NumChecker: A System Approach for Kernel Rootkit Detection and Identification

Xueyang Wang, Ph.D.
Xiaofei (Rex) Guo, Ph.D.
(xueyang.wang || xiaofei.rex.guo) *noSPAM* intel.com
We don’t speak for our employer. All the opinions and information here are our responsibility including mistakes and bad jokes.
Executive Summary

• Malware continues to proliferate
  — Increasing in number
  — Stealthier

• Traditional software-level detection mechanisms have limited effectiveness
  — Most of them relies on the correct functioning of OS
  — VMM-based approaches has semantic gap
  — Performance constraints

• A new solution: NumChecker
  — Analyzing software behaviors with rich hardware events
  — Low performance overhead
  — Focus on kernel rootkit
Agenda

• Kernel Rootkits
• Hardware Performance Counter
• NumChecker Design
• Kernel Rootkit Detection
• Kernel Rootkit Identification
• Conclusion
Kernel Rootkit

- **Rootkit**
  - Toolkits injected by attackers to hide malicious activities from the users and detection tools

- **Kernel Rootkit**
  - Rootkits that subvert the operating system kernel directly
  - Have unrestricted access to system resources
  - Used by attackers to hide their presence, open backdoors, gain root privilege, and disable defense mechanisms
Kernel Rootkit Behavior Classification

- Direct kernel object manipulation (DKOM)
  - Subvert the kernel by directly modifying data objects

- Kernel Object Hooking (KOH)
  - Hijack the kernel control-flow
  - A majority of Linux kernel rootkits persistently violate control-flow integrity
  - Hijack the kernel static control transfers (e.g., SucKIT rookit)
  - Hijack the kernel dynamic control transfers (e.g., Adore-ng)
Known Kernel Rootkit Detection Approaches

**Host-based rootkit detection**

- Run inside the target they are protecting
- Check kernel static and dynamic objects
Known Kernel Rootkit Detection Approaches

**Host-based rootkit detection**
- Run inside the target they are protecting
- Check kernel static and dynamic objects

Challenges:
-Detection tools themselves might be tampered with by advanced kernel rootkits, which have high privilege and can access the kernel memory
Known Kernel Rootkit Detection Approaches

Host-based rootkit detection
• Run inside the target they are protecting
• Check kernel static and dynamic objects

Challenges:
-Detection tools themselves might be tampered with by advanced kernel rootkits, which have high privilege and can access the kernel memory

Virtual Machine Monitor (VMM) based rootkit detection
• Run at the VMM level
• Check kernel static and dynamic objects
Known Kernel Rootkit Detection Approaches

**Host-based rootkit detection**
- Run inside the target they are protecting
- Check kernel static and dynamic objects

**Virtual Machine Monitor (VMM) based rootkit detection**
- Run at the VMM level
- Check kernel static and dynamic objects

**Challenges:**
- Detection tools themselves might be tampered with by advanced kernel rootkits, which have high privilege and can access the kernel memory
- "semantic gap" between the external and internal observation. The detection tools require detailed knowledge of the guest OS implementation
- Performance overhead
NumChecker: VMM-based kernel execution path checking using Hardware Performance Counters (HPCs)

- Runs at the VMM level
- Does not require detailed knowledge of the guest OS implementation
- Validates the execution path of guest system calls by checking the number of certain hardware events using HPCs

Hardware-Assisted Kernel Rootkit Detection

OS

VMM

Test program
Identifier

Guest VM

Process
Identifier

Run test programs in the guest VM

Test program

Identifier

Run test programs in the guest VM

Guest VM

Process
Identifier

Run test programs in the guest VM

Guest VM

Process
Identifier

Run test programs in the guest VM

Guest VM

Process
Identifier

Run test programs in the guest VM

Guest VM

Process
Identifier

Run test programs in the guest VM
Hardware Performance Counters (HPC)

- Performance monitoring unit (PMU)
  - Originally used for performance tuning
  - Performance counters
    - Intel Core i7 (11 counters per core)
    - AMD Quad-Core Opteron 1356 CPU (4 counters per core)
  - Event selectors
- Automatically count hardware events at the process level
- Typical events include clock cycles, instruction retirements, cache misses, TLB misses (100+ events)
- Details in the developer’s manuals
  - Intel® 64 and IA-32 Architectures Software Developer’s Manual
  - BIOS and Kernel Developer’s Guide (BKDG) for AMD Family 10h Processors
KVM in Linux

