Spectre: A Dependable Introspection Framework via System Management Mode
Fengwei Zhang, Kevin Leach, Kun Sun, and Angelos Stavrou.
In DSN'13.

Presented by Fengwei Zhang
Agenda

• Introduction
• Background
• System Framework
• Experimental Results
• Conclusion
Agenda

• Introduction
• Background
• System Framework
• Experimental Results
• Conclusion
Introduction

- Malware detection and analysis remain an open research problem
- Traditionally, malware detection is provided by installing anti-malware tools (e.g., anti-virus) within the OS
- However, these detection tools are vulnerable to malware running at the same level (e.g., rootkits)
- ‘Out-of-box’ introspection mechanism proposed for malware detection and analysis (e.g., Virtual machine introspection)
Introduction

• Virtual Machine Intropsection (VMI) systems run malware within a VM and use analysis tool to introspect the malware from outside
• VMI systems have been widely adopted for malware detection and analysis. They isolate the malware detection software from a vulnerable guest [4, 5, 6]
• Limitations of VMI systems:
  – Large Trusted Computing Base (TCB) (e.g., Xen 4.2 has 208K lines of code)
  – Armored malware can detect the presence of a VM and alter its own execution (e.g., anti-VM techniques)
  – High performance overhead
• We present Spectre, a dependable introspection framework via system management mode
Agenda

• Introduction
• Background
• System Framework
• Experimental Results
• Conclusion
Background

System Management Mode (SMM)
• A CPU mode on the x86 Architecture.
• After entering into SMM, it executes the System Management Interrupt (SMI) handler
• SMI handler stores at a sealed storage called System Management RAM (SMRAM)
• BIOS locks the SMRAM, and the SMRAM is inaccessible from any other CPU modes
• SMM-based systems
  – Integrity checking: HyperGuard [7], HyperCheck [8],
  – HyperSentry [1]
  – SMM rootkits [3, 2]
  – Attacks against SMM [9]
Background

Basic Input and Output System (BIOS) and Coreboot

• BIOS code is stored on-volatile ROM, and it is responsible for hardware initialization before OS starts.

• Coreboot is an open source project aimed to replace the BIOS in current computer

• Spectre uses a custom SMI handler in Coreboot
Agenda

- Introduction
- Background
- System Framework
- Experimental Results
- Conclusion
System Framework

SPECTRE system regularly introspects native memory on target machine

- Enter SMM
- Rebuild semantic data
- Check kernel data
- Check kernel code
- Check program data
- Optional custom module...
- Report alerts

Monitor Machine

'heartbeat' attack occurred?
System Framework

• Step 1: Periodic triggering of SMM
System Framework

• Step 1: Periodic triggering of SMM
• Step 2: Rebuilding semantic information
System Framework

- Step 1: Periodic triggering of SMM
- Step 2: Rebuilding semantic information
- Step 3: Running a detection module
System Framework

- Step 1: Periodic triggering of SMM
- Step 2: Rebuilding semantic information
- Step 3: Running a detection module
- Step 4: Communication with monitor server

Target Machine

SPECTRE system regularly introspects native memory on target machine

Enter SMM → Rebuild semantic data → Check kernel data → Check kernel code → Check program data → optional custom module → Report alerts

Monitor Machine

'heartbeat' attack occurred?
Step 1: Periodic Triggering of SMM

• Two ways to trigger an SMI
  – Software-based: write to an ACPI port specified by chipsets
  – Hardware-based: NIC card, keyboard, mouse, and hardware timer

• Hardware-based method is more reliable than software-based method, so we use a hardware timer at southbridge to periodically assert an SMI
Step 2: Rebuilding Semantic Information

- SMM only sees the raw memory, and does not know the semantics of the memory (e.g. OS data structures)
- Similar to the semantic gap problem in VMI systems
- We manually bridge the semantic gap in our prototype, automatically bridging (e.g., Virtuoso [6], VMST [4])
Semantic Gap Problem in VMI


- SMM-based Systems, TrustZone-based Systems, SGX, other hardware isolated execution environments (HIEEs)
Step 3: Running a Detection Module

• We demonstrate the capability of our framework with three memory-based attacks:
  – Detecting heap spray attacks
  – Detecting heap overflow attacks
  – Detecting rootkits

• Other checking modules can be extended into Spectre with corresponding detection algorithm
Step 4: Communication with Monitor Machine

