Predicting the future of stealth attacks

Aditya Kapoor, McAfee Labs
Rachit Mathur, McAfee Labs

Outline

- Introduction
- Last Decade of Stealth Malware
- Current Rootkit Threats and Challenges
- Discussion on Stealth Attack Trends
- Conclusions

Introduction

- Approximately10% of current malware use rootkit
- Rootkits are most prevalent in 32 bit Windows
- Handful of families so far for 64 bit
- Challenges once malcode enters kernel
  - Harder for rootkits to enter 64 bit kernel
  - Rootkits can infiltrate 64 bit OS Kernel by
    a) Bypassing driver signing check (e.g. using testsigning mode)
    b) Modifying the windows boot path (MBR etc)
    c) Kernel exploits in Windows kernel or third party drivers.
    d) Stealing valid digisigs (Similar to Stuxnet)

Few notable rootkits of the last decade

McAfee

- **2000**
  - **Ntrootkit-A** SSDT-Hook
- **2003**
  - **He4Hook** IRP
- **2004**
  - **FuRootkit** Eprocess
- **2005**
  - **CommonName** NTFS Filter
  - **Apropos** IDT, Kernel Inline
- **2006**
  - **Rustock** Kthread
  - **Rustock** SysEnter/Int2E
- **2008**
  - **StealthMBR** MBR, IRP
- **2009**
  - **Koutodoor** ParseProcedure
  - **StealthMBR** ParseProcedure
- **2010**
  - **Tdss.a** Lowest IRP
  - **Tdss.c** Lowest Device
- **2011**
  - **Tdss.E** Memory Forging
  - **Banker** Trust , Filter
  - **Cidox** VBR
  - **Mebromi** Bios, MBR, IRP.