- Kernel-based virtual machine (KVM)
  - Based on Intel (VT) or AMD (SVM)
  - Guest mode and host mode
  - Each VM is an individual process
- KVM kernel module
  - Handles interception
- Linux Perf_event kernel service
  - Initializes, enables/disables, reads, and closes HPCs
NumChecker Design

- NumChecker kernel module
  - Communicates with Perf_event kernel service and KVM kernel
- Configuration program
  - Dynamically configure the events and syscalls to be monitored
- Log
  - HPC results are stored and compared with the reference model
Two-Phase Detection and Identification

- Clean guest
- Launch NumChecker
- Offline
- Checking flow
Two-Phase Detection and Identification

- Execute test programs
- Log in to the guest
- Launch NumChecker
- Count events of monitored syscalls
- Clean guest
- Offline

Checking flow

Host

Clean guest
Two-Phase Detection and Identification

- Execute test programs
- Log in to the guest
- Launch NumChecker
- Count events of monitored syscalls
- Log HPC results
- Clean guest

Checking flow:

Offline

Host
Two-Phase Detection and Identification

Checking flow:
- Clean guest
  - Execute test programs
  - Count events of monitored syscalls
  - Log HPC results
- Log in to the guest
- Host
- Monitored guest
  - Launch NumChecker
  - Count events of monitored syscalls
  - Execute test programs
- Offline
- Online
Two-Phase Detection and Identification

- Execute test programs
- Log in to the guest
- Launch NumChecker
- Count events of monitored syscalls
- Log HPC results
- Analysis
- Execute test programs

Checking flow:

Clean guest

Host

Monitored guest

Offline

Online
Two-Phase Detection and Identification

**Checking flow**

- **Clean guest**
  - Execute test programs
  - Log in to the guest
  - Count events of monitored syscalls
  - Log HPC results

- **Host**

- **Monitored guest**
  - Send the request for checking
  - Execute test programs
  - Count events of monitored syscalls
  - Log HPC results
  - Compare

**Offline**

**Online**
Syscall Measurement

- Guest user
- Test program
- INT 0x80 (syscall A)
- Guest kernel
- trap
- Host kernel
- Initialize HPCs
- Host kernel
- HPCs

Time line
Syscall Measurement

Guest user
- Test program
- INT 0x80 (syscall A)

Guest kernel
- trap
- Run monitored syscall A

Host kernel
- Initialize HPCs

HPCs
- Count events

Time line
Syscall Measurement

- **Test program**
  - **INT 0x80 (syscall A)**
  - **Run monitored syscall A**
  - **IRET (syscall A)**
  - **Disable HPCs**
  - **Test program**

- **Guest user**
- **Guest kernel**
- **Host kernel**
- **HPCs**

- **Initialize HPCs**
- **Read HPCs**
- **Count events**

Time line
Syscall Measurement

- Test program
- INT 0x80 (syscall A)
- Run monitored syscall A
- Other interception
- IRET (syscall A)
- Disable/enable HPCs
- Count events
- Disable HPCs
- Read HPCs
- Time line

Guest user

Guest kernel

Host kernel

HPCs
Kernel Preemption Handling

Guest kernel
- Run syscall A
  - Task switch
    - Disable HPCs
      - Count events

Host kernel
- Time line

HPCs
Kernel Preemption Handling

Guest kernel

- Run syscall A
- Task switch
- Run syscall B invoked by another program

Host kernel

- Disable HPCs
- Count events

HPCs

Time line
Kernel Preemption Handling

- **Guest kernel**: Run syscall A → Task switch → Run syscall B invoked by another program → Task switch → Run syscall A

- **Host kernel**: Disable HPCs → Enable HPCs

- **HPCs**: Count events

Time line
Detection: Test Programs

- Select preamble system calls to allow VMM to identify the process
- Ensures that we control the system call execution with selected arguments
- A sequence of selected system calls for measurement
Detection: Choosing Proper Events

