Dynamic VM Monitoring using Hypervisor Probes

Z. J. Estrada, C. Pham, F. Deng, L. Yan, Z. Kalbarczyk, R. K. Iyer

European Dependable Computing Conference
2015-09-09
# Dynamic VM Monitoring

## Goal

On-demand VM Monitoring to reduce the effort required to harden computing systems against failures and attacks.

- ✓ Uptime requirements
- ✓ Effort required
- ✓ QA concerns
- ✓ Lack of knowledge
VM Monitoring

Reliability & Security Monitoring

Recording and analyzing a computer system to detect failures and attacks.

- Passive - polling based
- Active - event based
VM Monitoring

VM

Applications

OS

Hypervisor

KVM
VM Monitoring

VM

Applications

OS

Hypervisor

KVM

Monitor
VM Monitoring

- Hook based
- On-Demand - Add/Removal at Runtime
- Vulnerability, Hang, and Infinite Loop Detectors
- Userspace support
Monitor is running inside the hypervisor
VM execution reaches a hook
Control is transferred to the monitor
The monitor performs its monitoring function
Control is transferred back to the VM
The VM resumes normal execution
Hook-Based VM Monitoring

Previous techniques:

- Active monitoring
- Protected hooks
  - Guest OS only - no userspace
  - Not dynamic - boot time config
  - Require guest OS modifications
Goals

Hook-based monitoring should:

- be protected from attacks in the VM
- be simple to use
- not require guest OS modification
- be runtime adaptable
- allow for arbitrary hook placement
Hypervisor Probes
Hardware Assisted Virt.

Host Mode
(root)

User
Kernel

Guest Mode
(non-root)

User
Kernel

VMEntry
VMExit
Hypervisor Probes

- Event on guest execution
  - Event transfers control to hypervisor (VM Exit)
  - Perform monitoring after that event
- Hooks added/removed at runtime
- Monitors applications and the guest OS
Hprobe Architecture

Host System

- Status Checker
- Hprobe user agent
  - ioctl(…)
- Detector 1
- Detector 2
- Detector n

Host Linux kernel

- Hprobe Kernel agent
  - Insert/Remove probes
  - Set single step

VM

- Event Forwarder
- Helper APIs
- KVM Hypervisor
  - Probe

ECE ILLINOIS
Department of Electrical and Computer Engineering
Hprobes API

```c
int HPROBE_add_probe( );
int HPROBE_remove_probe( );
```

- `addr_info`: gva+cr3
- `vmid`: unique id for VM
- `vcpu_type`: vcpu state
Probe $\Rightarrow$ Event Forwarder

VM

Hypervisor

... pushl %eax
incl %eax
decl %ebx
...

probe hit
trap
rewrite
resume
...
Probe $\Rightarrow$ Event Forwarder

VM

Hypervisor

... pushl %eax
int3
decl %ebx
...

probe hit
(handler())
Reset inst.
single step execute inst.
trap rewrite int3 resume...

ECE ILLINOIS
Department of Electrical and Computer Engineering
Probe \Rightarrow Event Forwarder

VM

probe hit (int3)

Hypervisor

\text{handler()}

Detector

\ldots

\text{pushl} %eax

\text{int3}

\text{decl} %ebx

\ldots
Probe ⇒ Event Forwarder

VM
probe hit (int3)

Hypervisor
handler()
Reset inst.

... pushl %eax
incl %eax
decl %ebx
...

13
Probe ⇒ Event Forwarder

VM
probe hit (int3)

Hypervisor
handler() Reset inst.
single step

... pushl %eax incl %eax decl %ebx ...

...
Probe $\Rightarrow$ Event Forwarder

**Diagram:**

- **VM:**
  - probe hit (int3)
  - execute inst.
  - trap

- **Hypervisor:**
  - handler()
  - Reset inst.
  - single step
  - rewrite int3

**Code Snippet:**

```
pushl %eax
int3
decl %ebx
```

**Notes:**

- Probe hit (int3)
- Execute inst.
- Trap
- Handler
- Reset inst.
- Single step
- Rewrite int3
Probe $\Rightarrow$ Event Forwarder

... pushl %eax
int3
decl %ebx
...

VM

probe hit (int3)
execute inst.
trap
...

