In Combat against Rootkits

Robert Lipovsky
lipovsky@eset.sk
Virus Researcher

ESET

EICAR 2010
Presentation outline

1. Hiding vs. Detecting
   - NtQuerySystemInformation
   - ActiveProcessLinks list
   - Scheduler lists
   - SwapContext hook
   - Other methods...

2. POCs vs. Rootkits in the wild

3. Crooks in the wild
   - Mebroot
   - Rustock
   - Olmarik (TDL)

4. Conclusion
Hiding vs. Detecting
Windows API functions for enumerating processes

**CreateToolhelp32Snapshot** (kernel32.dll)
- parameter `dwFlags` = TH32CS_SNAPPROCESS
- Process32First, Process32Next

**EnumProcesses** (psapi.dll)
- Returns only PIDs

**NtQuerySystemInformation**
- ntdll.dll for User Mode
- Native API (system call)
- parameter `SystemInformationClass` = `SystemProcessInformation` (5)
- `CreateToolhelp32Snapshot` and `EnumProcesses` call this one
Hiding from the APIs

• processes can be excluded from the listing by hooking the relevant functions
• Native APIs are called via the SSDT
• Nt- and Zw- flavors of System calls
• hiding from NtQuerySystemInformation is trivial – SSDT hooking
• the detection of SSDT hooking is also trivial
  • check addresses in the SSDT - what module do they belong to?
  • replace with expected values from ntoskrnl.exe
SSDT hook
Kernel process objects

• Let's go deeper... how does NtQuerySystemInformation get its data?
• Processes are represented by EPROCESS structures in kernel memory
  • Contain a lot of information related to the process
  • These structures are linked in a double-linked list -
    • ActiveProcessLinks
• A list of processes can be acquired by traversing this list
• **Unlinking** an EPROCESS structure from the list is simple - just overwriting a few pointers
  
  • Direct Kernel Object Manipulation
  
  • ActiveProcessLinks list is for process *enumeration only*, so its modification doesn't affect the correct functioning of the hidden process
Figure: Unlinking an EPROCESS structure from ActiveProcessLinks list
Scheduler lists

- **Threads** are represented by **ETHREAD** structures in the kernel
  - Contain information related to threads (e.g. pointer to the owner **EPROCESS**)
- **Waiting threads** are in a kernel list beginning with **KiWaitListHead**
- **Ready threads** are in one of 32 lists for each priority (**KiDispatcherReadyListHead[32]**)
- However, since Windows XP, the scheduler has changed and not all processes can be found using this approach
- **Hiding**: A thread can be excluded from this list by modifying the scheduler
SwapContext

• We need a low level method which works on modern Windows
• The SwapContext function is used by the scheduler to replace the context (registers, stacks, etc.) of the currently running thread with the context of the next thread scheduled to run
• The function takes two input parameters - addresses of the ETHREADs being swapped
• We can intercept these with an inline hook to monitor system activity
SwapContext hook

Hiding: The inline hook can be detected and removed by the attacker. Alternatively, SwapContext could be bypassed with a modified scheduler
Other process detection methods

**PspCidTable** - Contains references to all processes and threads in the system. (Cid = Pid | Tid)

**CSRSS handles** - The Win32 subsystem process csrss.exe contains a handle to each running process and thread (except System, smss.exe and itself)

**HandleTableList** - Handle tables of each process are linked in HandleTableList. (Similar principle to ActiveProcessLinks)
Other process detection methods

PsSetCreateProcessNotifyRoutine - “Official” callback routine triggered by process creation/termination

sysenter hook - The sysenter instruction calls the system call handler when a Native API is called from User Mode (transition User Mode → Kernel Mode). The handler address is stored in the MSR register SYSENTER_EIP_MSR (0x176) and can be hooked.

others - Many other methods...(e.g. lame OpenProcess “bruteforce”)
Only processes in hiding?

Other contraband that rootkits try to hide:
  • files
  • Registry keys
  • loaded drivers / system modules
  • network connections
  • etc...

The approaches are similar to process hiding (hooking, DKOM)
POCs vs. Rootkits in the wild
POCs

- The process hiding era probably began with Hoglund's **NTRootkit** (2001)
  - SSDT hook of NtQuerySystemInformation introduced

- The **FU rootkit** used DKOM (unlinked EPROCESS structures from the list)
  - Successor FUTO removed handles from PspCidTable

- Other contestants in the Hide & Seek game: phide, phide_ex, Rkdemo, ZomBiE, etc.
Arms race or a real war?

