dAnubis

Dynamic Device Driver Analysis based on Virtual Machine Introspection

Matthias Neugschwandtner
Paolo Milani Comparetti
Christian Platzer
Ulrich Bayer

International Secure Systems Lab
Vienna University of Technology
Motivation

• Enormous number of new malware samples each day requires automated analysis

• Malware needs kernel-mode privileges to provide powerful functionality (e.g. Rootkits)
  – Stealth
  – Information gathering

• Aspect of device driver behavior has received less attention
Outline

• Overview
• \textit{dAnubis}
• Evaluation
• Conclusion
Our Approach

- Run malware in an emulated environment
- Monitor and evaluate analysis events
  - Executed code
  - Manipulated memory
- Reconstruct high-level semantics
- Generate human-readable analysis report
Process vs. Driver

<table>
<thead>
<tr>
<th></th>
<th>Process</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution context</strong></td>
<td>separate for each process</td>
<td>same as kernel</td>
</tr>
<tr>
<td><strong>View on memory</strong></td>
<td>unique page table directory</td>
<td>kernel memory space</td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td>limited through well-defined system call interface</td>
<td>unlimited possibilities</td>
</tr>
</tbody>
</table>

- **OS Kernel**
  - Process
  - Driver
  - System call

DIMVA 2010 – 7th Conference on Detection of Intrusions and Malware & Vulnerability Assessment
Rootkit: Goals

Inject code into the kernel
- Use a kernel exploit
- Load a device driver

Communicate with user mode application
- Use device communication

Provide stealth

Gather sensitive information
- Reroute kernel control flow
- Tamper with kernel data structures
Scenario: Process hiding

Application issues system call to list processes

Kernel system call dispatcher looks up system call in call table

Kernel calls NtEnumerateProcess

NtEnumerateProcess retrieves information from double-linked kernel process list

Information is returned to application

replace entry call table hook

apply runtime patch

direct kernel object manipulation
Scenario: File hiding

Application issues system call to list files in a directory

Kernel system call dispatcher looks up system call in call table

Kernel calls NtQueryDirectoryFile

NtQueryDirectoryFile requests information from the disk device

Information is returned to application

tamper with the device communication
Overview

- Dynamic analysis of Windows device drivers
- Virtual machine introspection using Qemu
- Derive high-level semantics of observed analysis events
- Provide driver context information for observed analysis events
- Perform first large scale study of kernel malware behavior
dAnubis
dAnubis

- Detect introduction of unknown device drivers
- Keep track of driver state
- Analyse
  - Generic behavioral aspects (e.g. called kernel functions)
  - Known typical Rootkit behavior (e.g. system call table hooks)
  - Licit OS - device driver interaction (e.g. device communication)
Challenges

• Bridging the semantic gap
  – Loss of semantic information when looking at memory from outside
  – Reestablish information by guest-view casting
  – Obtain necessary information from debugging symbols

• Tracking and attributing kernel-mode events
  – Code runs in arbitrary context
  – Identify event origin based on current program counter
  – Exact location of driver codebase in memory has to be known
Device Driver Analysis

• Intercept lowest-level loading mechanisms
  – Get codebase location
  – Get offsets of the driver’s exported functions

• Perform state-tracking
  – Assign analysis events to a driver’s context
Device Driver Analysis

- Monitor driver activity
  - Record calls to exported Windows kernel functions
  - Taint string occurrences in the driver image
- Monitor driver communication
  - Creation of devices and attaching to driver stacks
  - Intercepting IRP traffic
Device Driver Interaction

- Devices as communication endpoints
- Stacking of drivers for complex processing

Diagram:

- Application (User mode)
  - Read(device, ...)
  - Write(device, ...)
  - DeviceIoControl
- I/O System
- Driver
  - ReadDispatch
  - WriteDispatch
  - IoctlDispatch
  - ...
Memory Analysis

• Put certain memory regions under supervision
  – Kernel objects for process and driver bookkeeping
  – System call table
  – Kernel module codebases

• Track down and evaluate manipulations
  – Targets of call table manipulations
  – Consequences of kernel object manipulation
  – Detour patches of existing kernel code in memory
    • Determine which kernel function has been patched
Stimulation

• Rootkit functionality depends on external stimuli, e.g.
  – Keystrokes for keylogging
  – Process enumeration for process-hiding
• Stimulator component in the VM that repeatedly issues API calls
Evaluation

• Small evaluation with samples from www.rootkit.com to verify functionality
• Large-scale study
  – 64733 samples analyzed by Anubis in August 2009
  – 463 of these called NtLoadDeviceDriver
## Results: High-level Activity

