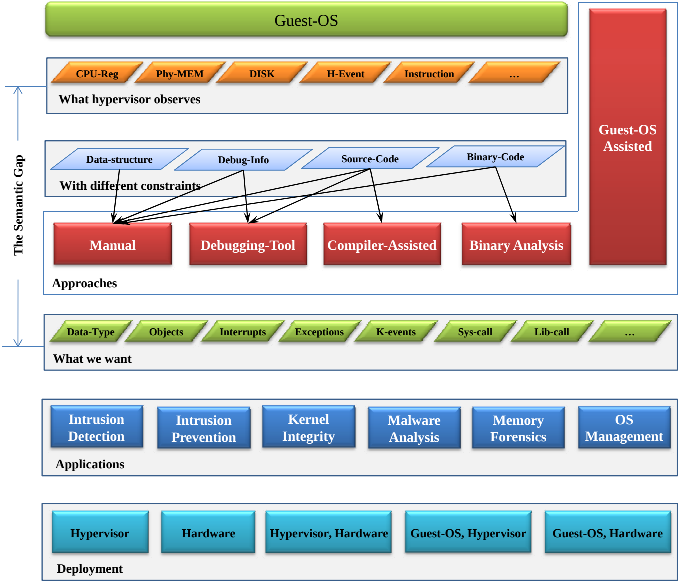
All You Ever Wanted to Know About Virtual Machine Introspection: The Semantic Gap Challenge

Zhiqiang Lin

Department of Computer Sciences
The University of Texas at Dallas

August 24^{th}, 2015

The Road Map


Guest-OS
CPU-Reg
Phy-MEM
DISK
H-Event
Instruction
...
What hypervisor observes
Data-structure
Debug-Info
Source-Code
Binary-Code
With different constraints
Manual
Debugging-Tool
Compiler-Assisted
Binary Analysis
Guest-OS Assisted
Approaches
The Semantic Gap
Data-Type
Objects
Interrupts
Exceptions
K-events
Sys-call
Lib-call
...
What we want
Intrusion Detection
Intrusion Prevention
Kernel Integrity
Malware Analysis
Memory Forensics
OS Management
Applications
Hypervisor
Hardware
Hypervisor, Hardware
Guest-OS, Hypervisor
Guest-OS, Hardware
Deployment

