam-mailbot.c

Filter driver

- SearchNet
- PigSearch

MBR

- StealthMBR

Revisiting Kernel Memory Scanners?

- Memory scanners have been talked about previously, this presentation covers:
  - Advances in kernel memory manipulation by malware.
  - Few ideas of efficient logic to pinpoint the suspicious objects.
  - Few ideas of how the scanner can help in correlation of suspicious data to aid in detection, cleaning and classification.
- Usually only interested in techniques that hinders detection or cleaning.
- Ironically memory manipulation techniques may aid in creating generic memory based detections.

Revisiting Memory Scanners?

- For an AV solution we need something more than an analyzer and heuristic detector.
  - Analyzers include tools like GMER, RKUnhooker, Rootkit Detective, IceSword etc.
  - Analogy can be hijackthis logs. ☺
- The role of a kernel mode scanner is to help in detection, classification and collating details, to clean the system and restoring the memory.

Concept

- Kernel mode manipulation categories
  - DKOM or DKOH
  - Detour based
  - Filter based

- Kernel memory scanner working
  - Module parsing
  - Detour Traversal
  - Hidden File/Process discovery

Concept (Module parsing)

- Enumerate listed modules
  - Scan the corresponding files or parse the memory structure to detect in memory

Advantages:

a) Simple implementation

b) No major changes required when new or unknown techniques of hooking are discovered

Disadvantages:

a) Ineffective when modules are hidden or not present.

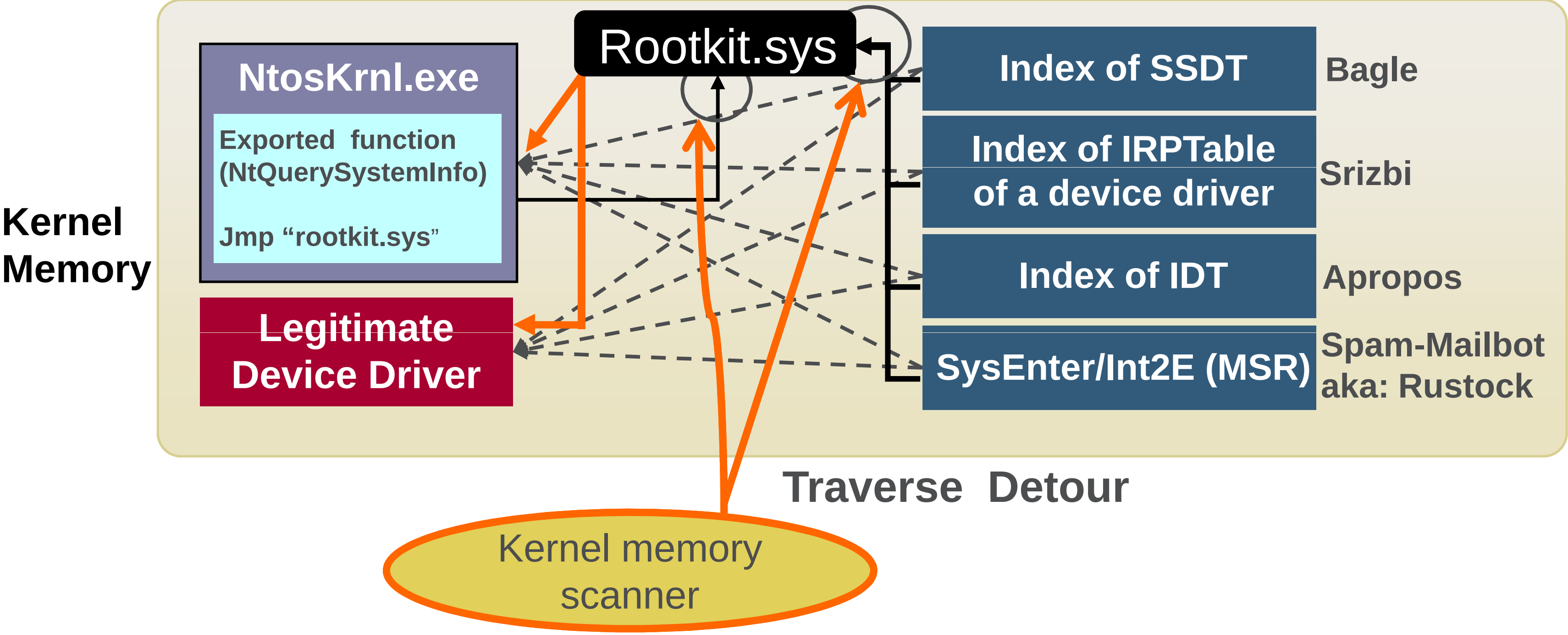
b) Performance intensive due to parsing the header of modules to scan the memory.

c) Costly to find relevant code patterns for detection.

d) Does not provide information that can aid in cleaning.

Concept (Detour Traversal)

- – Identify detour logics in memory
- – Traverse the detour to a memory region or a module’s memory.
- – Detect on the most relevant code.
- – Restore Detours.



Detour traversal

Advantages:

a) Improves scanning performance
b) Less likely to false due to context of scan object.
c) Detection tends to last longer.
d) Not dependent on module enumeration
e) Scalable once the framework is developed.

Disadvantages:

a) Needs to be updated when a new or unknown detour technique is encountered.

DKOM & DKOH.

