Seeing Through the Same Lens: Introspecting Guest Address Space at Native Speed

Siqi Zhao*, Xuhua Ding*, Wen Xu◆, Dawu Gu◇

* Singapore Management University
◆ Georgia Institute of Technology
◇ Shanghai JiaoTong University
Outline

• Problem
• Design
• Implementation
• Evaluations
Problem

• Considering introspecting kernel objects
  • Untrusted live VM
  • VMI tool running outside of the VM
• The VMI tool and the target objects are in different address spaces.
• The VMI tool needs to perform a sequence of operations for every kernel object access.
Problem

- Considering introspecting kernel objects
  - Untrusted live VM
  - VMI tool running outside of the VM
- The VMI tool and the target objects are in different address spaces.
- The VMI tool needs to perform a sequence of operations for every kernel object access.
Inadequacy of Page Table Walk

• Page table walk lies at the heart of VMI
• Slow, compared to native address translation by MMU
  • A number of loads from memory
• An experiment to evaluate the slowness of software based page table walk
  • Periodically modifies one task->cred pointer
  • Closely monitor the value of the pointer by repeatedly reading it from outside
  • Cannot catch up with frequent transient guest state changes
Inadequacy of Page Table Walk

- Mapping consistency with the target is not maintained
  - Uses any mappings: ample room for the guest to present false mappings
  - The target VM may also make transient changes to the page table
  - Caching techniques that aim to enhance efficiency further deteriorate the situation, giving up consistency for efficiency.
Immersive Execution Environment (ImEE) Architecture

- ImEE is essentially a special VM created on-demand by VMI applications.
- Scheduled by the hypervisor
- Consists of only a vCPU and a small amount of memory: code and data
- ImEE hosts a piece of code called *ImEE agent* that actually performs introspection
- ImEE acts as a memory access engine for VMI applications
  - Only perform memory read
  - Native speed read
  - Page table is consistent with target at any moment
Basic Idea

• An environment with a twisted address mappings
  • Cloned CR3 content
  • Cloned EPT, with mappings from target EPT and restricted permission
• The result: a VA is translated to the same HPA by the MMU in both environments
Making it work

• Implementation issues:
  • Need room in the virtual address space for
    • our own code
    • exchanging data with VMI tool
  • We want to avoid touching GPT

• What about the number of redirected pages?
Immersive Execution Environment (ImEE)

• Two address spaces in the ImEE
• Local address space is for the ImEE agent to interact with the VMI application
• The idea ‘incarnates’ as the target address space in ImEE.
• Introspection is only performed in the target address space
Target Address Space

- GPA space is split by the EPT
  - All address mappings are the same as in the target VM, with read-only permission
  - One page is redirected by EPT to the agent’s code page, with execute-only permission
- Two possible kinds of translation in the target address space:
  - Instruction fetch
  - Memory read
Target Address Space

• Example:

\[ 0xBFF0: \text{mov (0x1000), %eax} \]
Local Address Space

- Minimizes the number of redirected page in the target address space
- Only two pages are mapped
  - All pages except one are mapped to code
  - The remaining one is mapped to data
- Allowing the agent to be executed almost anywhere, because we do not know the load address beforehand
Local Address Spaces

• Example:

0xBFFA: \texttt{mov} \%eax, (0x2000)
The ImEE Agent

• The ImEE agent is the only code that runs inside ImEE
  • Reside within one page, self-contained
  • Position independent
  • Granted ring 0 privilege
• Initially, the agent is loaded at a page whose VA is mapped as executable in the guest page tables.
  • The hypervisor uses the page that the current IP points to
The ImEE Agent

• The agent’s execution straddles between the two address spaces.
• Simplified pseudo-code:

1. eax = data[request]
2. cr3 = target_cr3   /* switch to target address space */
3. xmm0 = *eax
4. cr3 = imee_cr3    /* switch to local address space */
5. data[result] = xmm0
Other Issues

- Blind spot
  - The code page in the target GPA space is redirected
  - Any virtual address mapped to this GPA cannot be read

- Cannot be eliminated

- Only detected when introspection is on the blind spot
  - Relying on EPT mappings
  - Relocate agent once detected
Advantages

• Native speed
  • Address translation is performed at native speed by hardware.