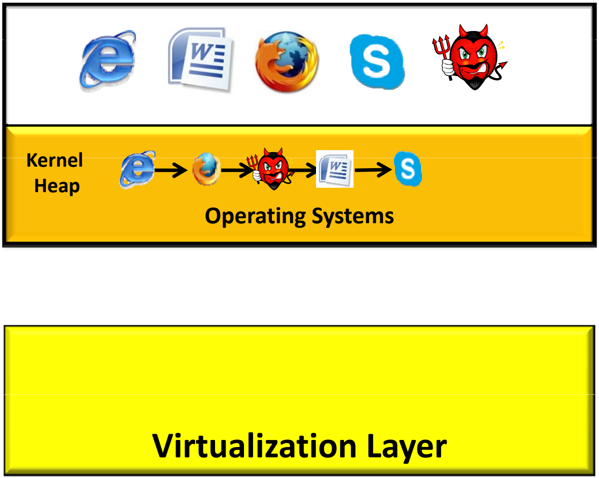
Outline

Outline

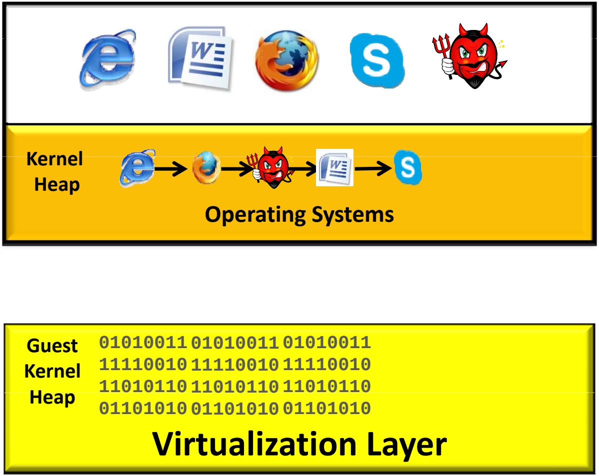
The Semantic Gap

The Semantic Gap

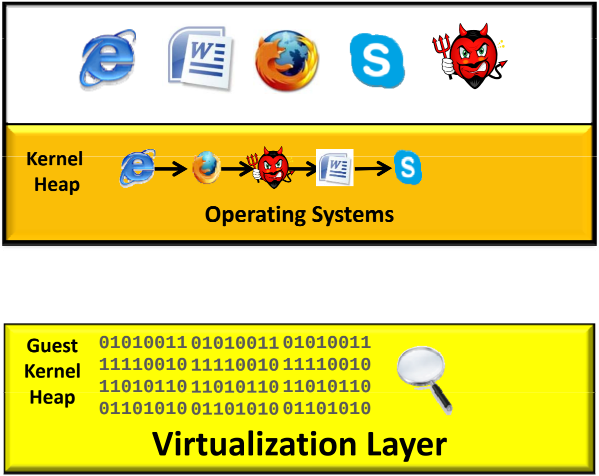
The Semantic Gap



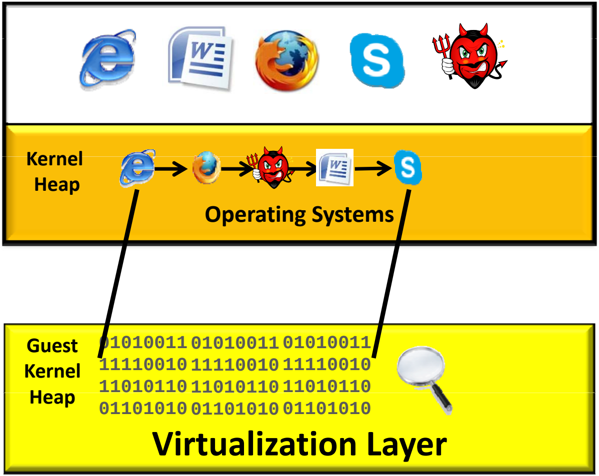
The Semantic Gap



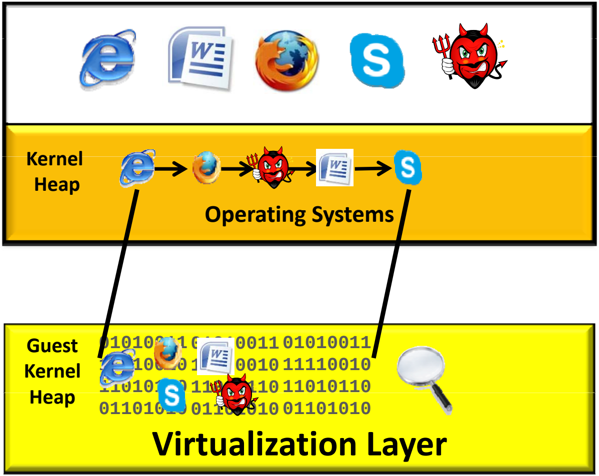
The Semantic Gap



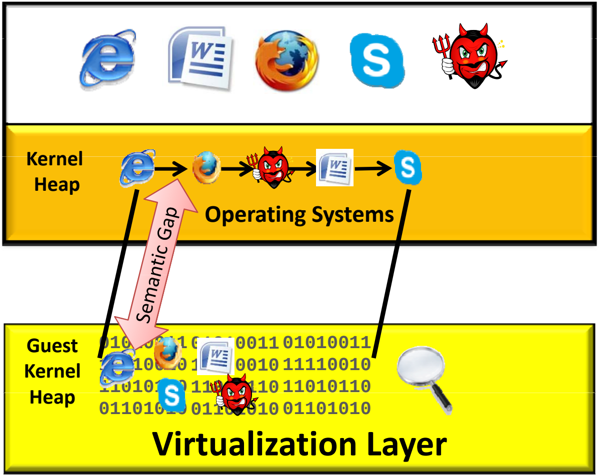
The Semantic Gap



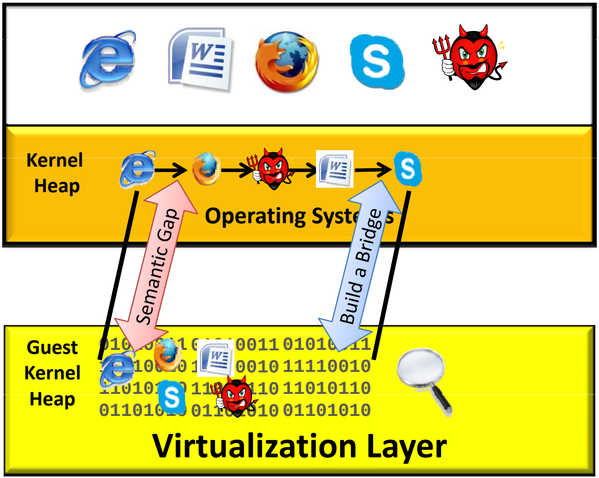
The Semantic Gap



The Semantic Gap



The Semantic Gap



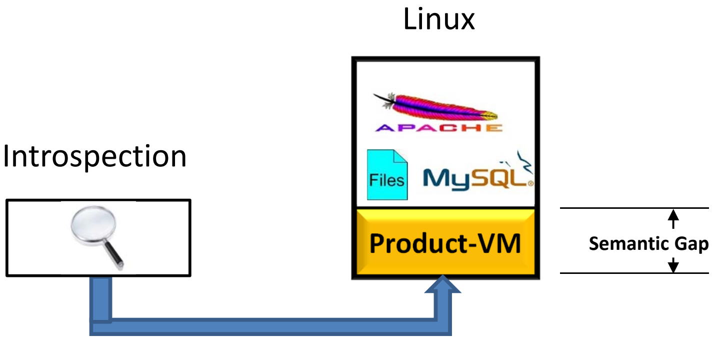
The Semantic Gap

- The primary advantage that in-VM systems have is their direct access to all kinds of OS-level abstractions. However, when using a hypervisor, access to all of the rich semantic abstractions inside the OS is lost.