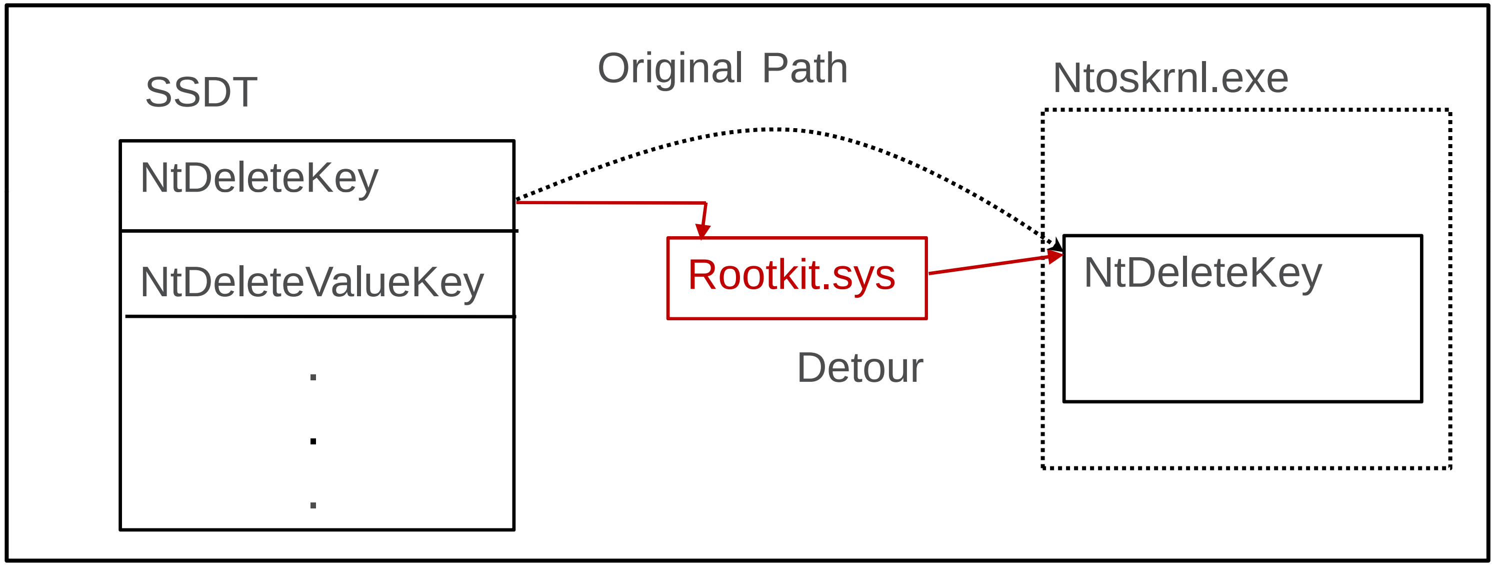
- Direct kernel manipulation and Direct kernel object hooking
  - The memory manipulation can be done via '\device\physicalmemory' access.
  - Or, using a kernel a component.
  - Example targets are EPROCESS list, module list and object_type structure.
  - DKOH is still detour based, so apply detour parsing.
  - In DKOM , there is no notion of kernel memory or module. Kernel scanner however can scan the hidden file or process memory.

Kernel scanning must haves.

1. Logic to determine that pointers are out of the ordinary Location.
2. Capability to disassemble and analyze portions of kernel memory.
3. Capability to read and analyze the most common kernel structures.
4. Capability to follow the jumps and detours.
5. Capability to scan and analyze any given kernel module.
6. Capability to write safely into kernel memory
   a) A rootkit can attack by watching for writes and taking action.
7. A static or runtime database of common pointer locations.
8. A programmable interface which provides access to low-level APIs.