• Many POCs have served as inspirations for malware, but...
• **SSDT hooking** is ‘popular’ in malware as well as in legitimate software

• However most ‘ambitious’ malware chose a different approach to stealth
  • *Primitive*, but sufficient and less suspicious techniques
  • More **advanced** methods used by ‘top’ malware rootkits
Simple stealth

No process hiding...

• ‘Clever’ naming of malware files (misspelled ‘explorer’, ‘smss’, etc. e.g. iexplorer.exe)
• Folder choices: e.g. ‘WINDOWS\system’ instead of ‘WINDOWS\system32’
• Remote code injection
  • WriteProcessMemory, CreateRemoteThread
  • WriteProcessMemory, SetThreadContext
  • Create a legitimate process (CreateProcess) with malicious shellcode as its argument and run it
  • When injecting into a trusted process - both stealth and firewall bypassing
Mebroot
Mebroot’s inspirations

• Back in the **MS-DOS days** viruses used the Master Boot Record (MBR) for startup
• About 20 years later, Derek Soeder and Ryan Permeh resurrected the principle in the **eEye BootRoot** project (2005)
• **NVLabs Vbootkit** (2007) by Nitin and Vipin Kumar was able to defeat the updated protections in Vista
  • **Vbootkit 2.0** (2009) could subvert Windows 7
• **Stoned Bootkit** by Peter Kleissner (2009)
Mebroot itself

• First variants began appearing in late 2007 and several upgraded variants have appeared since then

• Downloads and ‘supports' banking info stealing malware Sinowal
  • The ‘duo' is responsible for stealing over half a million credentials/online banking/credit card info

• Most distinguishing aspect - loading via infected MBR

• Stealth techniques
The Mebroot boot process

**Infected MBR**
- allocated @ 0x7c00
- reserve 2 KB from the end of conventional memory
- load sectors 60 and 61 from disk
- hook int 13
- load original MBR from sector 62

**Clean MBR**
- regular boot

**Boot sector (NTFS)**
- load NTLDI using hooked int 13

**OSLOADER (hooked)**
- part of NTLDI
- call <Kernel Patcher>

**ntoskrnl.exe (hooked)**
- call <Payload Loader>

**Kernel Patcher**
- intercept sector read and sector extended read operations
- search for specific signature in OSLOADER
- when found, write call <Kernel Patcher>
- overwrite it with call <Payload Loader>

**Payload Loader**
- call IoInitSystem
- load Mebroot driver from end of disk
- run DriverEntry
- return to kernel
Mebroot’s stealth

• No hiding of processes, Registry keys, or files ...
  • ...because it doesn't use them

• Rootkit code stored on physical sectors of the disk
  • This needs to be hidden

• Network communication is designed to stay under the radar
  • Operates in the NDIS layer
Tug of war with the AV industry

• First versions hooked IRP functions in disk.sys
  • IRP_MJ_READ hooked to return a clean version of the MBR instead of the infected one, and zeroes instead of the rootkit’s malicious code
  • IRP_MJ_WRITE hooked for protection of the malicious code

Detection: Compare the IRP functions to the expected values in ClassPnp.sys (ClassReadWrite)
Figure: Mebroot's hooks of disk.sys IRP functions
Mebroot’s improvements

• As a response, Mebroot hooked all IRP functions of disk.sys, modified the original pointers to ClassReadWrite in ClassPnp.sys (and the CdRom0 driver IRP functions) to defeat the detection

• Hooking lower level drivers: atapi.sys, acpi.sys, vmscsi.sys

• Installing and uninstalling the hook when necessary (when the disk object's handle is opened)

• Change the \Device\Harddisk0\DR0 object type to a copy of the 'Device' type with a patched ParseProcedure

• Self-defense mechanisms, reinfecting the MBR on Windows shutdown,…
Detecting Mebroot

Where to look?
• Master Boot Record
• Interrupt Descriptor Table
• 'DOS' memory
• signature-based detection
Rustock
Rustock characteristics

• Versions A, B, C, ...
• Used as a spam botnet, but its architecture enables it to do anything (password stealing, DDoS, etc.)
• Advanced polymorphic protector, hardware based and customized encryption before infection
• Two components:
  • DLL - responsible for spam distribution and botnet communication
  • rootkit - responsible for stealth
Rustock’s stealth

• Again no files, processes, Registry entries
• Virus - infects various system drivers
  • 'travels' from driver to driver - disinfecting one and infecting another
• This needs to be hidden
Rustock's hooks