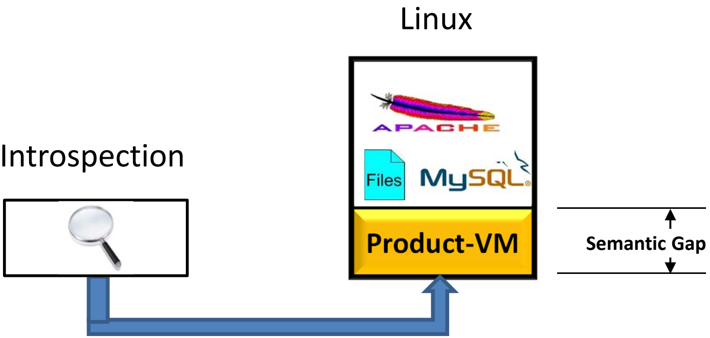
- Although hypervisors have a grand view of the entire state of the VMs they monitor, this grand view unfortunately consists of just ones and zeros with no context.

- Therefore, there is a semantic gap between **what we can observe** and **what we want**, and we must **bridge** it in order to provide effective monitoring services.

The Semantic Gap in Out-of-VM ([Chen and Noble HotOS'01])

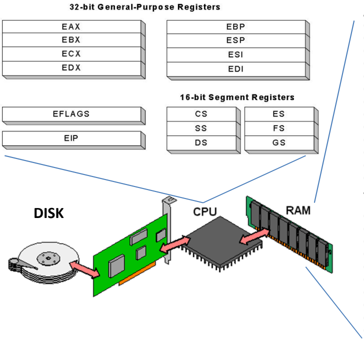


The Semantic Gap in Out-of-VM ([Chen and Noble HotOS'01])



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

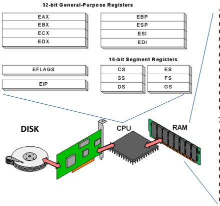
Example: Inspect `pids` of Guest Memory from VMM