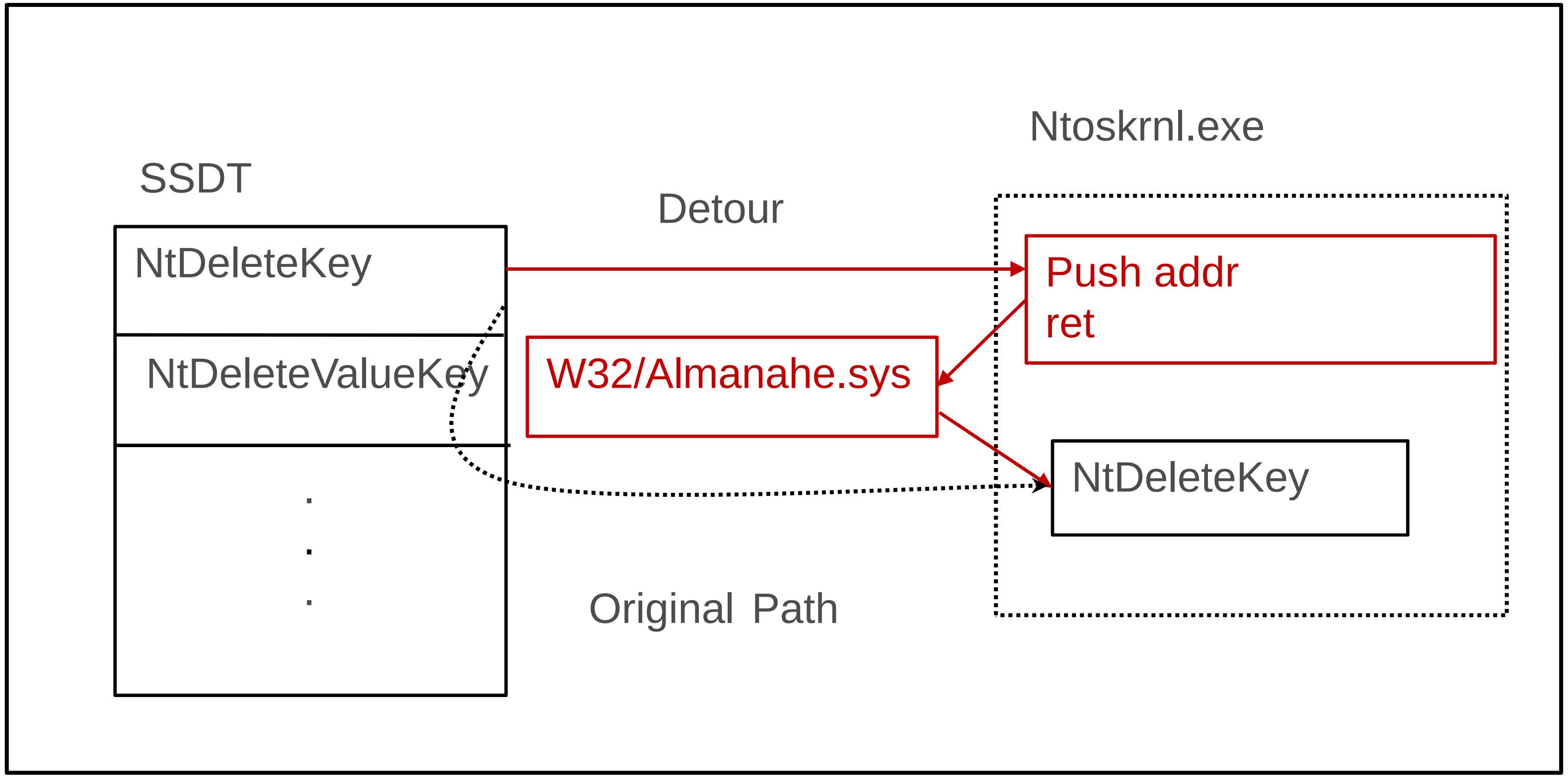
Workings and discussion

- It is desirable that the signature
  - be accurate, classify into families with no false positives
  - be quick, aid in repair and be generic
- Use combination of how we identify a rootkit module and fingerprint of the module.



Follow all detours

- Eventually lead to the rootkit module
  - Challenge in : *Capability to follow the jumps and detours to eventually lead to the malicious kernel module*



When in doubt

- If it is complex to follow the detour?
  - Challenge in : *Logic to determine that pointers are out of the ordinary location*

NtQuerySystemInformation (Original)		NtQuerySystemInformation (Apropos Hook)	
68 10 02 00 00	push 210h	68 10 02 00 00	push 210h
68 58 6E 41 00	**push 416E58**	**50**	**push eax**
E8 95 D9 F5 FF	**call sub_40BE73**	**8BC3**	**mov eax, ebx**
		2BC3	**sub eax, ebx**
		48	**dec eax**
		8B38	**mov edi, ptr:[eax]**

Raise exception

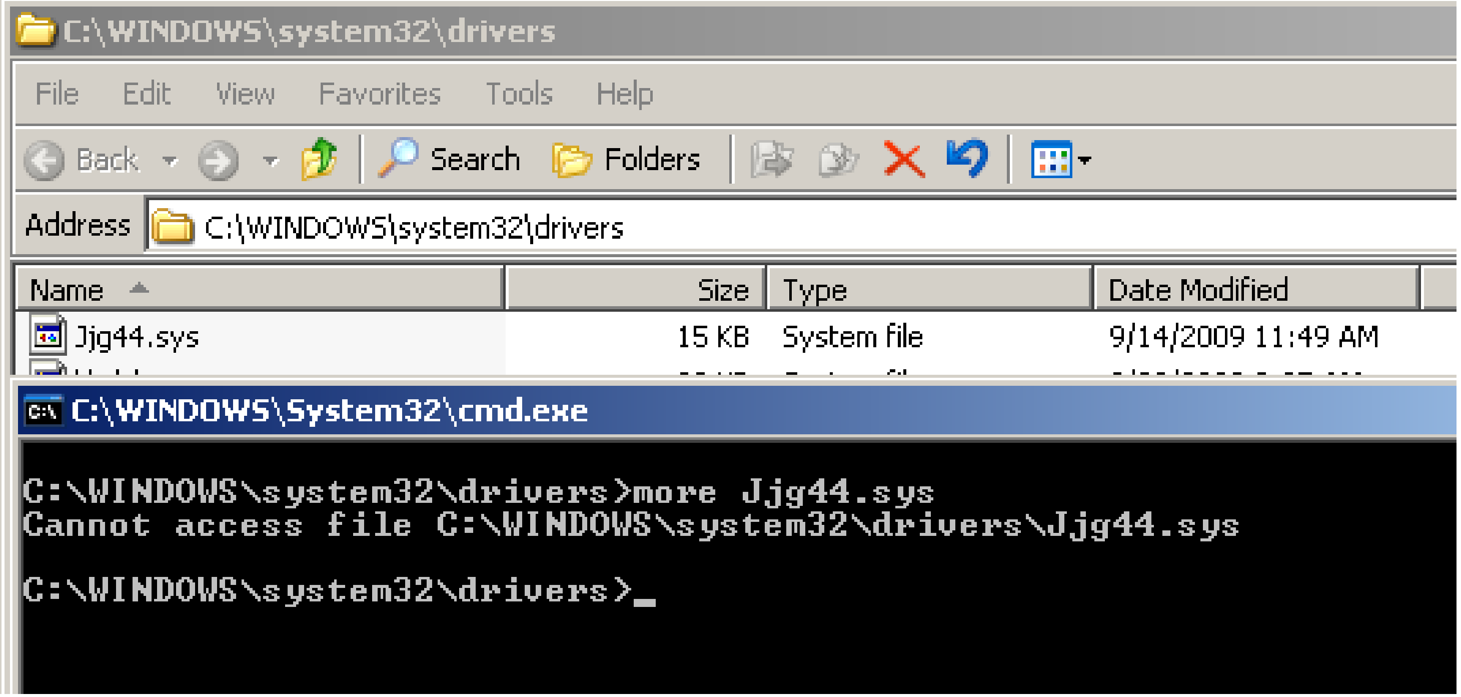
Apropos trojan

Case Studies

- Once the malware has infected and is active
  - Detect
  - Classify
  - Aid in cleaning
- Cutwail
- MBR rootkit