<table>
<thead>
<tr>
<th>Driver activity</th>
<th>Number of samples exhibiting behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device driver loaded</td>
<td>463</td>
</tr>
<tr>
<td>Windows kernel functions used</td>
<td>360</td>
</tr>
<tr>
<td>Windows device IO used</td>
<td>339</td>
</tr>
<tr>
<td>Strings accessed</td>
<td>300</td>
</tr>
<tr>
<td>Kernel code patched</td>
<td>76</td>
</tr>
<tr>
<td>Kernel call tables manipulated</td>
<td>37</td>
</tr>
<tr>
<td>MDL allocated</td>
<td>34</td>
</tr>
<tr>
<td>Kernel object manipulated</td>
<td>3</td>
</tr>
</tbody>
</table>
# Results: Device Activity

<table>
<thead>
<tr>
<th>Device activity</th>
<th>Number of samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device created</td>
<td>339</td>
</tr>
<tr>
<td>Driver’s device accessed from user mode</td>
<td>110</td>
</tr>
<tr>
<td>Driver’s device invoked from user mode</td>
<td>86</td>
</tr>
<tr>
<td>Strings detected during communication</td>
<td>24</td>
</tr>
<tr>
<td>Attaches to device stack</td>
<td>2</td>
</tr>
<tr>
<td>Registers completion routine</td>
<td>2</td>
</tr>
</tbody>
</table>
Results: Stealth

![Chart showing stealth results for various categories: Registry, File, Process, Driver, Network port. The Registry category has the highest value. The chart includes categories: Filter Driver, IRP Hook, DKOM, Runtime Patching, SSDT Hook.]
## Example Report A

<table>
<thead>
<tr>
<th>Driver name</th>
<th>syssrv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created devices</td>
<td>Device\MyDriver</td>
</tr>
<tr>
<td>Rootkit activity</td>
<td>NtEnumerateKey SSDT Hook</td>
</tr>
<tr>
<td></td>
<td>NtQueryDirectoryFile SSDT Hook</td>
</tr>
<tr>
<td></td>
<td>svchost.exe DKOM process hidden</td>
</tr>
<tr>
<td>Invoked major functions</td>
<td>CREATE 5x from user mode</td>
</tr>
<tr>
<td></td>
<td>DEVICE_CONTROL 5x from user mode</td>
</tr>
<tr>
<td></td>
<td>CLOSE 5x from kernel mode</td>
</tr>
<tr>
<td>Used strings</td>
<td>\WINDOWS\system32\mssrv32.exe in DEVICE_CONTROL IRP</td>
</tr>
<tr>
<td></td>
<td>\SOFTWARE...\CurrentVersion\Run\mssrv32 in DEVICE_CONTROL IRP</td>
</tr>
<tr>
<td>Used kernel functions</td>
<td>ObReferenceObjectByName during DEVICE_CONTROL</td>
</tr>
<tr>
<td></td>
<td>PsLookupProcessByProcessID during DEVICE_CONTROL</td>
</tr>
<tr>
<td></td>
<td>NtEnumerateKey during NtEnumerateKey hook</td>
</tr>
<tr>
<td></td>
<td>wcslen, wcspz, wcscat during NtEnumerateKey hook</td>
</tr>
<tr>
<td></td>
<td>NtQueryDirectoryFile during NtQueryDirectoryFile hook</td>
</tr>
</tbody>
</table>
## Example Report B

<table>
<thead>
<tr>
<th>Driver name</th>
<th>FILEMON701</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created devices</td>
<td>\Device\Filemon701</td>
</tr>
<tr>
<td></td>
<td>two unnamed devices</td>
</tr>
<tr>
<td>Attached to devices</td>
<td>sr</td>
</tr>
<tr>
<td></td>
<td>MRxSMB</td>
</tr>
<tr>
<td>Completion routine</td>
<td>QUERY_VOLUME_INFORMATION for device sr</td>
</tr>
<tr>
<td>Invoked major functions</td>
<td>CREATE from user mode</td>
</tr>
<tr>
<td></td>
<td>QUERY_VOLUME_INFORMATION from kernel mode</td>
</tr>
<tr>
<td></td>
<td>READ from kernel mode</td>
</tr>
<tr>
<td></td>
<td>CLEANUP, CLOSE from kernel mode</td>
</tr>
<tr>
<td></td>
<td>FastIoDeviceControl</td>
</tr>
<tr>
<td>Used kernel functions</td>
<td>IoCreateDevice during entry</td>
</tr>
<tr>
<td></td>
<td>IoGetCurrentProcess during entry</td>
</tr>
<tr>
<td></td>
<td>IoCreateDevice during FastIoControl</td>
</tr>
<tr>
<td></td>
<td>IoAttachDeviceByPointer during FastIoControl</td>
</tr>
</tbody>
</table>
Conclusions and Outlook

• *d*Anubis can provide a substantial amount of information on kernel-side malware

• Large scale analysis has given interesting first insight in the kernel-side malware landscape