FINDING EVIL IN MEMORY

Michael J. Graven
michael.graven@mandiant.com

I work at MANDIANT

- Targeted threat experts
  - Blamed for “APT” term ... sorry.
- Offices in DC, NY, LA, SF
- About 130 employees
  - About half carry Top Secret clearances
- Typical customers
  - 23 of Fortune 100
  - 69 of Fortune 500
  - 8 of 10 largest defense contractors
  - 3 of the largest banks

So do these guys

PETER SILBERMAN
- Researcher and coder
- Vulnerabilities and memory forensics
- Co-author of Rootkits book

JAMIE BUTLER
- Director of R&D in our memory forensics
- Certified Tough Guy
- Director of R&D in our product division
- Company vodka expert

Things IR nerds need

MEMORY ANALYSIS
- Driver enumeration
- Handles and sections
- Per-process memory space acquisition
- Kernel hook enumeration
- Injection detection
- Network connections

OTHER USEFUL STUFF
- Web history
  - URLs, cookies, forms, files
- Prefetch file parsing
- Raw or API access to disk, memory, registry
- ARP, DNS cache, route table
- Stealth

Things for super-geniuses

PE & DRIVER ANALYSIS
- Imports, exports
- Entropy calculations
- Anomaly detection
  - JMP detours
  - Past-EOF data
- PE checksums
- Compiler/packer signatures
- Signature validation
- Resources, strings

SCALED IR OPERATIONS
- Indicators of Compromise
  - Enable large-scale ops
- Reduce false positives*
Why is memory important?

Dammit, I can’t remember.

Malware In Memory

- Looking in memory means examining < 8GB
- Looking on disk can mean examining 2000 GB or more
- Malware must be in memory to run
  - userland or kernel land
- Incident Responder’s job is to identify, assess and remediate malware

Malware in Memory

- Examining memory for malware offers the following benefits:
  - Not worried about system call modification
  - Less data to sort through
  - Looking at malware in memory is the equivalent of being a peeping tom
  - You get to see the malware in its unprotected/unpacked or... (undressed) state

Malware In Memory

- To detect malware in memory
  - Look for the code
  - An injected DLL
  - A rogue driver
  - An allocated section of memory with binary code
  - Look at what is executing
    - Malicious process
    - Rogue thread
  - Look for entry points that achieve execution in the form of hooks
    - Data hooks or DKOM hooks
    - Inline hooks

Types of Malware

- Rootkits
- Backdoors
- Worms
- Information Stealers
  - Password Stealers
  - Keyloggers

Rootkits

- Used to hide presence/persistence on a machine
- Varying levels of complexity
  - Attackers should be embarrassed:
    - Userland Hooks
  - Easy:
    - SSDT index hooks
  - Moderate:
    - SSDT inline hooks
  - Hard:
    - IRP Hooks
    - DKOM hooks on processes
    - IDT hooks
  - State Funded:
    - DKOM attacks on TCP stacks
    - Attacking Memory Manager
Hiding Persistence/Presence

- Rootkits commonly are used to hide presence of attackers:
  - Disk
  - Registry
  - Processes
- Most commonly these techniques are old and easily detectable
  - SSDT hooks
- Remember: Rootkits don’t compromise a host, stupid developers and users do.

Rootkits: DKOM

I diddle with the kernel

FU

- Using the DKOM Method to hide a process
  1. Locate the EPROCESS block of the process to hide
  2. Change the process behind it to point to the process after the process you are hiding
  3. Change the process after it to point to the process before the one you are hiding

- Task Manager and other programs use the linked list of processes (beginning at PsActiveProcessHead) to enumerate processes. DKOM disconnects processes from the list, which makes them hidden.