• Consistency
  • Page table used is kept consistent with the current one in the target.
  • CR3 is synchronized
Implementation

• We implemented a prototype of ImEE
  • Hypervisor changes: modified KVM module
  • Consists of around 1400 SLOC
  • Two new IOCTLs as interface to user space
  • Optimized code path that handles ImEE specific VM exit
• Agent: specially crafted assembly code
  • Within one page, a few tens of instructions
  • Position independent
  • One data page for exchange data with VMI tool
Evaluation

• We use LibVMI as the base line.
  • LibVMI: the only open source tool
  • Serves as building block for various other tools such as Volatility

• Experiment setup:
  • Hardware: Intel Core i7-2600, 4GB DDR3 RAM
  • Guest VM: 1GB RAM and one vCPU
Evaluation

• ImEE Overhead
  • Launch time: time taken for the hypervisor to prepare relevant data such as the EPT
  • Activation time: time for a launched, but not running, ImEE to begin execute the agent code

<table>
<thead>
<tr>
<th></th>
<th>ImEE</th>
<th>LibVMI</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch time</td>
<td>97 μs</td>
<td>100 ms</td>
<td><strong>1031 times</strong></td>
</tr>
<tr>
<td>Activation time</td>
<td>3.2 μs</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Evaluation

• Guest access speed
  • We measure the time take to read a number of bytes from the target
  • LibVMI’s translation cache is on, data cache off

<table>
<thead>
<tr>
<th># of Bytes</th>
<th>ImEE</th>
<th>LibVMI (μs)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.353</td>
<td>18.4</td>
<td>52 times</td>
</tr>
<tr>
<td>64</td>
<td>0.358</td>
<td>18.5</td>
<td>52 times</td>
</tr>
<tr>
<td>128</td>
<td>0.389</td>
<td>18.4</td>
<td>47 times</td>
</tr>
<tr>
<td>512</td>
<td>1.643</td>
<td>18.9</td>
<td>11 times</td>
</tr>
<tr>
<td>1024</td>
<td>1.715</td>
<td>38.1</td>
<td>22 times</td>
</tr>
</tbody>
</table>
Tools

• `syscalldmp`: dumps totally 351 entries of the guest’s system call table
• `pidlist`: lists all process identifiers in the guest.
• `pslist`: lists all tasks’ identifiers and task names stored in task struct.
• `credlist`: lists all tasks’ credential structures referenced by the task struct’s cred pointer.
Evaluation

We ran our four tools in four setups: ImEE, kernel, LibVMI on KVM and LibVMI on Xen

Measure time taken to complete the task

Results:
- Comparable to kernel
- Significant speedup compared to LibVMI

<table>
<thead>
<tr>
<th>Tool</th>
<th>Kernel (μs)</th>
<th>LibVMI / KVM (μs)</th>
<th>LibVMI / Xen (μs)</th>
<th>ImEE (μs)</th>
<th>Speedup (KVM)</th>
<th>Speedup (Xen)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscalldmp</td>
<td>0.2</td>
<td>28.2</td>
<td>43</td>
<td>2.9</td>
<td>9 times</td>
<td>15 times</td>
</tr>
<tr>
<td>pidlist</td>
<td>10</td>
<td>5887</td>
<td>2180</td>
<td>31.6</td>
<td>186 times</td>
<td>68 times</td>
</tr>
<tr>
<td>pslist</td>
<td>10.4</td>
<td>8319</td>
<td>1477</td>
<td>38.6</td>
<td>215 times</td>
<td>38 times</td>
</tr>
<tr>
<td>credlist</td>
<td>25.3</td>
<td>8234</td>
<td>2274</td>
<td>25.6</td>
<td>321 times</td>
<td>88 times</td>
</tr>
</tbody>
</table>
Evaluation

• Scanning multiple VMs
  • We setup four VMs and measure:
    • Time to scan every VM
    • Time to switch scan target
  • Results:

<table>
<thead>
<tr>
<th></th>
<th>LibVMI</th>
<th>ImEE</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning all VMs</td>
<td>561 ms</td>
<td>377 µs</td>
<td>1400 times</td>
</tr>
<tr>
<td>Switching target VM</td>
<td>19 ms</td>
<td>4.4 µs</td>
<td>4300 times</td>
</tr>
</tbody>
</table>
Conclusion

• ImEE is a novel memory access engine for out-of-VM introspection applications for live VM.

• Based on hardware virtualization, ImEE shows remarkable speed up compared to existing approaches.

• ImEE maintains mapping consistency during introspection. Complemented by its high speed, ImEE is suitable for security sensitive VMI applications.
Questions?