Cutwail rootkit

- Drops a sys file and prevents access to it
  - %system%\drivers\Jjg44.sys



C:\WINDOWS\system32\drivers
File Edit View Favorites Tools Help
Back Search Folders
Address C:\WINDOWS\system32\drivers

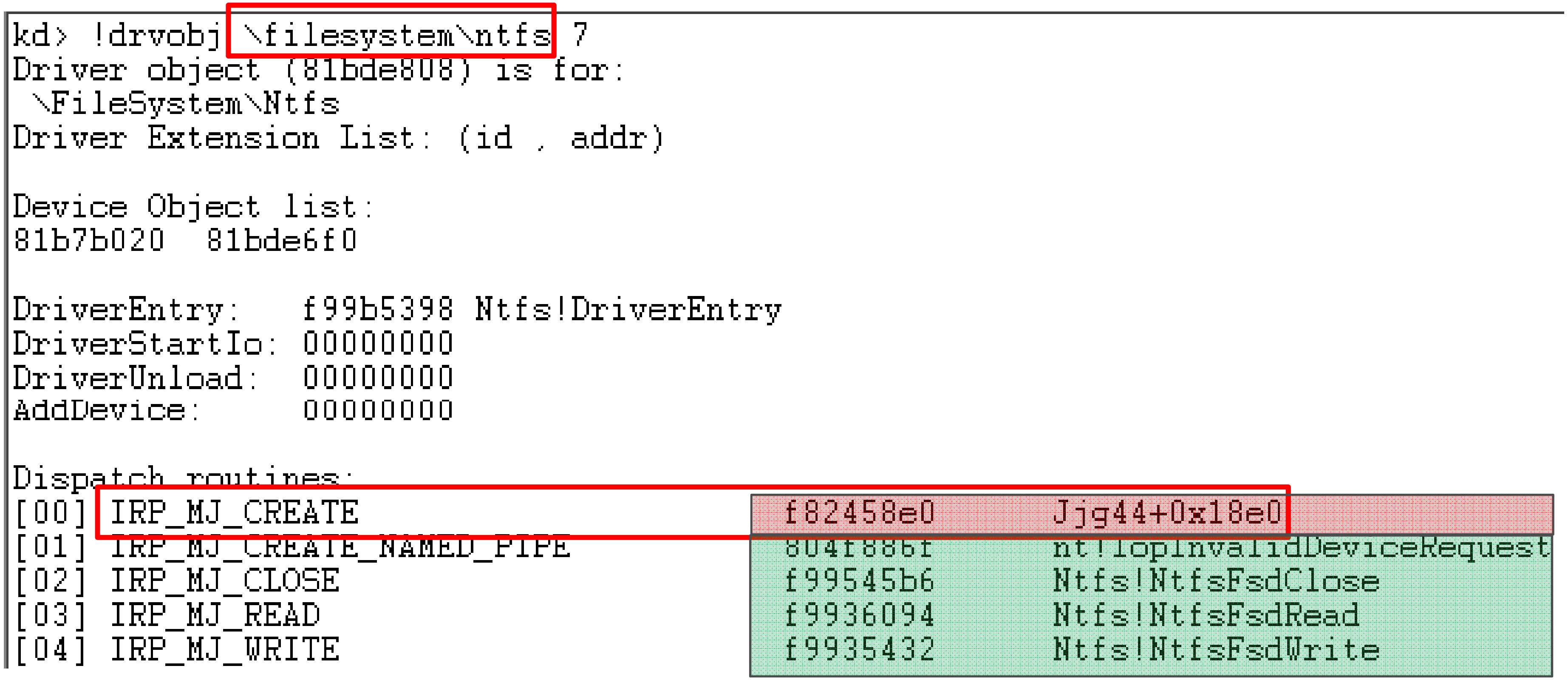
Name	Size	Type	Date Modified
Jjg44.sys	15 KB	System file	9/14/2009 11:49 AM

C:\WINDOWS\System32\cmd.exe

```
C:\WINDOWS\system32\drivers>more Jjg44.sys
Cannot access file C:\WINDOWS\system32\drivers\Jjg44.sys

C:\WINDOWS\system32\drivers>_
```

- File not hidden but cannot read to detect or delete this file.

