Keeping an Eye on Virtualization
VM Monitoring Techniques & Applications

Zak Estrada
Gary Wang
Dr. Zbigniew Kalbarczyk

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Who are these guys?

The DEPEND group focuses on the research, design and validation of highly-available, reliable and trustworthy computing systems and networks.

http://depend.csl.illinois.edu/
The Plan

8am - 9:15am
- Key Concepts
- Validation

15min break

9:30am-11am
- Virtual Machine Introspection
- Hardware Invariant Techniques

15min break

11:15am-noon
- Hook-based Monitoring
The Plan (cont.)

- All sections will end with discussion
- Ask questions at any time
Key Concepts: Virtualization
Virtual Machines

“VMs have finally arrived. Dismissed for a number of years as academic curiosities, they are now cost-effective techniques for organizing computer resources to provide system flexibility and support for unique applications.”
Virtual Machines

“VMs have finally arrived. Dismissed for a number of years as academic curiosities, they are now cost-effective techniques for organizing computer resources to provide system flexibility and support for unique applications.”

– Robert Goldberg, 1974
Virtual Machines

- Main idea: single machine $\Rightarrow$ many
- Only emulate privileged instructions
Virtualization Requirements

Virtualization Requirements\textsuperscript{[1]}

- **Efficiency**
  - Most instructions execute natively

\textsuperscript{[1]} Popek and Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures,” 1974
Virtualization Requirements

- Efficiency
  - Most instructions execute natively

- Equivalence
  - VM Behavior = Bare Metal Behavior

Virtualization Requirements

- **Efficiency**
  - Most instructions execute natively

- **Equivalence**
  - VM Behavior = Bare Metal Behavior

- **Resource control**
  - VMM has complete control

Types of hypervisors\[^{2}\]

![Diagram showing types of hypervisors: Type 1 (native, bare metal) leading to Xen, VMWare ESX]

Types of hypervisors

**TYPE 1** native (bare metal)

- Hardware
- Hypervisor
- OS

**Xen, VMWare ESX**

**TYPE 2** (hosted)

- Hardware
- OS
- Hypervisor
- OS

**Virtualbox, QEMU**

x86 Privilege Rings

Virtualization on x86

x86: unfriendly to virtualization

- Not all priv. ops generate traps (e.g. `popf`)
- No MMU virtualization
- 1998 VMware: binary translation

Virtualization on x86

x86: unfriendly to virtualization

- Not all priv. ops generate traps (e.g. `popf`)
- No MMU virtualization
- 1998 VMware: binary translation

![Diagram showing the ring levels and virtualization process]

Paravirtualization

Paravirtualization

Question
What are the advantages/disadvantages of Paravirtualization?
Hardware Assisted Virt.

In 2006, Intel VT-x and AMD AMD-V ⇒ VMM much easier to implement

- Allowed for an unmodified guest OS
- Guest-mode/Host-mode
- **VM Entry** → Guest mode
- **VM Exit** on priv. operations
- Added instructions, e.g. `vmlaunch/vmcall`
VM Entry/Exit Example

Guest mode

Host mode
VM Entry/Exit Example

Guest mode

Host mode

Launch VM
VM Entry/Exit Example

Guest mode

Host mode

Launch VM

VM Entry
VM Entry/Exit Example

Guest mode

Host mode

Launch VM

VM Entry
VM Entry/Exit Example

- Guest mode
- Host mode
- Launch VM

Allocate new memory
VM Entry/Exit Example

- Guest mode
- Host mode
- Launch VM
- Allocate new memory
- VM Exit: PAGE_FAULT
- VM Entry
VM Entry/Exit Example

- Guest mode
- Launch VM
- Allocate new memory
- VM Exit: PAGE_FAULT
- VM Entry

- Host mode
VM Entry/Exit Example
VM Entry/Exit Example

- Guest mode
- Host mode
- Launch VM
- Allocate new memory
- VM Exit: PAGE_FAULT
- VM Entry

