

Space Traveling across VM

Automatically Bridging the Semantic-Gap in Virtual Machine
Introspection via Online Kernel Data Redirection

Yangchun Fu, and **Zhiqiang Lin**

Department of Computer Sciences
The University of Texas at Dallas

May 23rd, 2012

Outline

- 1 Background and The Problem
- 2 State-of-the-Art
- 3 Our Approach: Data Space Traveling
- 4 Conclusion

Cloud Runs Virtual Machines (VM)

Windows XP



• •

Linux



Win-7



Virtualization Layer

Hardware Layer

Cloud Runs Virtual Machines (VM)

Windows XP



Linux



Win-7



• •

Virtualization Layer

Hardware Layer

Consolidation,
Multiplexing, Migration,
Isolation, Encapsulation,
Interposition, **Security**,
Reliability, Dependability

...

Cloud Runs Virtual Machines (VM)

Windows XP



••

Linux



Win-7



Virtualization Layer

Hardware Layer

Consolidation,
Multiplexing, Migration,
Isolation, Encapsulation,
Interposition, **Security**,
Reliability, Dependability

...

VMI [Garfinkel and Rosenblum,
NDSS'03]

Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum]

A Trusted OS



Linux



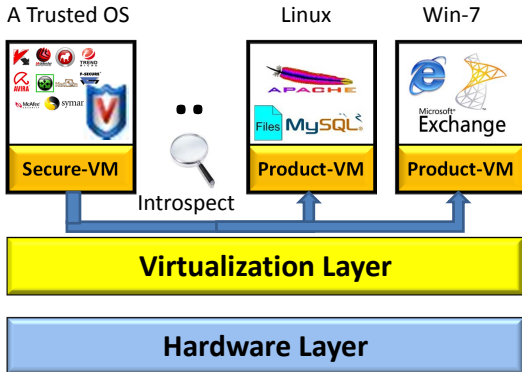
Win-7



Introspect

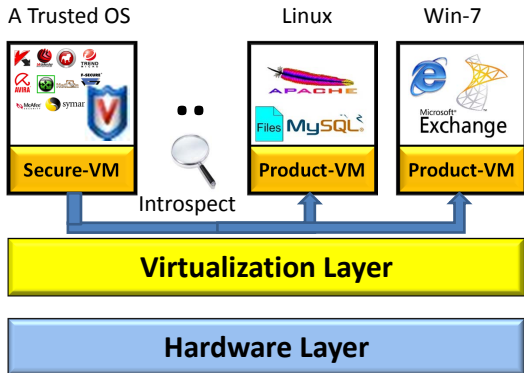


Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum]



Using a **trusted**, **isolated**, **dedicated** VM to **monitor** other VMs

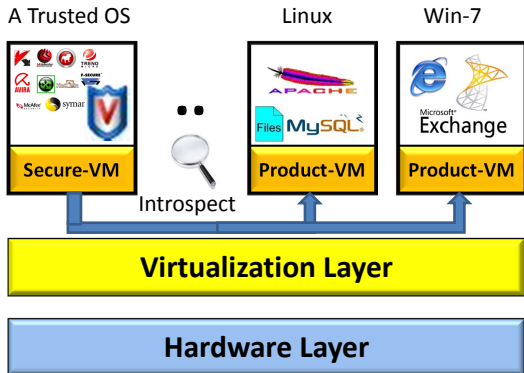
Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum]



Using a **trusted, isolated, dedicated** VM to **monitor** other VMs

- Intrusion Detection
- Malware Analysis
- Memory Forensics

Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum]



Using a **trusted, isolated, dedicated** VM to **monitor** other VMs

- Intrusion Detection
- Malware Analysis
- Memory Forensics

Semantic Gap Problem

The Semantic Gap in VMI ([Chen and Noble HotOS'01])

A Trusted OS



Linux



Introspect



Semantic Gap

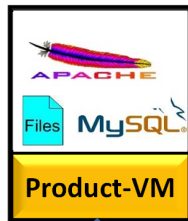


The Semantic Gap in VMI (Chen and Noble HotOS'01)