Cutwail detection

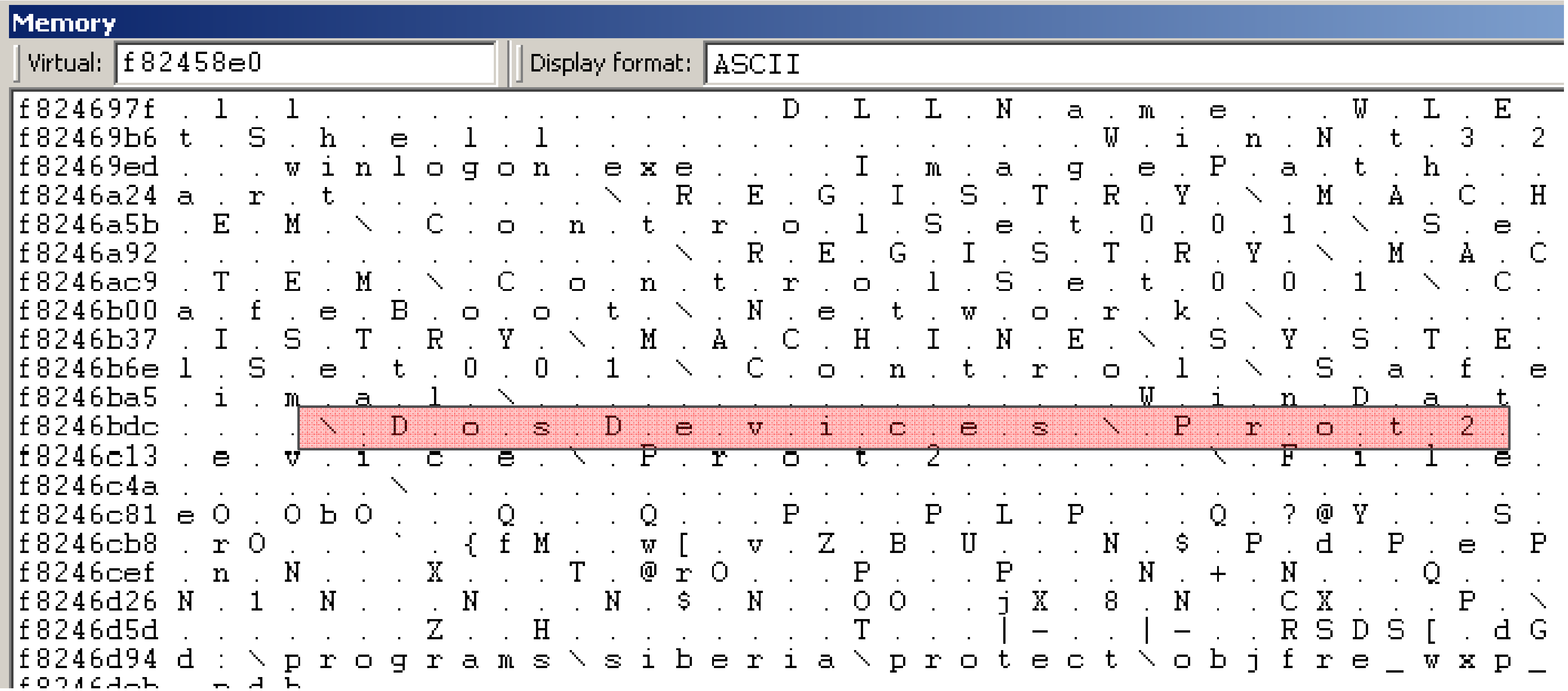


```
kd> !drvobj \filesystem\ntfs 7
Driver object (81bde808) is for:
 \FileSystem\Ntfs
Driver Extension List: (id , addr)

Device Object list:
81b7b020  81bde6f0

DriverEntry:   f99b5398 Ntfs!DriverEntry
DriverStartIo: 00000000
DriverUnload:  00000000
AddDevice:     00000000

Dispatch routines:
[00] IRP_MJ_CREATE                      f82458e0      Jjg44+0x18e0
[01] IRP_MJ_CREATE_NAMED_PIPE           804f886f      nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       f99545b6      Ntfs!NtfsFsdClose
[03] IRP_MJ_READ                        f9936094      Ntfs!NtfsFsdRead
[04] IRP_MJ_WRITE                       f9935432      Ntfs!NtfsFsdWrite
```

- File access is denied using hook on IRP_MJ_CREATE on NTFS.

Cutwail detection

- Hook directly lands into the malicious module
- Detection signature can be written
  - Detour path + byte fingerprint