Virtual Machine Monitor Layer

```
…
00001800  eb 40 1b 02 63 74 00 f0  00 00 00 00 00 00 00 00  |.@..ct..........|
00001810  00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00  |................|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001830  00 00 00 00 00 00 00 00  10 76 16 cc 00 00 00 00  |.........v......|
00001840  00 19 66 8c d0 50 b8 08  00 00 00 66 8e d0 53 8b  |..f..P.....f..S.|
00001850  d9 ff 2d 19 02 00 00 0f  20 c0 0f ba f0 1f 0f 22  |..-..... ......"|
00001860  c0 eb 00 b9 80 00 00 c0  0f 32 0f ba f0 08 0f 30  |.........2.....0|
00001870  0f 20 e0 0f ba f0 05 0f  22 e0 60 9c 8b d3 c1 ea  |. ......".`.....|
00001880  04 89 a3 76 02 00 00 0f  01 83 80 02 00 00 0f 01  |...v............|
00001890  8b 88 02 00 00 8b 8b 3c  00 00 00 0b c9 74 12 8b  |.......<.....t..|
000018a0  b3 38 00 00 00 8b fb 81  c7 00 30 00 00 2b f9 f3  |.8........0..+..|
000018b0  a4 0f 01 9b 90 02 00 00  0f 01 93 68 02 00 00 66  |...........h...f|
000018c0  b8 10 00 66 8e d8 66 8e  c0 66 8e d0 66 8e e0 66  |...f..f..f..f..f|
…
*
00100f60  00 00 00 00 00 00 00 00  00 f0 ff 5d 76 e3 f0 2f  |...........]v../|
00100f70  93 c9 a4 1d f9 48 be f8  6c c7 1d 92 4c 1e 6e 35  |.....H..l...L.n5|
00100f80  b4 f8 1b ae f6 69 e8 c0  b7 34 74 a1 4e 5a a7 93  |.....i...4t.NZ..|
00100f90  97 2f f3 47 cf d7 10 df  f0 d6 e3 9b f5 cf a9 23  |./.G...........#|
00100fa0  cd 9f 87 4f 37 7f 1e f1  fe dc 7d b9 f9 f3 7b ef  |...O7.....}...{.|
00100fb0  cf 95 bf 94 3f 8d 63 9a  cc 8a 36 5b 56 7b d2 76  |....?.c...6[V{.v|
00100fc0  b6 d9 ad ee 61 f6 90 a4  2c 2b 54 66 37 de 3d a9  |....a...,+Tf7.=.|
00100fd0  b9 d9 67 37 1e 7a b5 ce  ef 0c 58 ee 4d 30 d0 9b  |..g7.z....X.M0..|
00100fe0  c0 6e bc e7 3d f3 e7 d0  9a bf a4 82 1b c7 9c f1  |.n..=...........|
00100ff0  db 66 2b d8 38 cb 2a 91  80 ad 7d 25 d8 0a e5 db  |.f+.8.*...}%....|
```

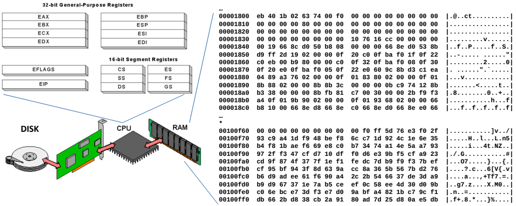
Example: Inspect `pid`s of Guest Memory from VMM




```
…
00001800  eb 40 1b 02 63 74 00 f0  00 00 00 00 00 00 00 00  |.@..ct..........|
00001810  00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00  |................|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001830  00 00 00 00 00 00 00 00  10 76 16 cc 00 00 00 00  |.........v......|
00001840  00 19 66 8c d0 50 b8 08  00 00 00 66 8e d0 53 8b  |..f..P.....f..S.|
00001850  d9 ff 2d 19 02 00 00 0f  20 c0 0f ba f0 1f 0f 22  |..-..... ......"|
00001860  c0 eb 00 b9 80 00 00 c0  0f 32 0f ba f0 08 0f 30  |.........2.....0|
00001870  0f 20 e0 0f ba f0 05 0f  22 e0 60 9c 8b d3 c1 ea  |. ......".`.....|
00001880  04 89 a3 76 02 00 00 0f  01 83 80 02 00 00 0f 01  |...v............|
00001890  8b 88 02 00 00 8b 8b 3c  00 00 00 0b c9 74 12 8b  |.......<.....t..|
000018a0  b3 38 00 00 00 8b fb 81  c7 00 30 00 00 2b f9 f3  |.8........0..+..|
000018b0  a4 0f 01 9b 90 02 00 00  0f 01 93 68 02 00 00 66  |...........h...f|
000018c0  b8 10 00 66 8e d8 66 8e  c0 66 8e d0 66 8e e0 66  |...f..f..f..f..f|
…
*
00100f60  00 00 00 00 00 00 00 00  00 f0 ff 5d 76 e3 f0 2f  |...........]v../|
00100f70  93 c9 a4 1d f9 48 be f8  6c c7 1d 92 4c 1e 6e 35  |.....H..l...L.n5|
00100f80  b4 f8 1b ae f6 69 e8 c0  b7 34 74 a1 4e 5a a7 93  |.....i...4t.NZ..|
00100f90  97 2f f3 47 cf d7 10 df  f0 d6 e3 9b f5 cf a9 23  |./.G...........#|
00100fa0  cd 9f 87 4f 37 7f 1e f1  fe dc 7d b9 f9 f3 7b ef  |...O7.....}...{.|
00100fb0  cf 95 bf 94 3f 8d 63 9a  cc 8a 36 5b 56 7b d2 76  |....?.c...6[V{.v|
00100fc0  b6 d9 ad ee 61 f6 90 a4  2c 2b 54 66 37 de 3d a9  |....a...,+Tf7.=.|
00100fd0  b9 d9 67 37 1e 7a b5 ce  ef 0c 58 ee 4d 30 d0 9b  |..g7.z....X.M0..|
00100fe0  c0 6e bc e7 3d f3 e7 d0  9a bf a4 82 1b c7 9c f1  |.n..=...........|
00100ff0  db 66 2b d8 38 cb 2a 91  80 ad 7d 25 d8 0a e5 db  |.f+.8.*...}%....|
```