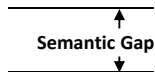
A Trusted OS



Linux



Introspect



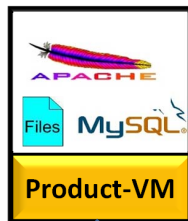
- View exposed by Virtual Machine Monitor is at low-level

The Semantic Gap in VMI (Chen and Noble HotOS'01)

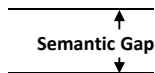
A Trusted OS



Linux



Introspect



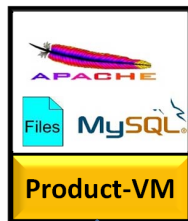
- View exposed by Virtual Machine Monitor is at low-level
- There is no abstraction and no APIs

The Semantic Gap in VMI (Chen and Noble HotOS'01)

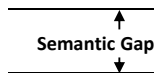
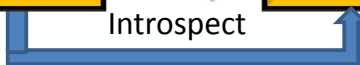
A Trusted OS



Linux

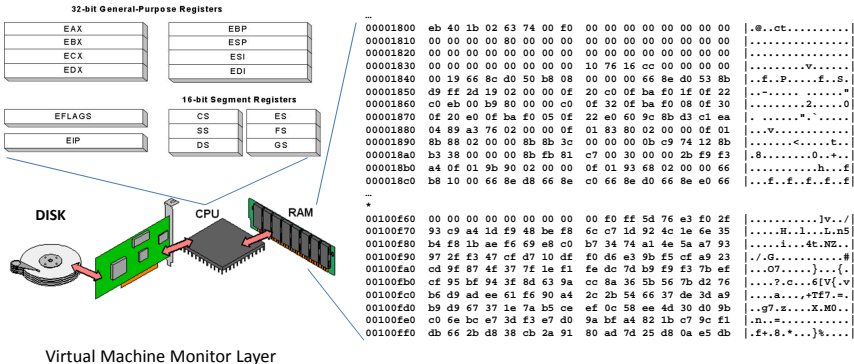


Introspect

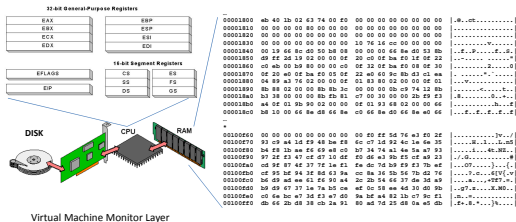


- View exposed by Virtual Machine Monitor is at low-level
- There is no abstraction and no APIs
- Need to reconstruct the guest-OS abstraction

Example: Inspect `pids` of Guest Memory from VMM



Example: Inspect `pid_t`s of Guest Memory from VMM



In Kernel 2.6.18

```


struct task_struct {
    ...
    [188] pid_t pid;
    [192] pid_t tgid;
    ...
    [356] uid_t uid;
    [360] uid_t euid;
    [364] uid_t suid;
    [368] uid_t fsuid;
    [372] gid_t gid;
    [376] gid_t egid;
    [380] gid_t sgid;
    [384] gid_t fsgid;
    ...
    [428] char comm[16];
    ...
}
SIZE: 1408

```


State-of-the-art



State-of-the-art

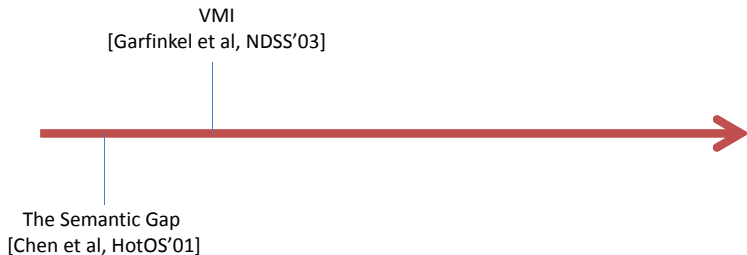


The Semantic Gap
[Chen et al, HotOS'01]

- In HotOS'01, Chen and Noble first raised **the semantic gap problem** in virtualization

“Services in the VM operate **below the abstractions** provided by the guest OS ... This can make it difficult to provide services.”