ECE ILLINOIS
Department of Electrical and Computer Engineering
VM Entry/Exit Example

Guest mode

Host mode

Launch VM

Allocate new memory

VM Exit: PAGE_FAULT

Context Switch

VM Entry

VM Entry
VM Entry/Exit Example

- **Guest mode**
- **Host mode**
- **Launch VM**

VM Entry: PAGE_FAULT

VM Entry: CR_ACCESS

Allocate new memory

Context Switch
VM Entry/Exit Example
VM Entry/Exit Example

Guest mode

Launch VM

Allocate new memory

VM Exit: PAGE_FAULT

VM Entry

Context Switch

VM Exit: CR_ACCESS

Host mode

...
VM Entry/Exit Example
Paged Virtual Memory
Paged Virtual Memory
Question

Should we allow the VM to control page tables?

- Cannot let VM control page tables
- But it still should think it does
- Use Shadow Tables in the Hypervisor
- Update w/VM Exits on paging operations
Shadow Page Tables

- Cannot let VM control page tables
  - But it still should think it does
- Use *Shadow* Tables in the Hypervisor
  - Update w/VM Exits on paging operations

Question: What are some drawbacks of Shadow Page Tables?
Shadow Page Tables

- Cannot let VM control page tables
  - But it still should think it does
- Use *Shadow* Tables in the Hypervisor
  - Update w/VM Exits on paging operations

*Question*

What are some drawbacks of Shadow Page Tables?
Second-Level Paging

- 2nd set of HW PTs for GPA → HPA
- Guest OS has full control over PTs

Question
What are the advantages/disadvantages of Hardware-Assisted Virtualization?
Open-Source Virtual Machine Monitors (VMMs)
Xen

- Amazon EC2
Kernel Virtual Machine

- VM1: qemu-kvm process
- VM2: qemu-kvm process
- VMn: qemu-kvm process

KVM
Host OS
Hardware

- Default for Openstack
Thoughts on Virtualization

- Virtualization has overhead
- Many VMs run only one application
- Many people running Linux on Linux (e.g. KVM w/Linux guest)

**Question**

Can we do better?
Linux Containers (LXC)

- Containers share kernel
  - no “trap and emulate”
- Isolation via cgroups/namespaces
- No HAV involved → always running in “host mode”
Linux Containers (LXC)
Linux Containers (LXC)

Question
What are the advantages/disadvantages of containers?
Managing VMs/Hypervisors

libvirt
VIRTUALIZATION API

virsh virt-manager virt-factory oVirt

KVM LXC OpenVZ UML Xen ESX other...
Virtualization Summary

Paravirtualization
  ▶ Xen

HAV
  ▶ VMExit
  ▶ Paging/EPT
  ▶ KVM

Lightweight Replacement
  ▶ Linux Containers

libvirt
Hook-based Monitoring
Formal Model for Hooks

Figure 1: Formal model of secure active monitoring shown with potential attacks.

Legend of attacks:
A1: Bypass hook
A2: Modify event context
A3: Tamper with security application
A4: Tamper with dependencies
A5: Tamper with response

Formal Model for Hooks

- Notification only on legitimate events
- VM state fixed during processing
- Monitor behavior can’t be altereded
- Response is enforced

Hook-based VM Monitoring Examples
Hooks - VM Monitoring

When designing, have to consider:

- How to hook?
- Where to run the monitor?
Lares

- How to hook?
  - `VMCALL` to hypervisor
- Where to run the monitor?
  - Separate security VM
Xen with windows guest
Lares

- Xen with windows guest
- Hook $\Rightarrow$ replace entries in dispatch table
Lares

- Xen with windows guest
- Hook $\Rightarrow$ replace entries in dispatch table
- Entries go to trampoline that does a `VMCALL`
Lares

