Nighthawk: Transparent System Introspection from Ring -3

ESORICS 2019

Lei Zhou¹ ² *, Jidong Xiao³, Kevin Leach⁴, Westley Weimer⁴, Fengwei Zhang⁵ ² **, Guojun Wang⁶

¹ Central South University, China
² Wayne State University, USA
³ Boise State University, USA
⁴ University of Michigan, USA
⁵ SUSTech, China
⁶ Guangzhou University, China

* Work was done while visiting COMPASS lab at WSU; ** The corresponding author
Outline

• Introduction and Background
• Architecture of Nighthawk
• Design and Implementation
• Evaluation: Effectiveness and Performance
• Conclusion
<table>
<thead>
<tr>
<th>Privilege Layers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring 3</td>
<td>User mode virus</td>
</tr>
<tr>
<td>Ring 0</td>
<td>Kernel mode rootkits</td>
</tr>
<tr>
<td>Ring -1</td>
<td>Hypervisor rootkits</td>
</tr>
<tr>
<td>Ring -2</td>
<td>SMM rootkits (SMM reload)</td>
</tr>
</tbody>
</table>
Defense Mechanism

How to defend against the attacks in each layer?
Defense Mechanism

How to defend against the attacks in each layer?

*Deploy a defense at the a more privileged layer!*
Existing Malware Detection

Virtualization based defensive approach (*ring -l*)

**Advantages** ---- Full control of VM.

**Limitations** ---- High performance overhead and more likely to be a new target of attack.
Existing Malware Detection

■ Virtualization based defensive approach (*ring -1*)

*Advantages* ---- Full control of VM.
*Limitations* ---- High performance overhead and more likely to be a new target of attack.

■ Hardware based defensive approach (*ring -2*)

*Advantages* ---- Small TCB and lower layer.
*Limitations* ---- Additional monitoring device or disturbing the normal system execution.
How to better defend against low-level attacks?
How to better defend against low-level attacks?

“Ring -3”?
Higher Privilege System In Intel Architecture

Intel ME system:
- Provide assistance protection for Host
- Strong Isolation but integrate into motherboard

Understanding DMA Malware (DIMVA 2012)
Intel Management Engine

- No Extra Hardware Needed
- Full Privilege
- Small TCB
- Transparency and low performance overhead
Intel Management Engine

- No Extra Hardware Needed
- Full Privilege
- Small TCB
- Transparency and low performance overhead

However, IME related resources are not public to users
Location

Microcontroller embedded in the PCH (older version in MCH)
Outline

- Introduction and Background
- Architecture of Nighthawk
- Design and Implementation
- Evaluation: Effectiveness and Performance
- Conclusion
If we are able to add introspection code into IME system, we can check arbitrary host physical memory.
Details of Components in Nighthawk
Outline

- Introduction and Background
- Architecture of Nighthawk
- Design and Implementation
- Evaluation: Effectiveness and Performance
- Conclusion
Nighthawk Design & Implementation

- Preparing the Target Machine
- Target Host Reconnaissance
- Measuring Integrity via Custom IME
- Command from Remote Machine
High-level Overview of the Implementation
Preparing the Target Machine
- Target Host Reconnaissance
- Measuring Integrity via Custom IME
- Command from Remote Machine
Preparing Target Machine (1) — Code Injection

The Process for introspection code injection in ME

1. Modify registers: TOUUD, REMAP_BASE, REMAP_LIMIT
2. Remap memory (mmap2)
3. Write custom code to accessible address
4. Restore configuration registers

Available Memory

IME External Memory

System DRAM

Management Engine

Target Host

Remote Machine

Target Machine

User Command

Verify Result

Memory Forensics

Report

① Inject custom ME code.

⑦ Report

⑥b mem dump

⑥a Response

③ Command

② Memory Reconnaissance.
How to Inject the Introspection Code

Through Reverse engineering of the ME system code, we find the ideal function entry in which to inject the code.
Preparing Target Machine (2) — Stop Reusing Injection

Stop reusing the injection in ME: leveraging the Intel TXT to lock the related registers.
Nighthawk Design & Implementation

- Preparing the Target Machine
- *Target Host Reconnaissance*
- Measuring Integrity via Custom IME
- Command from Remote Machine
Once the host system initializes, we fetch those basic information.