Few recent rootkits

Koutodoor

Tdss.c

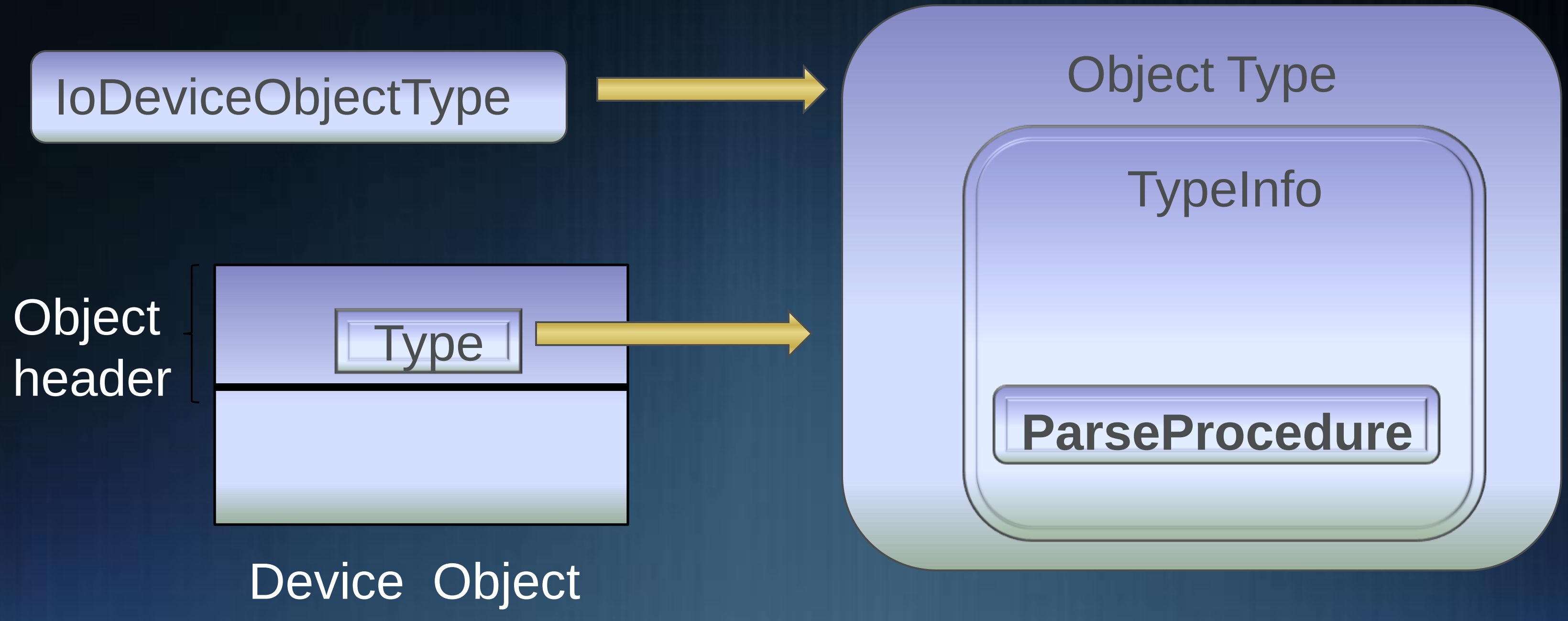
Tdss.e

Banker

Zeroaccess

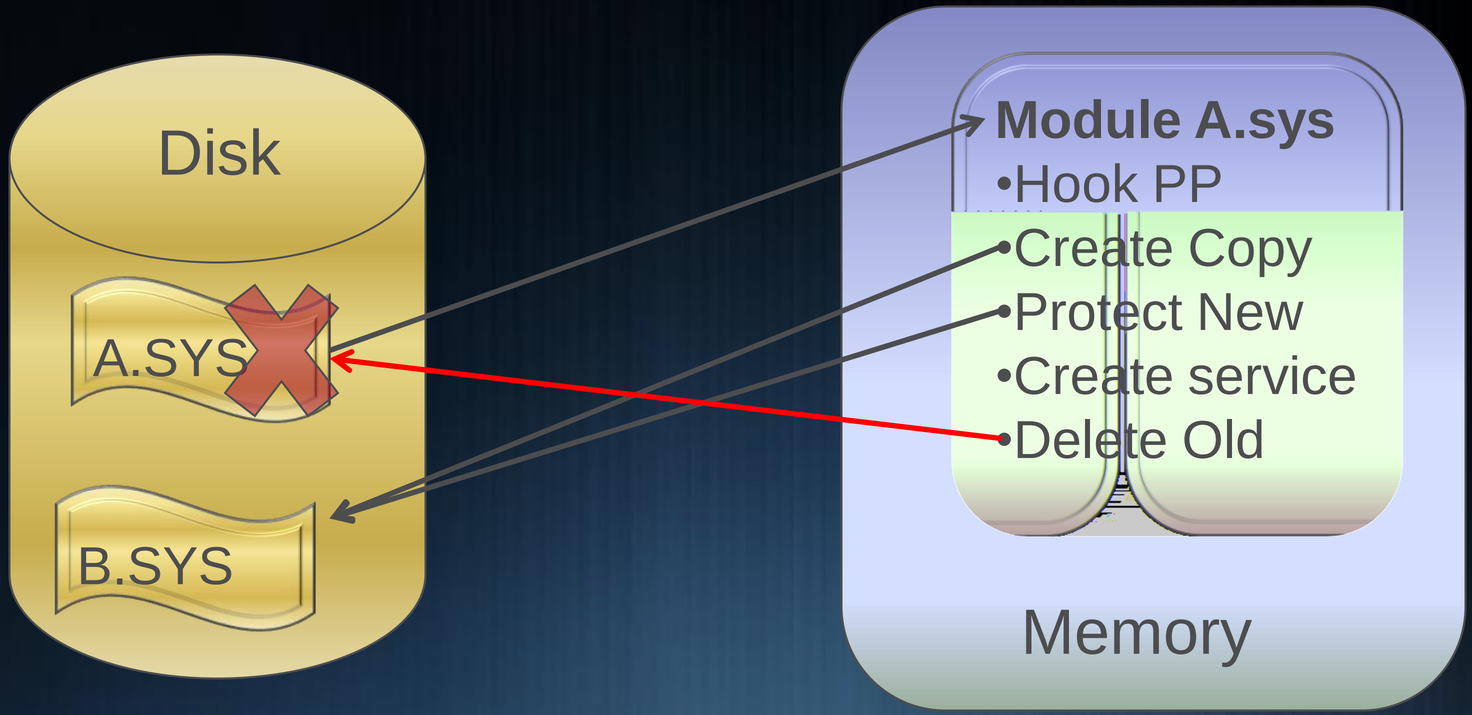
Koutodoor rootkit

- Polymorphic samples
- Majority of infections reported from China
- Protects its sys file in drivers directory (does not hide)
- Uses ParseProcedure hook for stealth