Hypervisor

handler()
Reset inst.
single step
rewrite int3
resume

ECE ILLINOIS
Department of Electrical and Computer Engineering
Userspace Probe Challenge

Guest Page Tables
Userspace Probe Challenge
Userspace Probe Challenge

Guest Page Tables
Extended Page Tables (EPT)

- Guest OS has full control over PTs
- 2nd set of HW PTs for GPA → HPA
- Use EPT to write-protect Guest Page Table

Goals

Hook-based monitoring should:
+ be protected from attacks in the VM
+ be simple to use
+ not require guest OS modification
+ be runtime adaptable
+ allow for arbitrary hook placement
Goals

Hook-based monitoring should:
✓ be protected from attacks in the VM
+ be simple to use
+ not require guest OS modification
+ be runtime adaptable
+ allow for arbitrary hook placement
Goals

Hook-based monitoring should:
✓ be protected from attacks in the VM
✓ be simple to use
+ not require guest OS modification
+ be runtime adaptable
+ allow for arbitrary hook placement
Goals

Hook-based monitoring should:
✓ be protected from attacks in the VM
✓ be simple to use
✓ not require guest OS modification
+ be runtime adaptable
+ allow for arbitrary hook placement
Goals

Hook-based monitoring should:
✓ be protected from attacks in the VM
✓ be simple to use
✓ not require guest OS modification
✓ be runtime adaptable
＋ allow for arbitrary hook placement
Goals

Hook-based monitoring should:
✓ be protected from attacks in the VM
✓ be simple to use
✓ not require guest OS modification
✓ be runtime adaptable
✓ allow for arbitrary hook placement
Hprobe Microbenchmarks

- probe @ `noop` kernel function
- execute 1M times

[2] Adapted from an image by Fei Deng
## Hook-based VM Monitoring

<table>
<thead>
<tr>
<th>Name</th>
<th>Latency</th>
<th>User</th>
<th>Dynamic</th>
<th>Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lares</td>
<td>28μs</td>
<td>No</td>
<td>No</td>
<td>Hypervisor/Guest</td>
</tr>
<tr>
<td>SIM</td>
<td>0.40μs</td>
<td>No</td>
<td>No</td>
<td>Hypervisor/Guest</td>
</tr>
<tr>
<td>hprobes</td>
<td>2.6μs</td>
<td>Yes</td>
<td>Yes</td>
<td>Hypervisor</td>
</tr>
</tbody>
</table>
Hook-based VM Monitoring

<table>
<thead>
<tr>
<th>Name</th>
<th>Latency</th>
<th>User</th>
<th>Dynamic</th>
<th>Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lares</td>
<td>28 µs</td>
<td>No</td>
<td>No</td>
<td>Hypervisor/Guest</td>
</tr>
<tr>
<td>SIM</td>
<td>0.40 µs</td>
<td>No</td>
<td>No</td>
<td>Hypervisor/Guest</td>
</tr>
<tr>
<td>hprobes</td>
<td>2.6 µs</td>
<td>Yes</td>
<td>Yes</td>
<td>Hypervisor</td>
</tr>
</tbody>
</table>

- as-a-Service is worth slight performance cost
Detectors

What detectors can we build with hprobes?
Detectors

What detectors can we build with hprobes?

- Arbitrarily chose events
- On-demand
- Access to VM memory & CPU state
Heartbeat/watchdog

App

Detector
Heartbeat/watchdog

App

Detector

Insert Probe

reset timer

申报失败
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

reset timer

ECE ILLINOIS
Department of Electrical and Computer Engineering
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

Probe Hit

reset timer

declare failure

ECE ILLINOIS
Department of Electrical and Computer Engineering
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

Probe Hit

reset timer

reset timer
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

reset timer

reset timer

Probe Hit

timer expires

declare failure
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

reset timer

reset timer

timer expires

declare failure
Watchdog - Performance

PI-QMC Main Loop Runtime

No Detector
With Detector

Internal Sample Loop Size
24000
26000
28000
30000
32000
34000

Time (msec)
Detectors

Infinite Loop Detector
Detectors

Infinite Loop Detector

- Kernel or App-level
- Previously determined threshold
- Or register
Infinite Loop Detector

```c
for(i=0; i<10; i++) {
    ...
}
//after loop
```
Infinite Loop Detector

```c
for(i=0; i<10; i++) {
    ...
}
//after loop
```
Infinite Loop Detector

```c
for(i=0; i<10; i++) {
  ...
}
//after loop
```
Without Infinite Loop