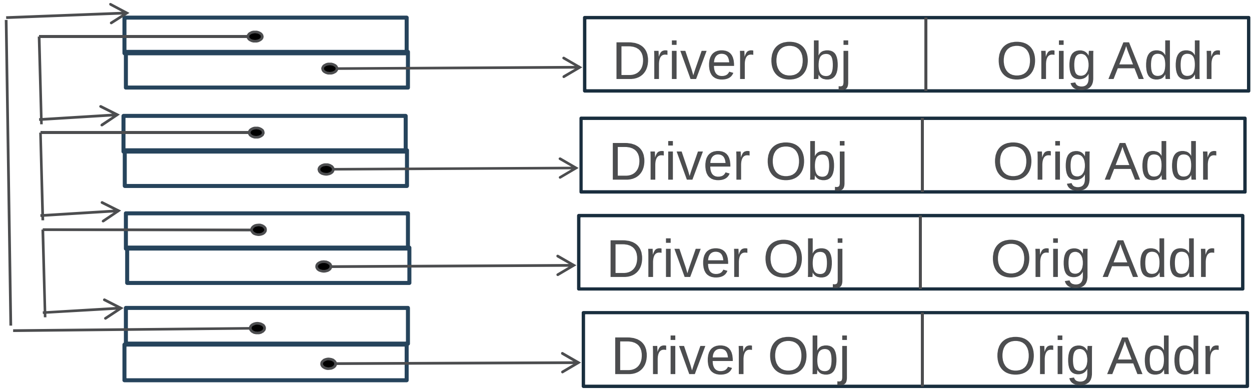


Virtual: f82458e0 Display format: ASCII

```
f824697f . l . l . . . . . . . . . . . . . D . L . L . N . a . m . e . . . W . L . E .
f82469b6 t . S . h . e . l . l . . . . . . . . . . . . . . . W . i . n . N . t . 3 . 2
f82469ed . . . w i n l o g o n . e x e . . . . I . m . a . g . e . P . a . t . h . . .
f8246a24 a . r . t . . . . . . . \ . R . E . G . I . S . T . R . Y . \ . M . A . C . H
f8246a5b . E . M . \ . C . o . n . t . r . o . l . S . e . t . 0 . 0 . 1 . \ . S . e .
f8246a92 . . . . . . . . . . . . . . \ . R . E . G . I . S . T . R . Y . \ . M . A . C
f8246ac9 . T . E . M . \ . C . o . n . t . r . o . l . S . e . t . 0 . 0 . 1 . \ . C .
f8246b00 a . f . e . B . o . o . t . \ . N . e . t . w . o . r . k . \ . . . . . . . .
f8246b37 . I . S . T . R . Y . \ . M . A . C . H . I . N . E . \ . S . Y . S . T . E .
f8246b6e l . S . e . t . 0 . 0 . 1 . \ . C . o . n . t . r . o . l . \ . S . a . f . e
f8246ba5 . i . m . a . l . \ . . . . . . . . . . . . . . . . . W . i . n . D . a . t .
f8246bdc . . . . \ . D . o . s . D . e . v . i . c . e . s . \ . P . r . o . t . 2 . .
f8246c13 . e . v . i . c . e . \ . P . r . o . t . 2 . . . . . . . \ . F . i . l . e .
f8246c4a . . . . . . \ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
f8246c81 e O . O b O . . . Q . . . Q . . . P . . . P . L . P . . . Q . ? @ Y . . . S .
f8246cb8 . r O . . . ` . { f M . . w [ . v . Z . B . U . . . N . $ . P . d . P . e . P
f8246cef . n . N . . . X . . . T . @ r O . . . P . . . P . . . N . + . N . . . Q . . .
f8246d26 N . 1 . N . . . N . . . N . $ . N . . O O . . j X . 8 . N . . C X . . . P . \
f8246d5d . . . . . . . Z . . H . . . . . . . . T . . . | - . . | - . . R S D S [ . d G
f8246d94 d : \ p r o g r a m s \ s i b e r i a \ p r o t e c t \ o b j f r e _ w x p _
f8246dcb   n d h
```

Cutwail cleaning

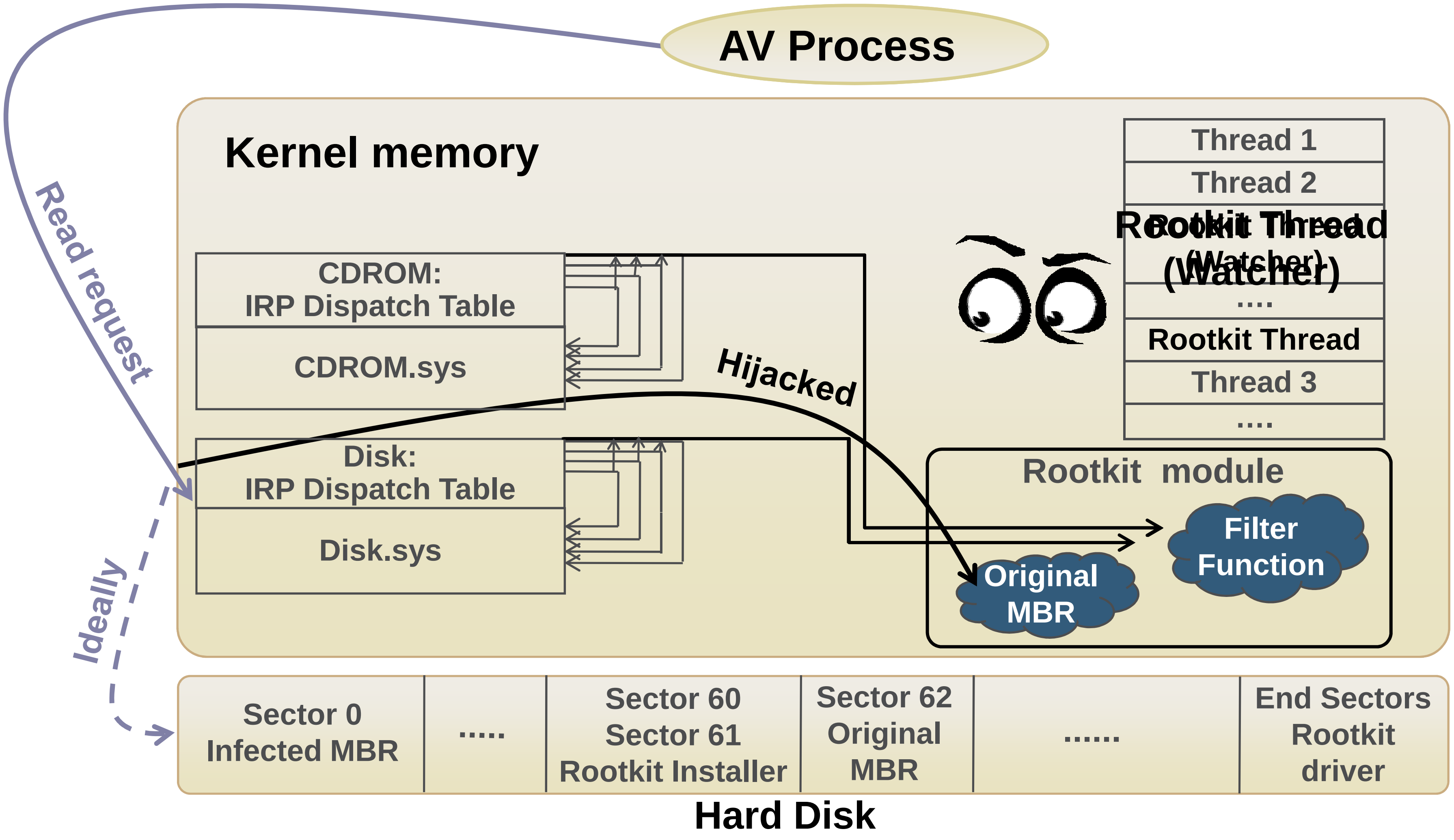
- Obtain module name
  - Disable unprotected registry
  - Delete file during reboot
- Hook restoration
  - Can be tricky!
  - Keep track of changes from early in boot process
  - Extract original address from malware itself
    - Challenge in: *Capability to disassemble and analyze any arbitrary portions of kernel memory*



StealthMBR rootkit

- StealthMBR aka Mebroot infects MBR to gain control very early in boot process
- Does not require any file or registry to sustain itself
- Prevents access to MBR
- Primarily hooks IRP dispatch table
- Challenge in : *Logic to determine that pointers are out of the ordinary location*

StealthMBR detection



AV Process

Read request

Ideally

Kernel memory

CDROM:
IRP Dispatch Table

CDROM.sys

Disk:
IRP Dispatch Table

Disk.sys

Hijacked

Thread 1

Thread 2

Rootkit Thread
(Watcher)

….