Koutodoor rootkit

- File name flip-flop on each boot



- Challenge in locating file-on-disk associated to the memory-module

Brazilian Banker rootkit

- Has both x86 and x64 versions
- Uses bcdedit to bypass driver signing requirements
  - DISABLE_INTEGRITY_CHECKS
  - TESTSIGNING ON
- Prevents security tools from loading
  - OS callbacks such as PsSetLoadImageNotifyRoutine

```
je      plusdriver+0x53e4 (872053e4)
mov     dword ptr [eax],1B8h
mov     dword ptr [eax+4],8C2C0h
push    edi
```

Patched Entry Point

```
B8010000c0    mov    eax, 0c0000001h
C20800        ret 8
................
```

Brazilian Banker rootkit

- Has both x86 and x64 versions

- Uses bcdedit to bypass driver signing requirements
  - DISABLE_INTEGRITY_CHECKS
  - TESTSIGNING ON

- Prevents security tools from loading
  - OS callbacks such as SetLoadImageNotifyRoutine
  - Also used by BlackEnergy rootkit
  - Image identification by name
  - Generic security tool identification

TDSS rootkit

- Has had many versions
  - Encrypted code in sectors

Tdss.c (TDL3)	Tdss.d (TDL4)	Tdss.e
• To persist reboot it infects random sys file and forges its contents • Hooks below device DR0 (of disk.sys) • Uses watcher mechanism to re-infect	• To persist reboot it infects MBR and forges its contents • Hooks below device DR0 (of disk.sys) • Uses watcher mechanism to re-infect	• To persist reboot it infects volsnap.sys and forges its contents • Dispatch table (IRP) hooks • Sets hardware breakpoints and hooks KiDebugRoutine to forge memory

Tdss - Memory forging

- Uses hardware breakpoints (DRX register setting) to monitor memory access
- Installs KiDebugRoutine hook to intercept breakpoint exception triggered by hardware breakpoint

```
mov     eax, [edi+CONTEXT._Esi]
mov     ecx, dword_41D818
cmp     eax, ecx            ; is ESI pointing to protected memory region?
jz      short loc_4040CF    ; jump to fake_memory
cmp     eax, edx
jnz     loc_403BC5          ; set as handled and return
cmp     eax, ecx
jnz     loc_403BC5          ; set as handled and return

fake_memory
                            ; CODE XREF: KiDebugRoutine_Hook+52E↑j
mov     eax, dword_41D830
add     eax, 38h
mov     [edi+CONTEXT._Esi], eax ; set thread ESI to fake memory location
jmp     loc_403BC5          ; set as handled and return
```

ZeroAccess / MAX++

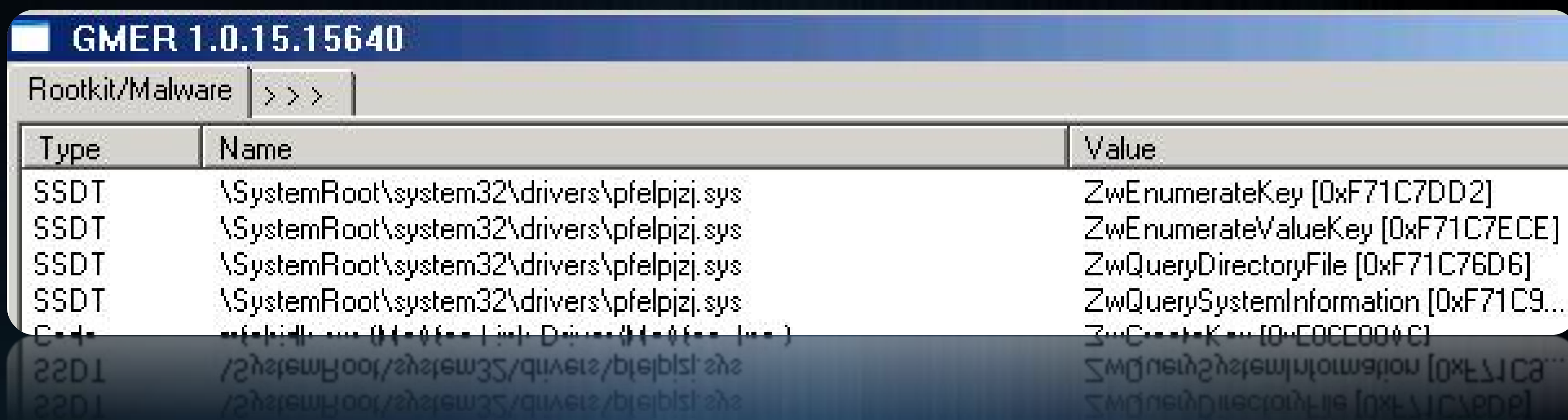
- To persist reboot it infects random sys file and forges its contents
- Hijacks LowerDeviceObject in DeviceExtension of DR0 device
- Attacks security software
  - Kill process
    - Schedules user-mode APC to call ExitProcess from within
  - Removes file permissions
    - ZwSetSecurityObject
  - Generic security tool identification
    - Bait process
    - Bail files etc