• The SMI handler alerts the monitor machine over a serial or Ethernet cable
• We port the NIC driver into SMI handler because we do want to trust any code in the OS
• ‘Heartbeat’ message can be used to detect denial of service attack
• Exit from SMM and resume OS states
Agenda

• Introduction
• Background
• System Framework
• Experimental Results
• Conclusion
Prototype Specification

• Hardware
  – Motherboard: ASUS-M2V MX SE
  – CPU: 2.2GHz AMD Sempron LE-1250
  – RAM: 2GB Kingston DDR2
  – NICs: Integrated NIC and Intel e1000 Gigabit with PCI

• Software
  – BIOS: Coreboot+SeaBIOS
  – OSes: Linux (Cent OS 5.5) and Windows XP SP3
Memory Attacks Detection

• Run various memory attacks, and measure the detection time in the SMM
• Detection time = Time at SMM exit - Time at SMM enter

<table>
<thead>
<tr>
<th>Modules</th>
<th>Attacks</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap Spary</td>
<td>Firefox CVE-2009-2478</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Internet Explorer CVE-2010-3971</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Adobe Acrobat CVE-2011-2462</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>Adobe Flash Player CVE-2011-6069</td>
<td>29</td>
</tr>
<tr>
<td>Heap Overflow</td>
<td>XnView CVE-2012-0276</td>
<td>32</td>
</tr>
<tr>
<td>Rootkit</td>
<td>Fu rootkit</td>
<td>8</td>
</tr>
</tbody>
</table>
System Overhead

- Spectre is OS-agnostic, and can detect memory attacks on both Windows and Linux platforms.
- Benchmark: PassMark on Windows and UnixBench on Linux
- First, we run different detection modules, and record their benchmark scores
  - Without detection module
  - Heap spray detection module
  - Heap Overflow detection module
  - Rootkits detection module
- Second, we change the SMI triggering rate, and it ranges from 1/16 s to 5s
System Overhead

- X-coordinate: Sampling interval
- Y-coordinate: Percent overhead

Windows

<table>
<thead>
<tr>
<th>Sampling interval / s</th>
<th>Without detection module</th>
<th>Heap spray module</th>
<th>Heap overflow module</th>
<th>Rootkit module</th>
</tr>
</thead>
<tbody>
<tr>
<td>5s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Linux

<table>
<thead>
<tr>
<th>Sampling interval / s</th>
<th>Without detection module</th>
<th>Heap spray detection module</th>
<th>Heap overflow module</th>
<th>Rootkit detection module</th>
</tr>
</thead>
<tbody>
<tr>
<td>5s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison with VMI Systems

• Smaller code base–Spectre only trust the BIOS, but VMI systems need to trust hypervisor
• More transparent–armored malware with anti-VM techniques cannot detect it
• Better Performance

**Table:** Runtime comparison of introspection programs between Spectre and Virtuoso

<table>
<thead>
<tr>
<th></th>
<th>Spectre (ms)</th>
<th>Virtuoso (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pslit</td>
<td>6.6</td>
<td>450.2</td>
</tr>
<tr>
<td>lsmod</td>
<td>7.6</td>
<td>698.1</td>
</tr>
<tr>
<td><strong>Linux</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pslit</td>
<td>4.3</td>
<td>6494.1</td>
</tr>
<tr>
<td>lsmod</td>
<td>4.4</td>
<td>2437.0</td>
</tr>
</tbody>
</table>
Agenda

• Introduction
• Background
• System Framework
• Experimental Results
• Conclusion
Conclusion

• We introduce a hardware-assisted framework that can examine code across all layers of a running system

• Spectre is OS-agnostic and fully transparent to higher level software

• We have implemented a prototype of our framework in both Linux and Windows, and demonstrates that our system can detect various memory attacks including heap spray, heap overflow and rootkits.
References

HyperSentry: enabling stealthy in-context measurement of hypervisor integrity.

[2] BSDaemon, coideloko, and D0nAnd0n.

SMM rootkits: a new breed of OS independent malware.

Space Traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection.

Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction.

Virtuoso: Narrowing the semantic gap in virtual machine introspection.

Preventing and detecting Xen hypervisor subversions.
Blackhat Briefings USA, 2008.

HyperCheck: A hardware-assisted integrity monitor.