- Xen with windows guest
- Hook $\Rightarrow$ replace entries in dispatch table
- Entries go to trampoline that does a `VMCALL`
- Command from target is passed to security driver in secure VM
Lares - Diagram

Secure-In VM Monitoring

- How to hook?
  - jump to entry gate
- Where to run the monitor?
  - Separate address space inside VM
Secure-In VM Monitoring

- Hardware-Assisted Virtualization w/KVM
Secure-In VM Monitoring

- Hardware-Assisted Virtualization w/KVM
- Sets up a protected address space in target
Secure-In VM Monitoring

- Hardware-Assisted Virtualization w/KVM
- Sets up a protected address space in target
- Hooks are inserted into kernel functions
Hooks - SIM

Properties of Lares, SIM, ... 

Pros/cons?
Properties of Lares, SIM, ...

Pros/cons?

+ Active monitoring
+ Protected hooks
  - guest OS only - no userspace
  - Not dynamic - boot time config
  - Require guest OS modifications
  - Lares - only dispatch tables?
  - SIM - limit on number of VMs
Detailed Example: hprobes
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

- VMExit on instruction execution (`int3`)
  - execute a `handler()`
- Hooks added/removed at runtime
- Can monitor userspace and the guest OS
Hprobe Architecture

Host System

- Status Checker
- Hprobe Kernel agent
  - ioctl(...)
- Detector 1
- Detector 2
- Detector n

VM

- Event Forwarder
- Helper APIs
- KVM Hypervisor

Host Linux kernel

- Set/Remove probes
- Insert/Remove probes
- Set single step
- Probe
- Probe
- Probe

ECE ILLINOIS
Department of Electrical and Computer Engineering
Hprobes API

```c
int HPROBE_add_probe(addr_info, vmid,
                      (*handler)(vcpu_type *VCPU));

int HPROBE_remove_probe(addr_info, vmid);
```
Hprobes API

```c
int HPROBE_add_probe(addr_info, vmid, (*)handler(vcpu_type *VCPU));

int HPROBE_remove Probe(addr_info, vmid);
```

- **addr_info**: gva+cr3
- **vmid**: unique id for VM
- **vcpu_type**: vcpu state
VM

probe hit (int3)

Hypervisor

handler()

Reset inst.
VM

probe hit (int3)

execute inst.

Hypervisor

handler()
Reset inst.

single step
VM

probe hit
(int3)

execute inst.

trap

Hypervisor

handler()

Reset inst.

single step

rewrite int3

...
VM

probe hit (int3)

execute inst.

trap...

Hypervisor

handler()
Reset inst.