- Driver's infection is camouflaged by File System hooks (e.g. ntfs.sys)
- The original driver bytes are returned instead
- Instead of hooking the driver's IRP dispatch functions, inline hooks are used in the driver's code
- Attempt to hide the inline hooks by placing them after garbage instructions at the beginning of the function
- Firewall bypassing achieved by hooking network drivers (tcpip.sys, ndis.sys, wanarp.sys)
Hiding the DLL

• The **DLL** component is **injected** into winlogon.exe (or services.exe on Windows Vista)
• It's **hidden** and protected by **hooking** a few **Native API** functions
• This was done by hooking **KiFastCallEntry**
  • (called by sysenter – SYSENTER_EIP_MSR (0x176) stores the address of KiFastCallEntry, which dispatches system service calls)
• Newer version patches **each thread's pointer to the SSDT** (KTHREAD → ServiceTable) to copy modified by the rootkit
  • This way the SSDT stays clean
• The DLL module is also removed from the process' **PEB LDR list**
Rustock’s self-defense

- Anti-debugging techniques
  - check `KdDebuggerEnabled`
  - clearing hardware breakpoints (Debug Registers)
  - checksums of its code

- Registers callback function (KeRegisterBugCheckCallback) to clear rootkit memory during a BSOD
Rustock's protector

• Multi-layer, heavy obfuscation

• Compressed with aPlib and encrypted with RC4
• Encryption key is hardware related - a unique copy is created upon infection
  • Usually doesn't run on different systems
Rustock detection

• Infected driver is a weakness

• Can be read if we read at a ‘low enough’ level

• Injected DLL can be detected rather easily, since we know how it's hidden
Olmarik alias TDL

I ar hiding

u kant c mee
Meet Olmarik

• Development: TDL1 (summer 2008), TDL2 (summer 2009), TDL3 (fall 2009), ...

• Tries to block AVs, backdoor, botnet functionality, downloader
  • Done by DLL components
• Olmarik acts as a hotel - haven for 3rdparty malware

• Youngest member of the rootkit family - inspirations from the ‘elders’
Installation tricks

- Installation by registering itself as a Printer Processor (AddPrintProcessor API)

- Obfuscation, anti-debugging techniques
  - Hooking an API function in its own IAT during unpacking
Infecting – the Olmarik way

- Loading on OS startup by infecting a storage miniport driver (e.g. atapi.sys)
  - Only a small loader stub is written in the .rsrc section of the driver
  - DriverEntry is redirected to this code
  - Rootkit body is loaded using a callback function (FS not yet accessible)
  - File size of the driver remains unchanged after infection
- Low level disk access using SCSI request
- “TDL3” string marks beginning of rootkit's sectors at the end of disk
Olmarik's File System

- TDL3 implements its **own hidden** (and RC4 encrypted) **file system** for storing its own files and downloaded malware

- Files:
  - `tdlcmd.dll, tdlswp.dll` - User Mode components injected into other processes
  - `config.ini` - configuration file - encryption keys, ID of infected machine, ...
  - `tdl` - rootkit body
  - `rsr.dat` - backup of original bytes of infected driver
Olmarik's improvements

• The true contents of malicious disk sectors were hidden using hooks of miniport driver's IRP routines
• Newer versions no longer hook IRP routines, but own the device object of the infected drive
  • AttachedDeviceObject → DriverObject = OlmarikDriverObject
• Authors also gave up the “TDL3” signature (slightly)
• Ability to infect various drivers, not only atapi.sys
Olmarik's sense of humor

• Unconventional **error codes**: STATUS_SECRET_TOO_LONG, STATUS_TOO_MANY_SECRETS

• Funny **debug messages**:

  **Fight Club**: I felt like putting a bullet between the eyes of every panda that wouldn't screw to save it's own species.

  **Homer Simpson**: Alright Brain, you don't like me, and I don't like you. But let's just do this and I can get back to killing you with beer.

  I'm normally not a praying man, but if you're up there, please save me, Superman
Olmarik’s sense of humor

**Forrest Gump**: Run, Forest! Run!
- When a removal attempt is detected by the self-defense thread

**Darth Vader**: Your powers are weak, old man!
- When changes by a cleaner are restored
Statistics

Figure: Relative detection shares of Mebroot, Rustock, Olmarik and other rootkits
Statistics

Figure: Relative detection shares of Mebroot, Rustock and Olmarik
Rootkits – What’s next?

• Tremendous evolvement of rootkit technology

• Advanced version of Hide & Seek

• The most technically challenging types of malware

• Low share among other malware - will this change?
Thank you for your attention...

Robert Lipovsky
lipovsky@eset.sk