Rootkit Thread

Thread 3

….

Rootkit module

Original
MBR

Filter
Function

Sector 0
Infected MBR

…..

Sector 60
Sector 61
Rootkit Installer

Sector 62
Original
MBR

…….

End Sectors
Rootkit
driver

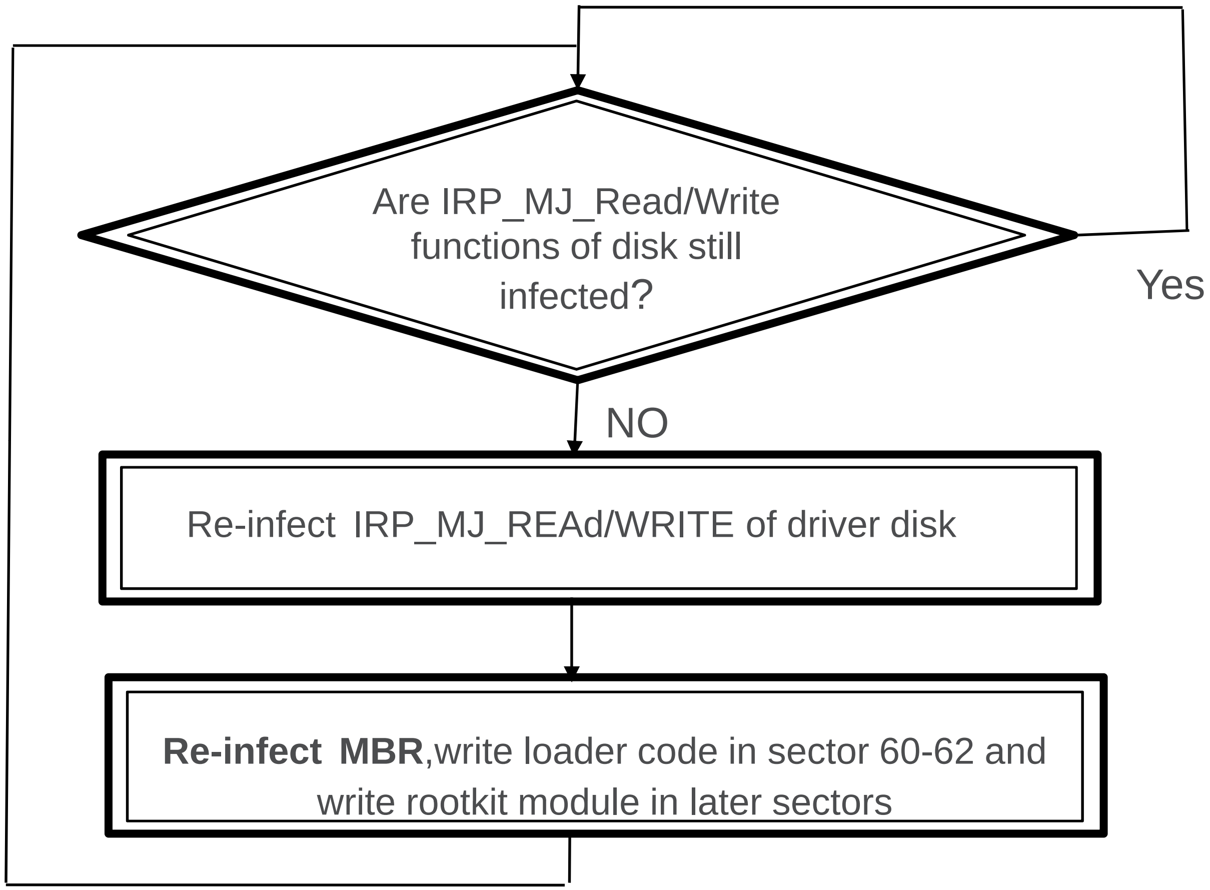
Hard Disk

Dispatch routines

IRP_MJ_CREATE	8196687e
IRP_MJ_CREATE_NAMED_PIPE	InvalidRequest
IRP_MJ_CLOSE	8196687e
IRP_MJ_READ	81961428
IRP_MJ_WRITE	81961428
IRP_MJ_QUERY_INFORMATION	InvalidRequest
IRP_MJ_SET_INFORMATION	InvalidRequest
IRP_MJ_QUERY_EA	InvalidRequest
IRP_MJ_SET_EA	InvalidRequest
IRP_MJ_FLUSH_BUFFERS	81966890
IRP_MJ_QUERY_VOLUME_INFORMATION	InvalidRequest
IRP_MJ_SET_VOLUME_INFORMATION	InvalidRequest
IRP_MJ_DIRECTORY_CONTROL	InvalidRequest
IRP_MJ_FILE_SYSTEM_CONTROL	InvalidRequest
IRP_MJ_DEVICE_CONTROL	8196688a
IRP_MJ_INTERNAL_DEVICE_CONTROL	81966884
IRP_MJ_SHUTDOWN	81966890