State-of-the-art

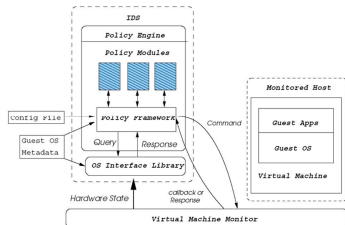


State-of-the-art

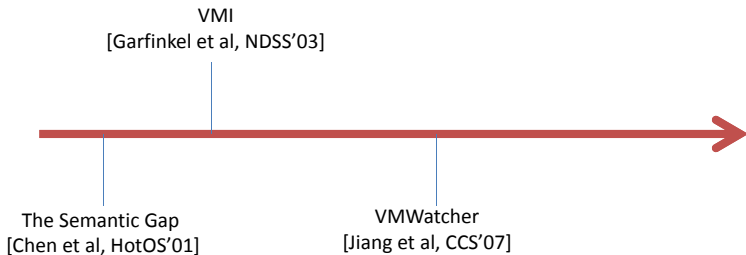
VMI
[Garfinkel et al, NDSS'03]

The Semantic Gap
[Chen et al, HotOS'01]

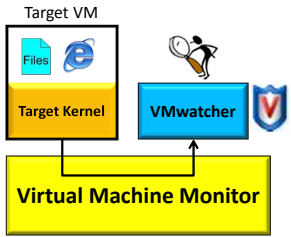
- In NDSS'03, Garfinkel et al. first proposed VMI, demonstrated for IDS
- Introspection routine is based on crash utility



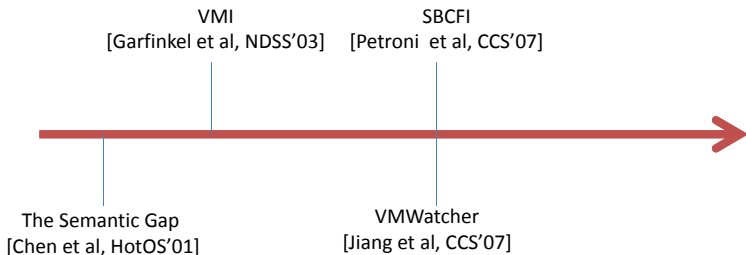
State-of-the-art



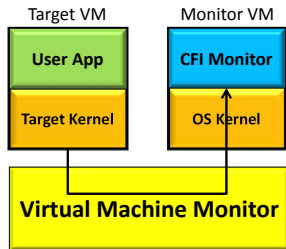
- In CCS'07, Jiang et al. proposed VMwatcher
- Introspection routine is based on manually created code



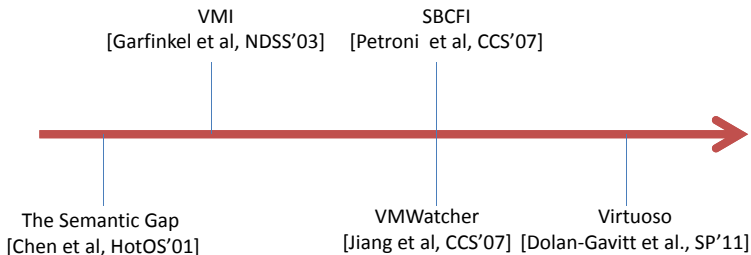
State-of-the-art



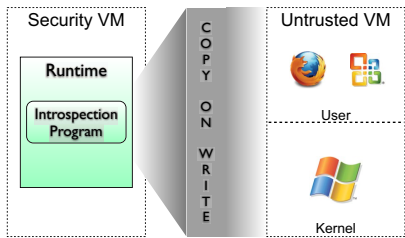
- In CCS'07, Petroni et al. proposed SBCFI
- Introspection routine is based on customized kernel source code