FUTo

- FUTo rootkit
  - Uninformed Journal Vol. 3 (http://www.uninformed.org)
  - New version of FU hence the ‘To’
  - Hides from detection tools that used process and thread handles as a way to find hidden processes.
Hidden Processes: FUTo

- Unhooks processes from the linked list of processes.
- Manipulates the PspCidTable
  - Job of PspCidTable is to keep track of all the processes and threads
  - The PspCidTable’s indexes are the PIDs of processes.
  - In the PspCidTable, the PID is the index into the table (handle number) of the EPROCESS with the same PID.
- Problems with using only the PspCidTable:
  - Relying on a single data structure is not a good idea
  - By altering one data structure, the OS has no idea a hidden process exists and cannot open the process.

FUTo

```
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, 132);
if(hProcess == INVALID_HANDLE)
    return 0;

NTTerminateProcess:
    PVOID obj = TranslateHandleToObject(hProcess);
    TranslateHandleToObject Process = PspCidTable[PsGetCurrentProcess()];
    if(Process == NULL) return 0;
    return Process->ObjectTable[hProcess];

ZwTerminateProcess(hProcess);
```

FUTo

- The future of process hiding should involve no process at all.
- Remember “Anything you can do [in userland], I can do better [in the kernel] with enough effort.”
- We are seeing a trend in rootkits creating system threads and obscuring the these threads.
  - Unreal

Rootkits: What to look for

Because binaries don’t speak human

The problem with hooking

- The problem identifying malicious hooks:
  - Windows hooks its own drivers
  - HIPS hooks everything
  - Both look malicious
- We will try to give you steps to identify what is what
Rootkits: What to look for (IRPs)

- Malicious IRP Hook identification:
  1. Is the driver NTFS, DISK, FAT, TCPIP, NDIS, KBDCLASS?
    ▪ These are all drivers an attacker might target
  2. No hooking module identified
    ▪ Indicates the hook points to pool memory and not a driver
  3. Look for a single hook
    IRP_MJ_DEVICE_CONTROL hook
  4. Is hooking driver signed?
  5. Google is your friend!

Rootkits: What to look for (IRPs)

Notice the hooked function

Notice the hooked module is TCPIP.sys

Notice that disk.sys has almost all its IRP handlers hooked by CLASSPNP.sys. This indicates a Windows hook.

Rootkits: What to look for (SSDT)

- SSDT hooks are the most commonly made by both attackers and HIPS
  ▪ Differentiating between the two is not easy
- Identifying Malicious SSDT hooks:
  1. Hooking module has no resolved drivers
  2. Only a few hooks in place
  3. Hooking driver is unsigned (only valid on 2k-Vista x86)
  4. Google is your friend!
Rootkits: What to look for (SSDT)

Suspicious driver name?

Rootkits: What to look for (IDT)

- IDT hooks are the holy grail for attackers
  - Uncommon and rarely seen
  - e-mail us if you find one 😊
- IDT hooks allows attackers to subvert things like:
  - Memory manager
  - Keyboard events
  - System calls (on older OS)
- Identifying suspicious IDT hooks is easy:
  1. Any IDT hooks is suspicious

Backdoors

Backdoor: Goals

- Used to maintain access to a computer
  - Some are stealthy some are garbage
- Remote access
  - Outbound/inbound connections
- Persistence
  - Backdoor needs to start up on reboot
- Exfiltrate/Infiltrate
  - Ability to download/upload/execute files

Worms

Worms: Goals

- Difference between Worms and Backdoors is propagation
  - Worms will spread
    - Infect removable media
    - Auto scan and exploit
  - Backdoors will be spread by attackers
- Otherwise goals are usually similar
Backdoors/Worms: What to look for

- Double infection prevention
  - Malware may infect the same machine multiple times
  - (same) Dumb user
  - (same) Unpatched box
- Multiple instances of the same malware can make the system unstable
  - An unstable system defeats the main purpose of infection
- Mutants/Events are commonly used to identify already infected systems

Worms: What to look for

- Top worm infections according to Virus List
  - Feb 2010: 1. Conficker 2. Salty.AA
  - May 2009: 1. Conficker