The information including:
System call table: 0x1653100
Kernel_text: 0x1000000
kvm_intel: 0xf8bc7000
...
Target Host Reconnaissance (2) — Special Case

To **mitigate some attacks like ATRA**, we leverage SMM to get the runtime CPU information after checking SMRAM.
Nighthawk Design & Implementation

- Preparing the Target Machine
- Target Host Reconnaissance
  - *Measuring Integrity via Custom IME*
- Command from Remote Machine
Measuring Integrity via Custom IME

Workflow of Introspection
Nighthawk Design & Implementation

- Preparing the Target Machine
- Target Host Reconnaissance
- Measuring Integrity via Custom IME
- *Command from Remote Machine*
### Command from Remote Machine

#### Diagram:
- **Remote Machine**
  - User Command
  - Verify Result
  - Memory Forensics
  - Message
  - Target Host
  - SMRAM

#### Table:
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Object Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Fetch the physical memory from Target Host to the IME.</td>
<td>SCT The information about System Call Table.</td>
</tr>
<tr>
<td>C</td>
<td>Compare the Target Host memory in the IME system.</td>
<td>LK The information about Linux Kernel.</td>
</tr>
<tr>
<td>T</td>
<td>Transmit the introspection results from the IME to Remote Machine.</td>
<td>HYP The information about Hypervisor.</td>
</tr>
<tr>
<td>D</td>
<td>Dump the Target Host memory from the IME to Remote Machine.</td>
<td>SMM The information about SMRAM.</td>
</tr>
</tbody>
</table>
Outline

• Introduction and Background
• Architecture of Nighthawk
• Design and Implementation
• Evaluation: Effectiveness and Performance
• Conclusion
Evaluation

The test environment platform:

- Intel DQ35JO motherboard with 3.0GHz Intel E8400 CPU, ICH9D0 I/O Controller Hub and 2GB RAM.

- Intel e1000e Gigabyte network card for the network communication.

- We use an earlier BIOS version (JOQ3510J.86A.0933) for injecting code into ME.

- We run Ubuntu with the Linux kernel version 2.6.x to 4.x, along with KVM- and Xen-based Hypervisor.
Effectiveness--General Attacks

Target Object and Attacks

<table>
<thead>
<tr>
<th>Object</th>
<th>Size (KB)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General data</td>
<td>1</td>
<td>0.258 ± 0.010</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.261 ± 0.010</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.267 ± 0.010</td>
</tr>
<tr>
<td>System Call Table</td>
<td>256</td>
<td>0.387 ± 0.120</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>2,048</td>
<td>3.06 ± 0.350</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>3,096</td>
<td>4.67 ± 0.430</td>
</tr>
<tr>
<td>IDT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swapper_pg_dir</td>
<td>4</td>
<td>0.263 ± 0.010</td>
</tr>
<tr>
<td>SMRAM(unlocked)</td>
<td>128</td>
<td>0.383 ± 0.120</td>
</tr>
<tr>
<td>Random</td>
<td>10,240</td>
<td>15.4 ± 3.920</td>
</tr>
</tbody>
</table>

To simulate the attacking environment, we use existing rootkits for OS kernel, SMM, etc., installed in the target system.

We manually modify the memory content in kernel, Xen, KVM and SMM modules.

Through experiments, all attacks illustrated in this table have been detected by Nighthawk.
Effectiveness -- Mitigating Special Attacks

ATRA Detection
We detect ATRA by testing for Page Global Directory and CR3 changes

Transient Attacks Detection
We simulate a transient attack using a toorkit-modified rootkit that changes the pointer address of the system call table.

Our results in the table show that Nighthawk can detect transient attacks in real world.

<table>
<thead>
<tr>
<th>Execution Time (ms)</th>
<th>Attacks Detected Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 8</td>
<td>&lt;2.5%</td>
</tr>
<tr>
<td>12</td>
<td>7.5%</td>
</tr>
<tr>
<td>63</td>
<td>8.3%</td>
</tr>
<tr>
<td>123</td>
<td>22.5%</td>
</tr>
<tr>
<td>218</td>
<td>33.3%</td>
</tr>
<tr>
<td>437</td>
<td>68.3%</td>
</tr>
<tr>
<td>515</td>
<td>81.4%</td>
</tr>
<tr>
<td>643</td>
<td>92.1%</td>
</tr>
<tr>
<td>&gt; 700</td>
<td>100%</td>
</tr>
</tbody>
</table>
Performance Evaluation

- DMA Fetching Overhead
- Integrity Checking Overhead
- Transmission Overhead
DMA Fetching Overhead

Time consumed by fetching data (Pages).
* represents the number of PTEs.
\( \alpha \) represents accessing times.