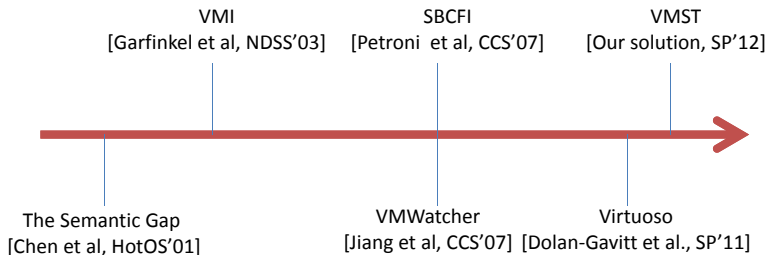
State-of-the-art



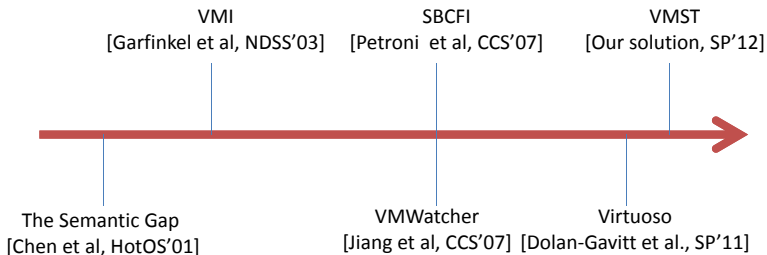
- In SP'11, Dolan-Gavitt et al. proposed Virtuoso
- Introspection routine is based on the trained user level and kernel level code



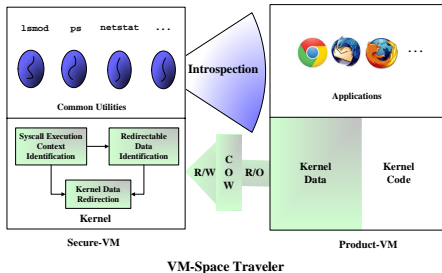
State-of-the-art



State-of-the-art



- In SP'12, we propose VM Space Traveler (VMST).
- Introspection routine is automatically generated from the **native** user level and kernel level code



Key Idea

Data can be transferred

- In Internet, data is transferred though network packet



Key Idea

Data can be transferred

- In Internet, data is transferred though network packet



Insight

An inspection program $\mathcal{P}(\mu, k)$ is often composed of static binary code \mathcal{P} , runtime dynamic user-level data μ (including user-level stack, heap, and global variables), and inspected kernel data k .

Key Idea

Data can be transferred

- In Internet, data is transferred though network packet



Insight

An inspection program $\mathcal{P}(\mu, k)$ is often composed of static binary code \mathcal{P} , runtime dynamic user-level data μ (including user-level stack, heap, and global variables), and inspected kernel data k .

- Transfer kernel space data k from one machine to the other

Key Idea

Data can be transferred

- In Internet, data is transferred though network packet



Insight

An inspection program $\mathcal{P}(\mu, k)$ is often composed of static binary code \mathcal{P} , runtime dynamic user-level data μ (including user-level stack, heap, and global variables), and inspected kernel data k .

- Transfer kernel space data k from one machine to the other
- `mov eax, [0x1c0eff08]`



Principles

Principles

$\mathcal{P}'(\mu, k) = \mathcal{P}(\mu, k')$, where

- \mathcal{P}' is the new introspection program
- \mathcal{P} is the old inspection program
- μ is the user level data
- k is the kernel data being inspected
- k' is from other machine

Principles

Principles

$\mathcal{P}'(\mu, k) = \mathcal{P}(\mu, k')$, where

- \mathcal{P}' is the new introspection program
- \mathcal{P} is the old inspection program
- μ is the user level data
- k is the kernel data being inspected
- k' is from other machine

Outcome

We reuse legacy binary code of \mathcal{P} to automatically generate new program \mathcal{P}'

How?