<table>
<thead>
<tr>
<th>Application</th>
<th>Time (s)</th>
<th>95% CI (s)</th>
<th>% overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>1.13</td>
<td>0.0325</td>
<td>N/A</td>
</tr>
<tr>
<td>Naïve ILD - Page</td>
<td>1.26</td>
<td>0.0229</td>
<td>11.5</td>
</tr>
<tr>
<td>Naïve ILD - No Page</td>
<td>1.26</td>
<td>0.0265</td>
<td>11.8</td>
</tr>
<tr>
<td>Smart ILD - Page</td>
<td>1.14</td>
<td>0.0267</td>
<td>1.15</td>
</tr>
<tr>
<td>Smart ILD - No Page</td>
<td>1.15</td>
<td>0.0215</td>
<td>1.9</td>
</tr>
</tbody>
</table>
Consider this situation
A vulnerability is announced
STOPGAP

At a later time, a patch is released
What can we do?
We may have to follow a maintenance window
Even when the bug and patch are coreleased
To mitigate risk, we would like a stopgap
Solution

Use an *Hprobe*-based Detector
Emergency Detector

Should be...

- easier than a patch
- simpler than a patch
- less disruptive than a patch
- less risky than a patch
Emergency Detector

- CVE-2008-0600 - Privilege Escalation in `vmsplice()` [3]

- Integer overflow in a `struct iovec` argument

- Corrupts OS (kernel) stack

- Execute attack payload

```c
struct iovec {
    void  *iov_base;
    size_t iov_len;
};
```

Emergency Detector

- Added to running guest OS
- Detects malicious value that causes overflow
- Two modes of operation
  - Read-only mode: does not change anything
  - Fix mode: malicious value $\Rightarrow$ benign value
Emergency Detector

- Probe at \texttt{vmsplice()} syscall
- Get value of \texttt{iov_len} off of the stack
procedure VMSPLICE_HANDLER(vcpu)

    iov_pointer ← read_guest(esp+arg_offset)
    iov_len ← read_guest_virt(iov_pointer)

    if iov_len ≥ BAD_VALUE then
        HANDLE_EXPLOIT_ATTEMPT(vcpu)
    end if

end procedure
Detector Performance

- Checkpoint/Restart In Userspace
- Two scientific computing applications
  - Folding @ Home
  - Path-integral Quantum Monte Carlo
- Three cases:
  - Normal: base case without monitoring
  - hprobe: only monitor `sys_vmsplice`
  - Naïve: monitor all system calls
Detector Performance

<table>
<thead>
<tr>
<th>Application</th>
<th>Runtime ± 95% CI (s)</th>
<th>overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F@H Normal</td>
<td>0.221 ± 0.0092</td>
<td>0</td>
</tr>
<tr>
<td>F@H w/hprobe</td>
<td>0.228 ± 0.012</td>
<td>3.30</td>
</tr>
<tr>
<td>F@H w/Naïve</td>
<td>0.253 ± 0.0085</td>
<td>14.4</td>
</tr>
<tr>
<td>pi-qmc Normal</td>
<td>0.137 ± 0.0063</td>
<td>0</td>
</tr>
<tr>
<td>pi-qmc w/hprobe</td>
<td>0.140 ± 0.0073</td>
<td>1.73</td>
</tr>
<tr>
<td>pi-qmc w/Naïve</td>
<td>0.152 ± 0.0051</td>
<td>11.1</td>
</tr>
</tbody>
</table>
Thoughts

- Zero overhead without `vmsplice()`
- Cloud provider doesn’t need tenant to update
- Can be used while official fix is in QA
- Don’t need full understanding of bug
## VM Monitoring Techniques

<table>
<thead>
<tr>
<th></th>
<th>Hprobes  (EDCC'15)</th>
<th>HyperTap  (DSN '14)</th>
<th>LiveWire  (NDSS'03)</th>
<th>LibVMI    (ACSAC'07)</th>
<th>LIM  (CCS'09)</th>
<th>Limes (SP'08)</th>
<th>Lycosid  (VEE'08)</th>
<th>Osck  (ASPLOS'11)</th>
<th>Virtuoso (SP'11)</th>
<th>VMST (SP'12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-demand Add/Remove</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Changes to VM</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Userspace Monitoring</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Root-of-trust (invariant)</td>
<td>OS</td>
<td>HW</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>HW</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
</tr>
<tr>
<td>Active/Passive Mon.</td>
<td>A (Hook)</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>A (Hook)</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Auto-generate Monitoring</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**This Presentation**

<table>
<thead>
<tr>
<th>Desirable Feature</th>
<th>✓ = Supported Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ = Unsupported Feature</td>
<td></td>
</tr>
</tbody>
</table>
Acknowledgements

- Collaborators:
  Cuong Pham, Fei Deng, Dr. Lok Yan, Prof. Zbigniew Kalbarczyk, Prof. Ravi Iyer
Summary

- VM Monitoring
- How hprobes work
- Microbenchmarks
- Emergency Detector