single step

rewrite int3

resume
What an Hprobe Looks Like

Original

... pushl %eax incl %eax decl %ebx ...
...
What an Hprobe Looks Like

With Probe

...  
pushl %eax  
int3  
decl %ebx  
...
What an Hprobe Looks Like

**With Probe**

```assembly
... pushl %eax
int3
decl %ebx
...```

...
What an Hprobe Looks Like

With Probe

```assembly
pushl %eax
int3
decl %ebx
...
```

Execute

```assembly
pushl %eax
incl %eax
decl %ebx
...
```
What an Hprobe Looks Like

<table>
<thead>
<tr>
<th>With Probe</th>
<th>Execute</th>
<th>Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushl %eax</td>
<td>pushl %eax</td>
<td>pushl %eax</td>
</tr>
<tr>
<td>int3</td>
<td>incl %eax</td>
<td>int3</td>
</tr>
<tr>
<td>decl %ebx</td>
<td>decl %ebx</td>
<td>decl %ebx</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Demo: Hprobe Example

- Single kernelspace probe
- `sync` system call
- Probe added at runtime
Demo: Hprobe Example

- Single kernelspace probe
- `sync` system call
- Probe added at runtime
Userspace Programs

- Need to overwrite instructions in user programs
- Use CR3 to identify process
- But... just using CR3 isn’t good enough
Userspace Programs

- Need to overwrite instructions in user programs
- Use CR3 to identify process
- But... just using CR3 isn’t good enough

Question
Why might CR3 not be enough?
Pages Aren’t Always in RAM!
Use Page Permissions

Translate GVA to GPA
Use Page Permissions

Translate GVA to GPA

Translation present?

Yes

No
Use Page Permissions

Translate GVA to GPA

Add probe

Yes

Translation present?

ECE ILLINOIS
Department of Electrical and Computer Engineering
Use Page Permissions

Translate GVA to GPA

Add probe

Translation present?

Yes

No

Set write protect on Guest PTE
Use Page Permissions

1. Translate GVA to GPA
2. Check if translation is present?
   - Yes: Add probe
   - No: Set write protect on guest PTE

EPT violation on PTE write attempt
Use Page Permissions

Translate GVA to GPA

Translation present?

Add probe

Yes

No

Set write protect on Guest PTE

EPT_violation on PTE write attempt

Set single-step mode
Use Page Permissions

1. **Translate GVA to GPA**
   - **Yes**: Add probe
   - **No**: Set write protect on Guest PTE

2. Add probe if translation present?
   - **Yes**: PTE written after instruction executes
   - **No**: Set single-step mode

3. EPT violation on PTE write attempt
Hprobes

- Active monitoring
- Transparent - OS agnostic
- Dynamic
- Protect w/EPT
- Simple
Hprobe microbenchmarks

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

Hprobe Single Probe Latency

Xeon E5430 (2007) vs. Xeon E5-2660 (2012)
# Hook-based Techniques

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

What can we do with this?
Detectors

What can we do with this?

- Arbitrarily chose events
  - At runtime
- App-level monitoring
Vulnerability & Patch Public
Vulnerability Public

Patch Released
Vulnerability Public

Patch Released

?
Vulnerability Public

Patch Released

Maintenance Window

?
Vulnerability & Patch Public

Maintenance Window

?
Solution
Use an *Emergency Detector*
Emergency Vulnerability Detector

- CVE-2008-0600 - Privilege Escalation in `vmsplice()` \[10\]
- integer overflow in a `struct iovec` system call argument
- insert probe at `vmsplice()` syscall
- get value of argument 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 ≥ 0xFFFFFFFF
      HANDLE_EXPLOIT_ATTEMPT(vcpu)
   end if

end procedure
Demo: Emergency Vulnerability Detector

- Single kernelspace probe, added at runtime
- One **HOST** and one **GUEST**
- “notroot” user in **GUEST** escalates privileges

![Diagram showing the flow of operations between HOST and GUEST systems.](image-url)
Checkpoint/Restart In Userspace

Checkpoint two test applications
### Checkpoint/Restart In Userspace

Checkpoint two test applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Runtime (s)</th>
<th>95% CI (s)</th>
<th>% overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>F@H Normal</td>
<td>0.221</td>
<td>0.00922</td>
<td>0</td>
</tr>
<tr>
<td>F@H w/hprobe</td>
<td>0.228</td>
<td>0.0122</td>
<td>3.30</td>
</tr>
<tr>
<td>F@H w/Naïve</td>
<td>0.253</td>
<td>0.00851</td>
<td>14.4</td>
</tr>
<tr>
<td>pi-qmc Normal</td>
<td>0.137</td>
<td>0.00635</td>
<td>0</td>
</tr>
<tr>
<td>pi-qmc w/hprobe</td>
<td>0.140</td>
<td>0.00736</td>
<td>1.73</td>
</tr>
<tr>