Virtual Machine Monitor Layer

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

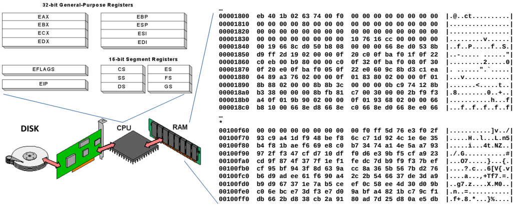


Virtual Machine Monitor Layer

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
```

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



Virtual Machine Monitor Layer

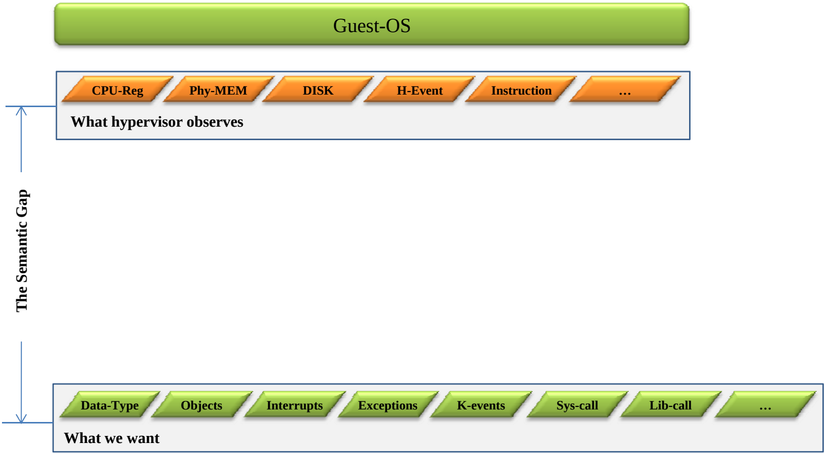
- Kernel specific data structure definition
- Kernel symbols (global variable)
- Virtual to physical (V2P) translation
- OS updates and patches can break the existing introspection utilities

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
```

Outline

What we can observe



What we can observe

From Native Hypervisor

1. **CPU Registers.** All of the CPU registers can be read by the hypervisor when it gains control because it runs at the highest privilege level.
2. **Guest OS Memory.** The entire guest OS memory state can also be observed. However, hypervisors only have access to physical addresses, which have to be translated to virtual addresses when accessing them.
3. **Hard Disk Contents.** Similar to the memory image, the content of the guest OS's disk image, if not encrypted, is also visible to the hypervisor.
4. **Hardware Events.** All hardware-level events, including timers, interrupts, and exceptions, can also be observed.
5. **I/O Traffic.** The hypervisor also oversees all the I/O traffic, including network traffic, disk I/O, and keystrokes.