How?

strace of a getpid program

```
1 execve("./getpid",..) = 0
2 brk(0)          = 0x83b8000
3 access("/etc/ld.so.nohwcap",..) = -1
23 getpid()       = 13849
26 write(1, "pid=13849\n", 10) = 10
27 exit_group(0) = ?
```

How?

strace of a getpid program

```
1 execve("./getpid",..) = 0
2 brk(0)          = 0x83b8000
3 access("/etc/ld.so.nohwcap",..) = -1
23 getpid()       = 13849
26 write(1, "pid=13849\n", 10) = 10
27 exit_group(0)  = ?
```

Three Key Components

- Syscall execution context identification

How?

strace of a getpid program

```
1 execve("./getpid",..) = 0
2 brk(0)          = 0x83b8000
3 access("/etc/ld.so.nohwcap",..) = -1
23 getpid()      = 13849
26 write(1, "pid=13849\n", 10) = 10
27 exit_group(0) = ?
```

Three Key Components

- Syscall execution context identification
- Redirectable data identification

How?

strace of a getpid program

```
1 execve("./getpid",..) = 0
2 brk(0)          = 0x83b8000
3 access("/etc/ld.so.nohwcap",..) = -1
23 getpid()       = 13849
26 write(1, "pid=13849\n", 10) = 10
27 exit_group(0)  = ?
```

Three Key Components

- Syscall execution context identification
- Redirectable data identification
- Kernel data redirection

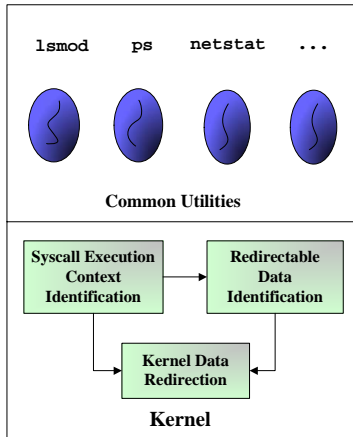
How?

strace of a getpid program

```
1 execve("./getpid",..) = 0
2 brk(0)          = 0x83b8000
3 access("/etc/ld.so.nohwcap",..) = -1
23 getpid()       = 13849
26 write(1, "pid=13849\n", 10) = 10
27 exit_group(0)  = ?
```

Three Key Components

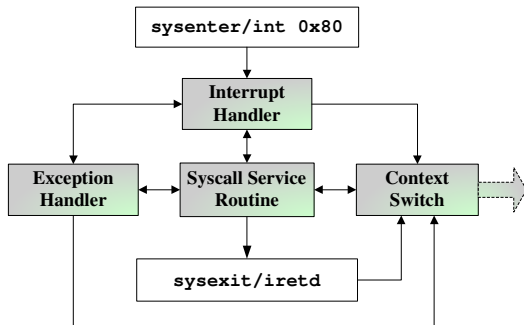
- Syscall execution context identification
- Redirectable data identification
- Kernel data redirection