Backdoors/Worms: What to look for

- Identifying suspicious mutants:
  1. Practice, look at uninfected windows machines
  2. Practice, look at infected windows machines
  3. Review malware behaviors
  4. Use Redline’s handle trust feature

Backdoors/Worms: What to look for

- Virus Name | Mutex

<table>
<thead>
<tr>
<th>Virus Name</th>
<th>Mutex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kidd.Haka Conficker</td>
<td>.-x and .-99 and [randomascii]</td>
</tr>
<tr>
<td>Salty AA</td>
<td>OpmuteX9</td>
</tr>
<tr>
<td>Flystd ??</td>
<td>Hacker.com.cn_MUTEX</td>
</tr>
<tr>
<td>NetSky</td>
<td>&quot;D'r'o'l'p'q'e'd'S'k'y'N'e't'</td>
</tr>
<tr>
<td></td>
<td>_-00}X</td>
</tr>
<tr>
<td></td>
<td>YY999nFY</td>
</tr>
<tr>
<td></td>
<td><strong><strong>&gt;&gt;vstructor&lt;</strong></strong></td>
</tr>
<tr>
<td>Salty W</td>
<td>u_joker_v1.06</td>
</tr>
<tr>
<td>Poison Ivy</td>
<td>jVlogA14</td>
</tr>
</tbody>
</table>

Information Stealers

- Suspicious open ports, and connections:
  - We’ll cover a ton more in the Behaviors section
**Info Stealers: Goals**

- Simple: To steal information
- Includes:
  - Keyloggers
  - Password stealers
  - Browser hijacking?
- Covertly exfiltrate stolen information
  - Log information:
    - (to) memory send out over network
    - ftp/http/https/custom
    - (to) disk
    - Remotely download
    - Auto e-mail after a certain size

**Info Stealers: What to look for**

- Info Stealers need to do two things:
  - Steal keystrokes
  - (potentially) log key strokes
- In userland/kernel there are a finite number of ways to steal keystrokes

**Info Stealers: What to look for**

METHOD 1
- SetWindowsHookEx
- CallNextHookEx
- GetKeyNameText*
- GetAsyncKeyState*
- GetKeyboardState*
- * not required API call to function.

METHOD 2
- GetAsyncKeyState
- GetKeyState*
- GetProcessWindowStation*
- OpenWindowStation*
- SetProcessWindowStation*
- OpenInputDesktop*
- SetThreadDesktop*

**Info Stealers: What to look for**

- Most info stealers “log” to some place…

<table>
<thead>
<tr>
<th>Keylogger Name</th>
<th>Process</th>
<th>Log File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klog</td>
<td>System</td>
<td>Klog.txt</td>
</tr>
</tbody>
</table>
| Advanced Keylogger | Explorer | \WIN\Help\dsclientsoxx\help | \WIN\Help\dsclientsoxx\help
| Spector Pro    | Explorer | A645B2AF87DEACD5B3B222B22FEE1C6B5C.zup |

**Malware Behaviors**

Behaving badly was never so unsexy

**Malware Behavior Background**

- The term “Behavioral” has been destroyed by AV marketing
- Malware behavior is what the malware is doing
  - Most interested in how it looks when running in memory
    - Handles
    - Signed/unsigned binaries
    - Execution path
    - Executing user
**Malware Behaviors**

- Malware itself isn’t bad
  - A command shell isn’t malware
    - Guns don’t kill people, people kill people
- Maliciousness of software depends on
  - The software author’s intent
  - How the software is used
- Misuse creates anomalies on the system
  - cmd.exe is not supposed to be run by svchost.exe
  - This is bad behavior by a good piece of software

**Malware Behaviors**

- Finding unique or anomalous behavior can lead to fast triage of infected hosts
- APT malware appears “normal”
  - But still is identifiable through behavior

**Malware Behaviors**

- Process execution
  - Path
  - User
- Suspicious handles
- Suspicious imports/DLLs
- Unsigned binaries
- Argument Verification
- CMD Id
- Handle leaks