StealthMBR Cleaning

- Use watcher thread to repair MBR for you ☺



Are IRP_MJ_Read/Write functions of disk still infected?
Yes
NO
Re-infect IRP_MJ_REAd/WRITE of driver disk
Re-infect MBR,write loader code in sector 60-62 and write rootkit module in later sectors

StealthMBR Cleaning

- Use watcher thread to repair MBR for you ☺
- Create special IRP that can go through the rootkit filter
- Patch into areas that are not watched
- Hook restoration
  - – Suspend or kill watcher thread
  - – Restore IRP hooks
    - Challenge in : *A static or runtime database of common pointer locations*
  - – Repair MBR

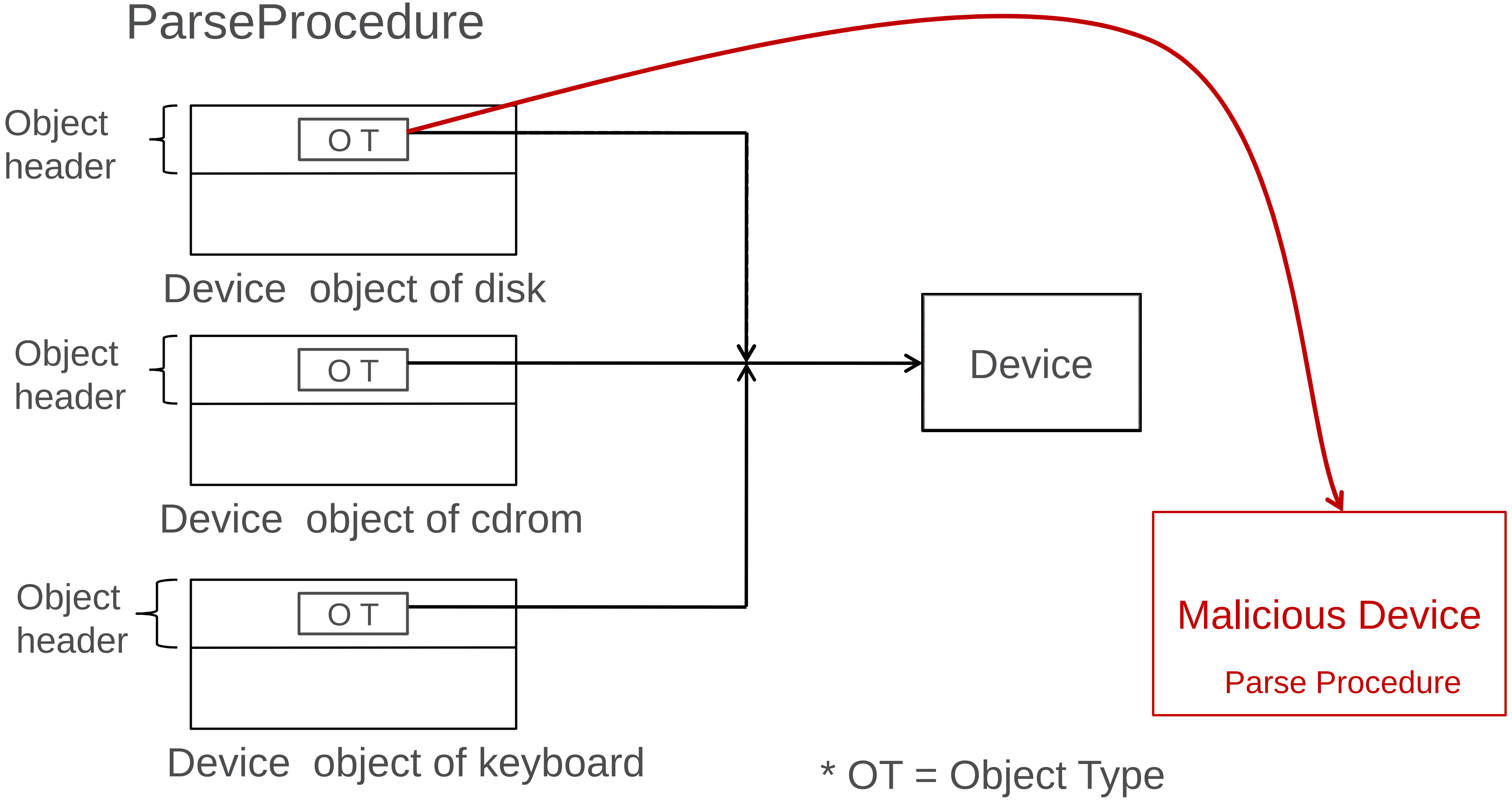
StealthMBR Cleaning

- Finding original address?
  - Hooks early so monitoring is difficult
  - Look for areas that are still not patched
  - Look inside malicious code

```
jmp     8196147e
mov     eax, [819D1F08h] ={CLASSPNP!ClassReadWrite}
mov     dword ptr [ebp-28h],eax
test    ebx,ebx
je      81961527
mov     eax,dword ptr [esi+40h]
```

Other variants

- Use Direct Kernel Object Hijacking
  - Hijack disk ‘Device’ _OBJECT_TYPE with ‘special’ ParseProcedure



* OT = Object Type

Other variants

- Use Direct Kernel Object Hijacking
  - Hijack disk ‘Device’ _OBJECT_TYPE with ‘special’ ParseProcedure
- Install IRP hooks on-demand
- For detection to start we can check if keyboard and mouse device have same _OBJECT_TYPE
- Some directly hook IRP of driver below \Driver\Disk in device stack of \\Device\\Harddisk0\\DR0

Thank You!

Suggestions & Questions:
Email: Aditya_Kapoor@avertlabs.com
Email: Rachit_Mathur@avertlabs.com

McAfee®