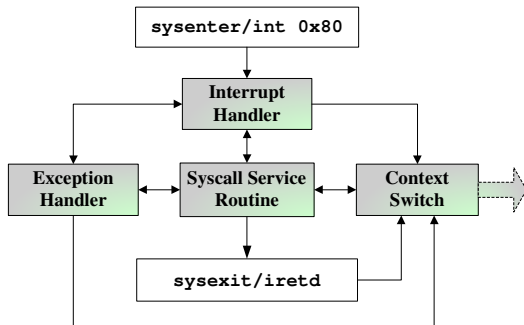
Secure-VM

I. Syscall Execution Context Identification

I. Syscall Execution Context Identification



I. Syscall Execution Context Identification

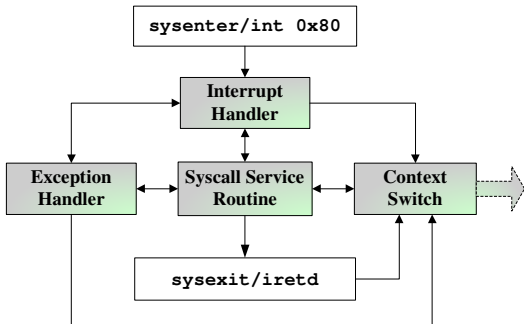


One intuitive approach

Hard-code all the starting and ending PC of

- Interrupt
- Exception
- Context switch

I. Syscall Execution Context Identification



One intuitive approach

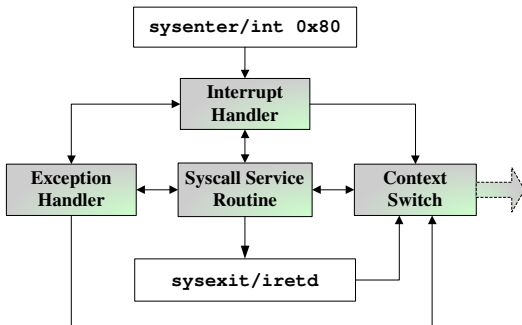
Hard-code all the starting and ending PC of

- Interrupt
- Exception
- Context switch

Our OS-agnostic solution

- Instrument VMM interrupt/exception handler to capture the starting and ending point of interrupt/exception

I. Syscall Execution Context Identification



One intuitive approach

Hard-code all the starting and ending PC of

- Interrupt
- Exception
- Context switch

Our OS-agnostic solution

- Instrument VMM interrupt/exception handler to capture the starting and ending point of interrupt/exception
- Disable the context switch by disabling the timer

II. Redirectable Data Identification

II. Redirectable Data Identification

Challenges

- Identify kernel stack data (kernel control flow related)

II. Redirectable Data Identification

Challenges

- Identify kernel stack data (kernel control flow related)
- Differentiate kernel stack, heap, and global variable

II. Redirectable Data Identification

Challenges

- Identify kernel stack data (kernel control flow related)
- Differentiate kernel stack, heap, and global variable
- Differentiate kernel code and data

II. Redirectable Data Identification

Challenges

- Identify kernel stack data (kernel control flow related)
- Differentiate kernel stack, heap, and global variable
- Differentiate kernel code and data

Our solution: a variant of dynamic data flow analysis

- Identify the kernel global and kernel heap (derived from kernel global), and **redirect** their memory access

II. Redirectable Data Identification

Challenges

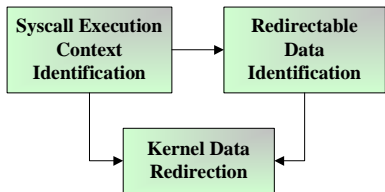
- Identify kernel stack data (kernel control flow related)
- Differentiate kernel stack, heap, and global variable
- Differentiate kernel code and data

Our solution: a variant of dynamic data flow analysis

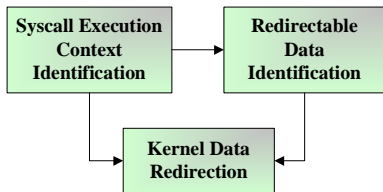
- Identify the kernel global and kernel heap (derived from kernel global), and **redirect** their memory access
- **Alternatively, identify only the stack variable** (derived from `esp`), and **no redirection** for them.