Time consumed by DMA (User Cases).

### Table: Object Size and Time Consumed

<table>
<thead>
<tr>
<th>Object</th>
<th>Size (KB)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(General data)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.261</td>
<td>0.258 ± 0.010</td>
</tr>
<tr>
<td>4</td>
<td>0.523</td>
<td>0.261 ± 0.010</td>
</tr>
<tr>
<td>64</td>
<td>0.815</td>
<td>0.267 ± 0.010</td>
</tr>
<tr>
<td>256</td>
<td>0.912</td>
<td>0.387 ± 0.120</td>
</tr>
<tr>
<td>2,048</td>
<td>1.28</td>
<td>3.06 ± 0.350</td>
</tr>
<tr>
<td>3,096</td>
<td></td>
<td>4.67 ± 0.430</td>
</tr>
<tr>
<td>System Call Table</td>
<td>4</td>
<td>0.261 ± 0.010</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>6,466</td>
<td>9.75 ± 1.300</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>336</td>
<td>1.31 ± 0.130</td>
</tr>
<tr>
<td>IDT</td>
<td>1</td>
<td>0.258 ± 0.010</td>
</tr>
<tr>
<td>Swapper_pg_dir</td>
<td>4</td>
<td>0.263 ± 0.010</td>
</tr>
<tr>
<td>SMRAM(unlocked)</td>
<td>128</td>
<td>0.383 ± 0.120</td>
</tr>
<tr>
<td>Random</td>
<td>10,240</td>
<td>15.4 ± 3.920</td>
</tr>
</tbody>
</table>

Fetching data from host memory to ME memory
Memory Degradation Due To Introspection

With the benchmark test, the results show that Nighthawk has a very small performance impact to host.
Integrity Checking Overhead

- Time cost depends on the hash algorithm we choose. -- For 4KB memory page, it takes 7.3ms for checking under SDBM hash.

- Note that, for more complexity hash algorithm, e.g., sha1, it takes more time for checking.

- Compared to the fetching time, the checking time is very lower.
Comparison for Checking Overhead

With the SDBM hash verification test, we found the computing performance is much lower than it is in Host. For example, comparing a 6.3MB data, 25s is needed in ME, and 10 ms in Host.

Main factor: ME CPU core has a significantly lower computational capability.

We develop a CPU speed testing program, and the experimental result shows that the ME CPU executes approximately 15 million instructions each second (Meanwhile, billions per second on regular CPUs).
Transmission Overhead

- For a small message (<1KB), takes 228ms on average to pass the data.

- For a dumping data (i.e., > 64KB), we divide the data into multiple packets and transmit via multiple messages. e.g., 64KB data takes 4.9s.
## Performance Evaluation Summary

<table>
<thead>
<tr>
<th>Object</th>
<th>Size (KB)</th>
<th>Data Fetching Time (s)</th>
<th>Comparison Time (s)</th>
<th>Data Transmission Time (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call table</td>
<td>4</td>
<td>0.26±0.010</td>
<td>0.007±0.001</td>
<td>0.224±0.030</td>
<td>0.50±0.030</td>
</tr>
<tr>
<td>kvm_intel.ko</td>
<td>336</td>
<td>1.31±0.130</td>
<td>0.601±0.010</td>
<td>0.231±0.030</td>
<td>2.14±0.150</td>
</tr>
<tr>
<td>PDE</td>
<td>4</td>
<td>0.52±0.010</td>
<td>0.007±0.001</td>
<td>0.230±0.030</td>
<td>0.76±0.040</td>
</tr>
<tr>
<td>SMRAM(unlocked)</td>
<td>128</td>
<td>0.39±0.150</td>
<td>0.320±0.005</td>
<td>0.228±0.030</td>
<td>0.94±0.200</td>
</tr>
</tbody>
</table>
Conclusion

✧ Nighthawk—a transparent introspection framework
  — Leveraging Intel ME
  — High privilege: ring -3
  — Small TCB

✧ Attack scenarios
  — Real-world attacks against OS kernels, type-I and type-II hypervisors, and unlocked system management RAM

✧ Introducing almost zero overhead
Thank you! Questions?

zhangfw@sustech.edu.cn

https://fengweiz.github.com/