• Events that occur frequently during the syscall
• Events that are statistically more stable in the presence of noises
• Events selected
  — UOPS: retired micro-ops
  — INST: retired instructions
  — NRET: retired near returns
  — BRAN: retired branch instructions
  — BRNT: retired branch taken instructions
Detection: Deviation Threshold

• Deviation
  — Event: $E_x$, system call: $S_y$
  — Count: $C(E_x, S_y)$

$$D_{test}(x, y) = \frac{C_{test}(E_x, S_y) - C_{ref}(E_x, S_y)}{C_{ref}(E_x, S_y)}$$

• Deviation threshold
  — Pick the threshold with the least false positive rate
  — HPC deviations is smaller than 5%
  — If the deviation exceeds 5%, malicious modifications are suggested
## Detection: Kernel Rootkits Detected

Detection capabilities. The numbers are deviations (%) from uninfected executions. Any deviation of more than 5% suggests a malicious modification.

<table>
<thead>
<tr>
<th>Guest OS</th>
<th>Rootkit</th>
<th>Events counted</th>
<th>System calls monitored</th>
<th>Detected?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>sys_open</strong></td>
<td><strong>sys_close</strong></td>
</tr>
<tr>
<td>Linux 3.0</td>
<td>Matias</td>
<td>UOPS  0.0</td>
<td>0.0</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST  0.0</td>
<td>0.0</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET  0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN  0.0</td>
<td>-0.7</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT  0.6</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Linux 3.0</td>
<td>Suterosu</td>
<td>UOPS  0.0</td>
<td>0.0</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST  0.0</td>
<td>0.0</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET  0.0</td>
<td>2.4</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN  0.2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT  0.6</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Linux 3.0</td>
<td>KBeast</td>
<td>UOPS  24.8</td>
<td>-0.1</td>
<td>129.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST  13.5</td>
<td>0.0</td>
<td>72.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET  9.9</td>
<td>0.0</td>
<td>56.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN  12.7</td>
<td>0.0</td>
<td>86.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT  12.0</td>
<td>0.0</td>
<td>82.1</td>
</tr>
</tbody>
</table>
### Detection: Kernel Rootkits Detected

<table>
<thead>
<tr>
<th>Guest OS</th>
<th>Rootkit</th>
<th>Events counted</th>
<th>System calls monitored</th>
<th>Detected?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>sys_open</td>
<td>sys_close</td>
</tr>
<tr>
<td>Linux 2.6</td>
<td>Enyelkm 1.1</td>
<td>UOPS</td>
<td>0.2</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>0.8</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>4.0</td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>1.7</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>0.8</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>Phalanx b6</td>
<td>UOPS</td>
<td>5.6</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>8.5</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>14.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>19.5</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>19.8</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>Sebek 3.2</td>
<td>UOPS</td>
<td>0.7</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>9.4</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>8.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>13.8</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>10.3</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>Adore-ng</td>
<td>UOPS</td>
<td>-10.7</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>0.0</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>-1.2</td>
<td>1.0</td>
</tr>
</tbody>
</table>
## Detection: Kernel Rootkits Detected

<table>
<thead>
<tr>
<th>Guest OS</th>
<th>Rootkit</th>
<th>Events counted</th>
<th>System calls monitored</th>
<th>Detected?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>sys_open</td>
<td>sys_close</td>
</tr>
<tr>
<td>Linux 2.4</td>
<td>SucKIT 1.3b</td>
<td>UOPS</td>
<td>923.9</td>
<td>13.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>836.1</td>
<td>8.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>676.5</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>1294.2</td>
<td>72.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>1125.6</td>
<td>21.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UOPS</td>
<td>175.8</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>99.4</td>
<td>10.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>123.5</td>
<td>25.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>119.9</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>119.2</td>
<td>9.1</td>
</tr>
<tr>
<td>Linux 2.4</td>
<td>Adore 0.42</td>
<td>UOPS</td>
<td>384.2</td>
<td>22.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>363.4</td>
<td>52.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>488.2</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>359.2</td>
<td>128.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>365.6</td>
<td>27.3</td>
</tr>
<tr>
<td>Linux 2.4</td>
<td>Sk2rc2</td>
<td>UOPS</td>
<td>955.4</td>
<td>13.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>827.8</td>
<td>10.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NRET</td>
<td>535.3</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRAN</td>
<td>1399.5</td>
<td>28.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BRNT</td>
<td>1071.2</td>
<td>21.2</td>
</tr>
</tbody>
</table>
Detection: Performance Evaluation