III. Kernel Data Redirection

III. Kernel Data Redirection



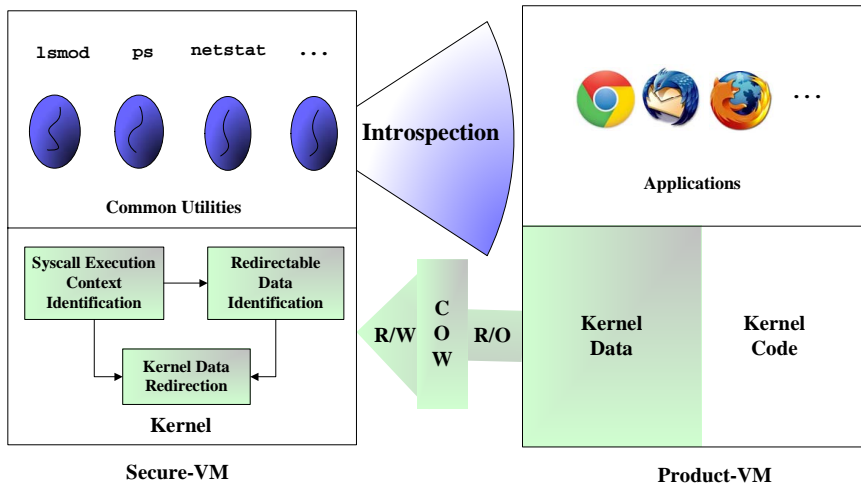
III. Kernel Data Redirection



The Algorithm

```
1: DynamicInstInstrument(i):
2:   if SysExecContext(s):
3:     if SysRedirect(s):
4:       RedirectableDataTracking(i);
5:       for  $\alpha$  in MemoryAddress(i):
6:         if DataRead( $\alpha$ ):
7:            $PA(\alpha) \leftarrow V2P(\alpha)$ 
8:           Load( $PA(\alpha)$ )
9:         else:
10:          if NotDirty( $\alpha$ ):
11:            CopyOnWritePage( $\alpha$ )
12:            UpdatePageEntryInSTLB( $\alpha$ )
13:             $PA(\alpha) \leftarrow V2P(\alpha)$ 
14:            Store( $PA(\alpha)$ )
```


Architecture

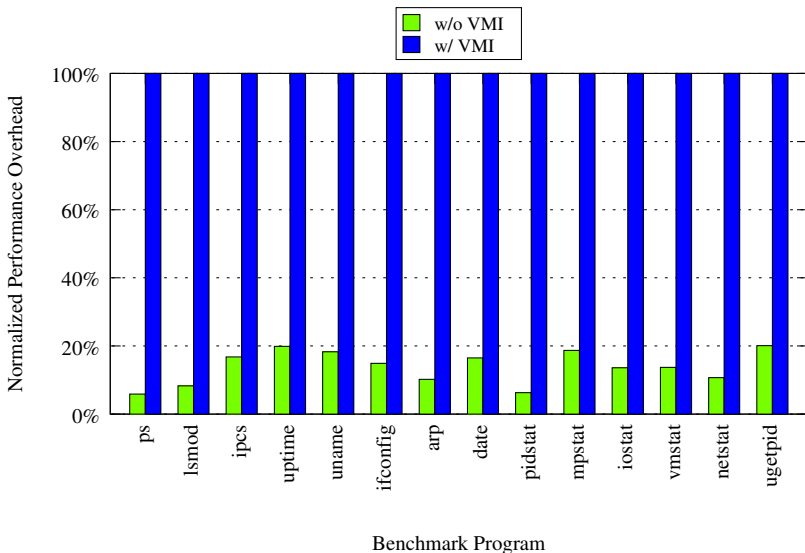


VM-Space Traveler

Automatic VMI Tool Generation

Utilities w/ options	Description	Syntax? (diff)	Semantics? (Manual)
<code>ps -A</code>	Reports a snapshot of all processes	✗	✓
<code>lsmod</code>	Shows the status of modules	✓	✓
<code>lsof -c p</code>	Lists opened files by a process p	✓	✓
<code>ipcs</code>	Displays IPC facility status	✓	✓
<code>netstat -s</code>	Displays network statistics	✓	✓
<code>uptime</code>	Reports how long the system running	✗	✓
<code>ifconfig</code>	Reports network interface parameters	✓	✓
<code>uname -a</code>	Displays system information	✓	✓
<code>arp</code>	Displays ARP tables	✓	✓
<code>free</code>	Displays amount of free memory	✗	✓
<code>date</code>	Print the system date and time	✗	✓
<code>pidstat</code>	Reports statistics for Linux tasks	✗	✓
<code>mpstat</code>	Reports CPU related statistics	✗	✓
<code>iostat</code>	Displays I/O statistics	✗	✓
<code>vmstat</code>	Displays VM statistics	✗	✓