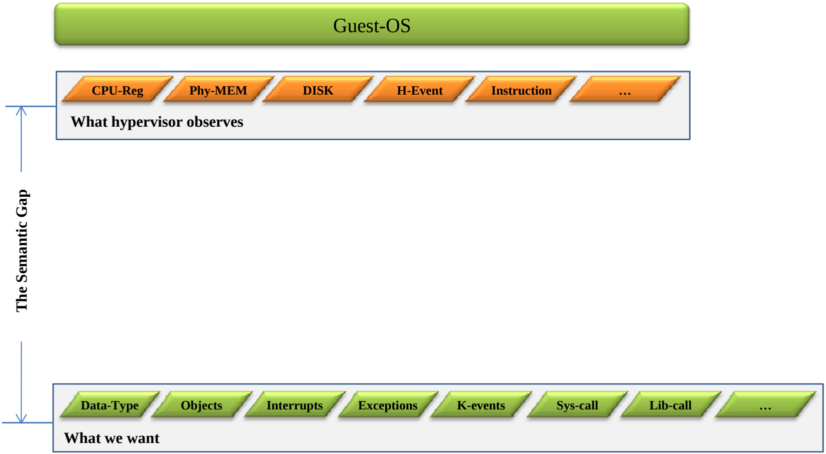
What we can observe

From Emulation-based Hypervisor

1. **Program Counter**. They can know which instructions get executed and their disassembly code.
2. **Instruction Opcode and Operand**. For each executed instruction, they can observe its opcode and operand.
3. **Control Flow Transfer**. All control flow transfers (e.g., `call/jmp/ret`, conditional branches) can be observed, along with their source and destination addresses if there are any.
4. **Call Stack**. The stack can be traversed if a stack frame pointer exists, or instructions can be transparently instrumented to build the call stack information.
5. **Context-Switch**. Each specific process or thread execution context can also be observed.

Outline

What we want


Guest-OS
CPU-Reg
Phy-MEM
DISK
H-Event
Instruction
...
What hypervisor observes
The Semantic Gap
Data-Type
Objects
Interrupts
Exceptions
K-events
Sys-call
Lib-call
...
What we want

What we want

Data State Abstraction (Snapshot View)

1. **Variables, Objects, and Virtual Address Spaces.** Given the physical memory of a guest OS, we want to know where kernel or monitored process variables (or objects) of interest are, and how to locate them.
2. **Data Structure Types and Their Connections.** We also would like to know object types, data structures, and their point-to relations
3. **File Systems and Files.** Given the disk image, we are interested in which type of file system is being used and where files are located.
4. **Interrupts, Exceptions, and Other Kernel Events.** For an observed hardware event, we would like to obtain additional details about it; for an interrupt or exception, we want to know which specific interrupt or exception it is.

What we want

Control State Abstraction (Contiguous View)

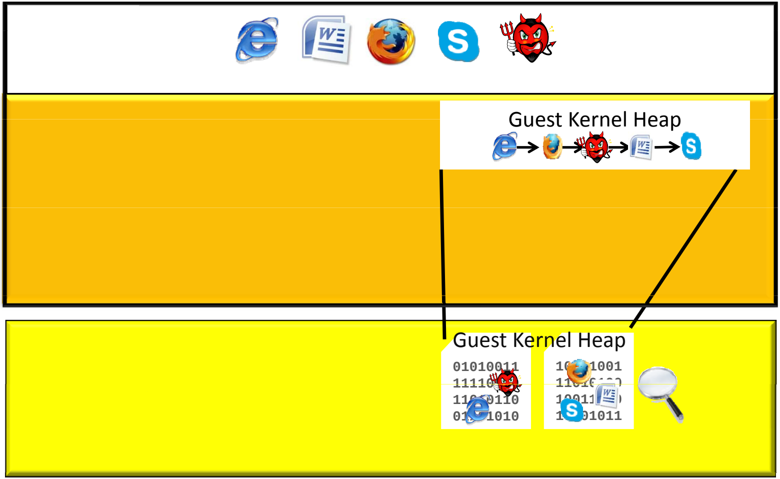
1. **Instructions, Control Path, and Call Stack.** Knowledge of which instruction the VM is executing, which control path it belongs to, and what the calling context is can help the out-of-VM monitor precisely understand the current execution context of the guest OS.
2. **Function Calls, System Calls, Library Calls, and Hooks.** As instruction-level monitoring usually significantly slows down the VM execution, we could instead monitor at the level of function call execution, or at certain system calls, library calls, or hooks of monitor interest.
3. **Processes, Threads, and Execution Context.** When there is a context switch, we would like to know which process (thread) is switched (from) to, as control flow is often thread specific.