<td>pi-qmc w/Naïve</td>
<td>0.152</td>
<td>0.00513</td>
<td>11.1</td>
</tr>
</tbody>
</table>
Summary

- Hooks
- Lares, SIM
- Hprobes
  - How hprobes work
  - Performance
  - Demo
  - Emergency Detector
BACKUP
SIM Entry/Exit Gates

ENTRY GATE
- CLI
- PUSH A
- MOV EAX, SIM_SHADOW
- MOV CR3, EAX

EXIT GATE
- MOV ESP, [P_ESP]
- MOV EAX, P_SHADOW
- MOV CR3, EAX
- POP A
- STI
- JMP RETURN_POINT

Invocation hook
Address space switch
Why hprobes vs. OS-level?
Why hprobes vs. OS-level?

- Isolation/Protection
Why hprobes vs. OS-level?

- Isolation/Protection
- More views
Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
  - Less variance in hypervisors
Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
  - Less variance in hypervisors
  - Reliability-as-Service
Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
  - Less variance in hypervisors
  - Reliability-as-Service
- Unikernel/OSv
“Hello World” Detector

Heartbeat/watchdog

- App-level
“Hello World” Detector

Heartbeat/watchdog
- App-level
- Critical code sections
“Hello World” Detector

Heartbeat/watchdog

- App-level
- Critical code sections
- Act?
Heartbeat/watchdog

App

Detector
Heartbeat/watchdog

App

Detector

Insert Probe

reset timer

timer expires

declare failure
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit
Heartbeat/watchdog

- Insert Probe
- Probe Hit
- reset timer

**App**

**Detector**
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

Probe Hit

reset timer
Heartbeat/watchdog

App

Detector

Probe Hit

reset timer

reset timer

Insert Probe
Heartbeat/watchdog

App

Detector

Insert Probe

Probe Hit

Probe Hit

reset timer

reset timer
Heartbeat/watchdog

App

Detector

- Insert Probe
- Probe Hit
- Probe Hit
- reset timer
- reset timer
- timer expires
- declare failure
Watchdog - Performance

PI-QMC Main Loop Runtime

- **No Detector**
- **With Detector**

<table>
<thead>
<tr>
<th>Internal Sample Loop Size</th>
<th>Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td></td>
</tr>
<tr>
<td>26000</td>
<td></td>
</tr>
<tr>
<td>28000</td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>32000</td>
<td></td>
</tr>
<tr>
<td>34000</td>
<td></td>
</tr>
</tbody>
</table>
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

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

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>
Conclusions
VMI

- Aims to achieve high fidelity monitoring while maintaining monitor fidelity
- Simplifies writing memory introspection based VMI monitors
- Passive monitoring has some inherent weaknesses
HW Invariant Techniques

- Ties high-level operations to fundamental HW functionality
- Protected against in-VM attackers
- Limited in scope?
Hook-Based Techniques

- Detectors are invoked on execution events
- Simple and flexible
- But... you need to understand execution
## VM Monitoring Overview

<table>
<thead>
<tr>
<th>HyperTap</th>
<th>LiveWire</th>
<th>LibVM</th>
<th>SIM</th>
<th>Larens</th>
<th>Lycosid</th>
<th>Antfarm</th>
<th>Nitro</th>
<th>Ether</th>
<th>Virtuoso</th>
<th>VMST</th>
<th>Txlintro</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(TBD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(HPC4'14)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>HyperTap</th>
<th>LiveWire</th>
<th>LibVM</th>
<th>SIM</th>
<th>Larens</th>
<th>Lycosid</th>
<th>Antfarm</th>
<th>Nitro</th>
<th>Ether</th>
<th>Virtuoso</th>
<th>VMST</th>
<th>Txlintro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root-of-trust (invariant)</td>
<td>HW</td>
<td>OS</td>
<td>OS</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>
<td>OS</td>
</tr>
<tr>
<td>Active/Polling Mon.</td>
<td>A</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>A</td>
<td>A</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</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>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Custom Auditors</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Online Detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Auto-generate Invariants</td>
<td>✗</td>
<td>✗</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>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Disclaimer: there are some gray areas!
Thank You!

Summary

▶ Key Concepts
▶ Validation
▶ Virtual Machine Introspection
▶ Hardware Invariant Techniques
▶ Hook-based Monitoring