- **Test program execution time**
  - Each test program contains 500 iterations to repeatedly invoke the corresponding system call

- **Guest performance overhead**
  - Throughput degradation of the guest VM when NumChecker is invoked every 5 and 10 seconds

<table>
<thead>
<tr>
<th></th>
<th>Redhat 7.3</th>
<th>Fedora Core 4</th>
<th>Ubuntu 11.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test_open&amp;close</td>
<td>44.9 ms</td>
<td>52.7 ms</td>
<td>50.9 ms</td>
</tr>
<tr>
<td>Test_read</td>
<td>50.5 ms</td>
<td>69.1 ms</td>
<td>65.5 ms</td>
</tr>
<tr>
<td>Test_getdents64</td>
<td>61.0 ms</td>
<td>75.7 ms</td>
<td>69.3 ms</td>
</tr>
<tr>
<td>Test_stat64</td>
<td>27.2 ms</td>
<td>40.5 ms</td>
<td>20.3 ms</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>45.6 ms</td>
<td>59.5 ms</td>
<td>51.5 ms</td>
</tr>
</tbody>
</table>
- **HPC-based behavior signature**
  - Let $C(E_x, S_y)$ denote the count of event $x$ from the execution of system call $y$.
  - $m$ hardware events
  - $n$ system calls
  - an vector $V$ with $m \times n$ elements can be obtained:

$$V = [C(E_1, S_1), C(E_2, S_1), \ldots, C(E_m, S_1), C(E_1, S_2), C(E_2, S_2), \ldots, C(E_m, S_n)]$$
Identification: HPC-based Signature

The deviation of the element in the tested vector from the one in the reference vector is:

\[ D_{\text{test}}(x, y) = \left| \frac{C_{\text{test}}(E_x, S_y) - C_{\text{ref}}(E_x, S_y)}{C_{\text{ref}}(E_x, S_y)} \right| \]

\( D_{\text{test}} \) is calculated for each element in the tested vector and the largest one \( D_{\text{test\_max}} \) is determined:

\[ D_{\text{test\_max}} = \max_{1 \leq x \leq m, 1 \leq y \leq n} D_{\text{test}}(x, y) \]

Average deviation from the rootkit reference denoted as \( D_{\text{test\_avg}} \) and the Fitting Rate (FR) on the rootkit reference, which is defined as follows:

\[ \text{FR} = \frac{\text{no. of elements fitted to the targeted reference}}{\text{no. of elements in the tested vector}} \]
## Identification: Kernel Rootkits Identified

<table>
<thead>
<tr>
<th>Rootkit under test</th>
<th>SucKIT 1.3b</th>
<th>Adore 0.42</th>
<th>Sk2rc2</th>
<th>Superkit</th>
<th>Identified?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D_{test_max}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SucKIT 1.3b</td>
<td>3.80</td>
<td>538.49</td>
<td>592.73</td>
<td>38.28</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>1.70</td>
<td>111.40</td>
<td>115.65</td>
<td>4.95*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>8</td>
<td>12</td>
<td>84*</td>
<td></td>
</tr>
<tr>
<td>Adore 0.42</td>
<td>84.79</td>
<td>3.77</td>
<td>762.92</td>
<td>85.86</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>40.32</td>
<td>2.10</td>
<td>118.00</td>
<td>40.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>100</td>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Sk2rc2</td>
<td>710.34</td>
<td>168.00</td>
<td>3.71</td>
<td>85.46</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>127.78</td>
<td>69.32</td>
<td>1.91</td>
<td>42.76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Superkit</td>
<td>30.00</td>
<td>569.41</td>
<td>572.96</td>
<td>3.65</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>5.51*</td>
<td>111.39</td>
<td>114.88</td>
<td>1.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td>84*</td>
<td>12</td>
<td>12</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
## Identification: Kernel Rootkits Identified