Outline

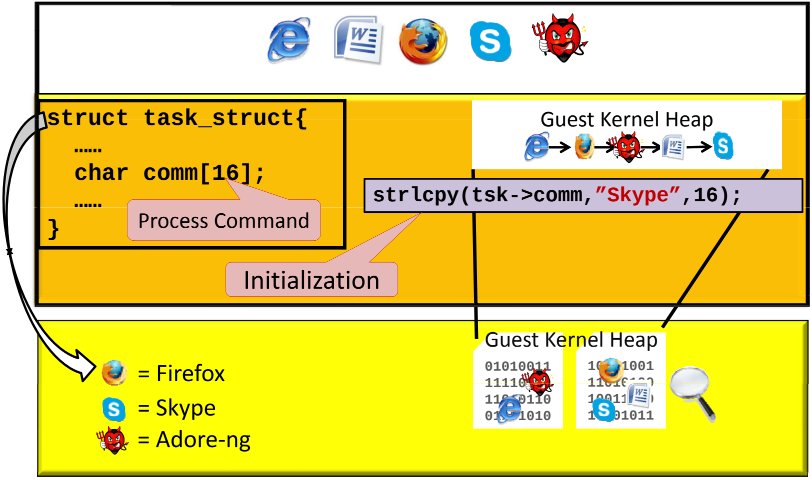
The Semantic Gap: Summary of the Different Views

	Snapshot-View					**Contiguous-View**				
What We Observe	CPU Registers	Physical Memory	Disk Data	Hardware Events	I/O Data	Program Counter	Opcode & Operand	Control Flow Transfer	Call-Stack	Context-Switch
Native Hypervisor	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
Emulation Hypervisor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Emulation Hypervisor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Native Hypervisor	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
What We Want	Variables,Objects	Variables and Types	File system and Files	Interrupt/Exceptions	Packets, Buffers	Instruction Semantics	Variables, Pointers	Calls, Hooks, Branches	Execution Context	Processes, Threads
	Data and Control State Abstractions									

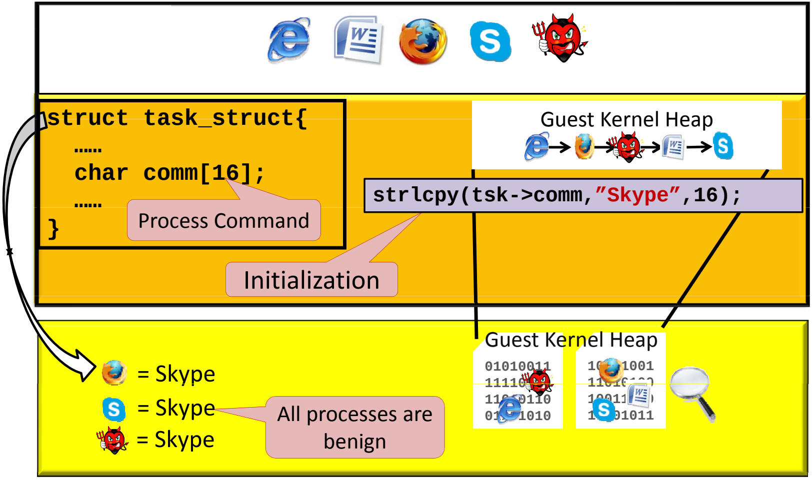
Kernel Can be Untrusted



Kernel Can be Untrusted



Kernel Can be Untrusted



On the trust of the semantic-gap [Jain et al, SP'14]

Challenge	App	Guest OS	Hypervisor
Semantic Gap	U	T	T

T Trusted

U Untrusted

On the trust of the semantic-gap [Jain et al, SP'14]

Challenge	App	Guest OS	Hypervisor
Semantic Gap	U	T	T
	U	U	T

T Trusted

U Untrusted

On the trust of the semantic-gap [Jain et al, SP'14]

Challenge	App	Guest OS	Hypervisor
Weak Semantic Gap	U	T	T
Strong Semantic Gap	U	U	T

T Trusted U Untrusted

On the trust of the semantic-gap [Jain et al, SP'14]

Challenge	App	Guest OS	Hypervisor
Weak Semantic Gap	U	T	T
Strong Semantic Gap	U	U	T
Untrusted Guest OS	T	U	T
Untrusted Cloud Hypervisor	T	T	U
Untrusted Guest OS and Hypervisor	T	U	U

T Trusted U Untrusted