Performance Overhead



OS-Agnostic Testing

Linux Distribution	Kernel Version	Release Date	OS-agnostic?	LOC
Redhat-9	2.4.20-31	11/28/2002	✗	53
Fedora-6	2.6.18-1.2798.fc6	10/14/2006	✗	53
Fedora-15	2.6.38.6-26.rc1.fc15	05/09/2011	✓	0
OpenSUSE-11.3	2.6.34-12-default	09/13/2010	✓	0
	2.6.35	08/10/2010	✓	0
OpenSUSE-11.4	2.6.37.1-1.2-default	02/17/2011	✓	0
	2.6.39.4	08/03/2011	✓	0
Debian 3.0	2.4.27-3	08/07/2004	✗	53
Debian 4.0	2.6.18-6	12/17/2006	✗	53
Debian 6.0	2.6.32-5	01/22/2010	✓	0
	2.6.32-rc8	02/09/2010	✓	0
Ubuntu-4.10	2.6.8.1-3	08/14/2004	✗	53
Ubuntu-5.10	2.6.12-9	08/29/2005	✗	53
Ubuntu-10.04	2.6.32.27	12/09/2010	✓	0
	2.6.33	03/15/2010	✓	0
	2.6.34	07/05/2010	✓	0
	2.6.36	11/22/2010	✓	0
	2.6.37.6	03/27/2010	✓	0
Ubuntu-11.04	2.6.38-8-generic	06/03/2011	✓	0
Ubuntu-11.10	3.0.0-12-generic	08/05/2011	✓	0

Limitations and Future Work

Limitations

- Need an identical trusted kernel
- Not entirely transparent to arbitrary OS kernels (relies on syscall knowledge)
- Non-blocking system call
- Does not inspect any disk data, memory swapped to disk

Limitations and Future Work

Limitations

- Need an identical trusted kernel
- Not entirely transparent to arbitrary OS kernels (relies on syscall knowledge)
- Non-blocking system call
- Does not inspect any disk data, memory swapped to disk

Future Work

- Kernel version inference in cloud VM
- Porting to Windows OS
- Addressing the non-blocking issue

Conclusion

- VMST has **automatically bridged the semantic gap**, and **automatically generated the introspection tools** by reusing the legacy code (no training involved)

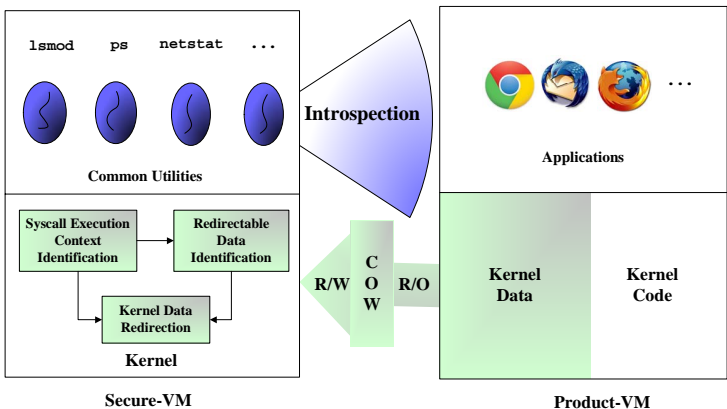
Conclusion

- VMST has **automatically bridged the semantic gap**, and **automatically generated the introspection tools** by reusing the legacy code (no training involved)
- It also enables **native VMI tool development**.

Conclusion

- VMST has **automatically bridged the semantic gap**, and **automatically generated the introspection tools** by reusing the legacy code (no training involved)
- It also enables **native VMI tool development**.
- (We hope) Cloud/VM/OS Providers, and AV-Software Vendors, could benefit from our techniques (for **VMI and memory forensics**).

Thank You



zhiqiang.lin@utdallas.edu