<table>
<thead>
<tr>
<th>Rootkit under test</th>
<th>SucKIT 1.3b</th>
<th>Adore 0.42</th>
<th>Sk2rc2</th>
<th>Superkit</th>
<th>Identified?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D_{test_max}$</td>
<td>$D_{test_avg}$</td>
<td>FR</td>
<td>$D_{test_max}$</td>
<td>$D_{test_avg}$</td>
</tr>
<tr>
<td>SucKIT 1.3b</td>
<td>3.80</td>
<td>538.49</td>
<td>100</td>
<td>84*</td>
<td>38.28</td>
</tr>
<tr>
<td>Adore 0.42</td>
<td>84.79</td>
<td>3.77</td>
<td>8</td>
<td>84*</td>
<td>762.92</td>
</tr>
<tr>
<td>Sk2rc2</td>
<td>710.34</td>
<td>168.00</td>
<td>100</td>
<td>3.71</td>
<td>85.46</td>
</tr>
<tr>
<td>Superkit</td>
<td>30.00</td>
<td>569.41</td>
<td>100</td>
<td>3.65</td>
<td>572.96</td>
</tr>
</tbody>
</table>
Identification: Periodic Sampling

The system call being monitored

Time line

Guest

Host

HPC₁: Counting instruction retired

HPC₂: Counting event X

Read HPC₂

Read HPC₂

Read HPC₂

Read HPC₂

N inst

N inst

N inst

N inst
Identification: Periodic Sampling

sys_open()

sys_close()

sys_read()

sys_getdent()

sys_stat()
Identification: Periodic Sampling

- **W/O periodic sampling**

<table>
<thead>
<tr>
<th>Rootkit under test</th>
<th>SucKIT 1.3b</th>
<th>Superkit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D_{test\ _{max}}$</td>
<td>3.80</td>
</tr>
<tr>
<td>SucKIT 1.3b</td>
<td>$D_{test\ _{avg}}$</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>FR</td>
<td>100</td>
</tr>
<tr>
<td>Superkit</td>
<td>$D_{test\ _{max}}$</td>
<td>30.00</td>
</tr>
<tr>
<td></td>
<td>$D_{test\ _{avg}}$</td>
<td>5.51</td>
</tr>
<tr>
<td></td>
<td>FR</td>
<td>84</td>
</tr>
</tbody>
</table>

- **With periodic sampling**

<table>
<thead>
<tr>
<th>Rootkit under test</th>
<th>SucKIT 1.3b</th>
<th>Superkit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D_{test\ _{max}}$</td>
<td>3.90</td>
</tr>
<tr>
<td>SucKIT 1.3b</td>
<td>$D_{test\ _{avg}}$</td>
<td>1.35</td>
</tr>
<tr>
<td></td>
<td>FR</td>
<td>100</td>
</tr>
<tr>
<td>Superkit</td>
<td>$D_{test\ _{max}}$</td>
<td>75.00</td>
</tr>
<tr>
<td></td>
<td>$D_{test\ _{avg}}$</td>
<td>14.19</td>
</tr>
<tr>
<td></td>
<td>FR</td>
<td>45</td>
</tr>
</tbody>
</table>
Security Analysis

- **Rootkit may try to tamper with the HPCs**
  - HPCs are controlled by host (VMM)
- **Rootkit may tamper with the analysis process**
  - Analysis process is done by host (VMM)
- **Rootkit may try to predict the “good” number**
  - The test program can be considered as a “secret key” and can be updated
  - The number of system call, system call argument, and hardware events are huge.
Security Analysis

- Rootkit may undo modifications
  - Rootkit is not aware of the test program
    - Not knowing the monitor time
  - Rootkit tries to identify the test program
    - VMM updates test program
  - Rootkit detects the test program and tries to undo the modification
    - Do or undo dilemma
    - Randomized sampling period
  - Strong rootkit detects the test program accurately and undo all modifications
    - Remove the test program and use machine learning approach
Conclusion

- NumChecker effectively detects and identifies kernel rootkits
  - VMM-based framework (can be applied to different types of virtualizations)
  - Validating the execution of guest system calls (can be changed to work with other software flows)
  - Based on hardware events (free to choose from hundreds of events)

- Using Hardware Performance Counters
  - Feature supported by hardware (Intel, AMD, etc.)
  - Very low performance overhead
  - Tamper-resistant from guest OS
  - Can be applied to other malware
Acknowledgement

• Rodrigo Branco
• Alexander Matrosov
• Nam Nguyen
• Jason Fung
• Mickey Shkatov