**Process Path Execution**

- Malware names itself after a system file:
  - svchost.exe
  - explorer.exe
  - iexplore.exe
- Processes are launched from some non-standard directory

**Process User Execution**

- Malware executes a process such as:
  - Svchost
  - The process is not launched normally
  - The owning user is a non standard user
  - Process replacement
  - Process injection

**Unsigned binaries**

- Malware isn’t usually digitally signed
- If malware uses file infection as persistence mechanism
  - Creates executables where a signature exists but is not verified
Suspicious Imports

- Imports tell us a very important story
  - Capabilities
  - File makeup
  - Intent

Suspicious Imports Ex #1

- What do you think when you see?
  - GetProcAddress
  - LoadLibrary

- What do you think when ALL you see is?
  - GetProcAddress
  - LoadLibrary

- Thoughts:
  - Probably packed
  - Unknown capabilities, but anything is possible

Suspicious DLLs

- Suspicious does not refer to name
- Certain DLLs offer known functionality to processes
  - Ws2_32.dll / msock32.dll – socket functionality
  - Wininet.dll – HTTP control
  - Netapi32.dll – Netbios

Suspicious DLLs

- What if:
  - calc.exe had ws2_32.dll loaded?
  - services.exe had wininet.dll loaded?
  - iexplore.exe has ws2_32.dll loaded?
  - These suspicious DLLs offer functionality to processes that don’t need it.
Suspicious Handles

- Malware needs to accomplish its *goals*
  - To do this they must access system resources
    - Spawn processes
    - Modify registry
    - Open files
    - Write files
    - Log files
    - Create mutants
    - Etc

Argument Verification

- Standard system executables are spawned with a finite “known” set of parameters:
  - svchost
- A deviation of known arguments can indicate a malicious copy of the process:

Process Unmap/Switch

- Increasingly common attack technique
  - Allows malware to masquerade as any process with a legit path
  1. Malware spawns any binary in a suspended state
  2. Malware frees originally allocated binary section
  3. Malware allocates and maps in a new binary section
  4. Malware acquires thread CONTEXT and changes EIP to point to new memory
  5. Malware resumes thread

Process Unmap/Switch

- Side Effects:
  - Process in suspended state doesn’t fully import required DLLs
  - Usually Memoryze will identify an injected memory section
  - Dependent on what portion of the binary gets mapped in
  - When malware frees’ executable it invalidates the VAD

CMD Id

- Malware will spawn processes from a command shell.
- Being able to identify when this occurs is valuable.

CMD Id

Notice Path is all lower case in process.
CMD Id

Notice Path is all upper case in one process

CMD Id

All lower case indicates the attacker typed out the full path to their executable. Seeing that or a parent of cmd.exe indicates a process was launched from a command shell.

Credit goes to Jesse Kornblum for this behavior

Resources
The bad guys have them, do you?

MANDIANT Redline
- Guided memory analysis
- Aimed at less-expert responder
- MRI calculations
  - Process-user match
  - Injection detection
  - Digital signature validation
  - DLL scoring
- Word report generation

MANDIANT Memoryze
- Free memory forensic analysis tool
- XML output
- Runs on live system or memory image
- Part of Redline
- Same tech as MIR

$MANDIANT Black Hat classes
- Incident Response
- Intro to Malware Analysis
- Advanced Malware Analysis
- Additional Topics in Advanced Incident Handling
  - Held at the Shadow Bar
Free resources

- Free tools
  - Redline
  - IOCe
  - Memoryze
  - Highlighter
  - Red Curtain
  - Web Historian
  - First Response

- Resources
  - M-trends
  - M-union
  - blog.mandiant.com

- Education
  - Black Hat classes
  - Custom classes

- Webinar series
  - State of the Hack
  - Fresh Prints of Malware

Books

- INCIDENT RESPONSE
- Computer Forensics
- Cyber Incident Response & Computer Forensics