Trends in Stealth Malware

File forging	Memory forging	Self protection
Attack AV	Disassociating memory from file-on-disk	Removing dependency on files
	Untrusting the trusted	

Disassociating memory from file-on-disk

- Rootkit's memory can give-away its associated file on disk
  - For example: NtQuerySystemInformation with SysModuleInfo



- Memory scanner needs to know the file
  - For remediation the file-on-disk needs to be removed
  - Can be very generic for early load anti-rootkit solutions
- Techniques like : Koutodoor, patch Module_Entry etc
- Problem because it can be difficult to track kernel memory

File Content Forging

- Overwrite existing files and forge the ‘view’ such that AV gets the clean view instead of malicious

- File forging used by Tdss.c, ZeroAccess

- Better than hiding files
  - Hidden files can be located using file-system parsers etc
  - Comparing file contents is time consuming
  - Forging can work well from layers below file-system (NTFS)

- Can expect
  - Forging incorrect contents (mebromi forges zeros for MBR)
  - Create new malicious files but forge clean contents

Removing Dependency on Files

- Scanners based on direct file-system (FS) parsers worked well

- Therefore having no file in the FS helps rootkits, so :
  - Move malicious code to boot process: MBR or VBR or ???
  - Move malicious code to bios ☹
  - And move encrypted malicious code to raw sectors or as a file

- Has added advantages in 64-bit Windows

- Recent examples: TDSS.d, Whistler, Fisp, Popureb, Mebromi

Memory Forging

- Memory scanner relies on its view of kernel memory
- Most common places to hook can go unnoticed when ‘invisible’
- Hardware breakpoints and KiDebugRoutine hook
  - Hook and code variations can exist
  - Con: Complicated to implement and AV’s ability to counter by looking out for h/w breakpoints
- Other PoC have their own complexity and limitations as well

From Limited Self Preservation…

- Defend components and/or attack security components

- Watcher threads
  - To monitor and protect memory hooks and disk changes
    - TDSS, StealthMBR etc
  - To rewrite registry, files, gain exclusive locks from SYSTEM etc
    - Festi, NtRootkit-H, Tdss

- Attack from kernel (mostly name based)
  - Callback registration to attack during process, module load etc
    - Storm worm, BlackEnergy
  - ObfDereferenceObject
    - Simfect / QVOD

To Holistic Self Defenses

- Behavioral identification of AV
  - Identify the AV activity by monitoring the behavior of the process or thread, if it triggers their behavioral detection logic, the AV is terminated.

- Untrusting the AV and whitelisting of legitimate applications
  - Trust based deterrence in the rootkits
  - Threats establish trust on the essential drivers for the system and everything else could be locked out.
  - AV now has to find ways to get trusted by malware to get a chance to even load.

Conclusions

- Rootkit attacks continue to gain sophistication
  - Trends likely to rise
- Solution areas
  - Trust
  - Monitoring and tracking (like behavior, access protection)
  - Hypervisor

Thank You!

- Contact:
  - Aditya_Kapoor@McAfee.com
  - Rachit_Mathur@McAfee.com