Hypervisor Introspection:

A Technique for Evading Passive Virtual Machine Monitoring

Gary Wang[1,2], Zachary J. Estrada[1], Cuong Pham[1], Zbigniew Kalbarczyk[1], Ravishankar K. Iyer[1]

[1]University of Illinois at Urbana-Champaign, [2]MIT Lincoln Laboratory

USENIX WOOT ‘15

ECE ILLINOIS

ILLINOIS

Overview

- Motivation and goals
- Identifying and exploiting a side-channel
- Example attacks
- Defenses
- Conclusions

Motivation

- Security in the cloud, especially Infrastructure-as-a-Service (IaaS)
- Isolation in virtualized environments enables the cloud
- Do current techniques do enough to protect these environments?

Virtual machine introspection (VMI)

- Let's put security monitoring in the hypervisor!
  - Can observe guest VM's hardware state
- Active and passive monitoring
- This is not about the semantic gap problem

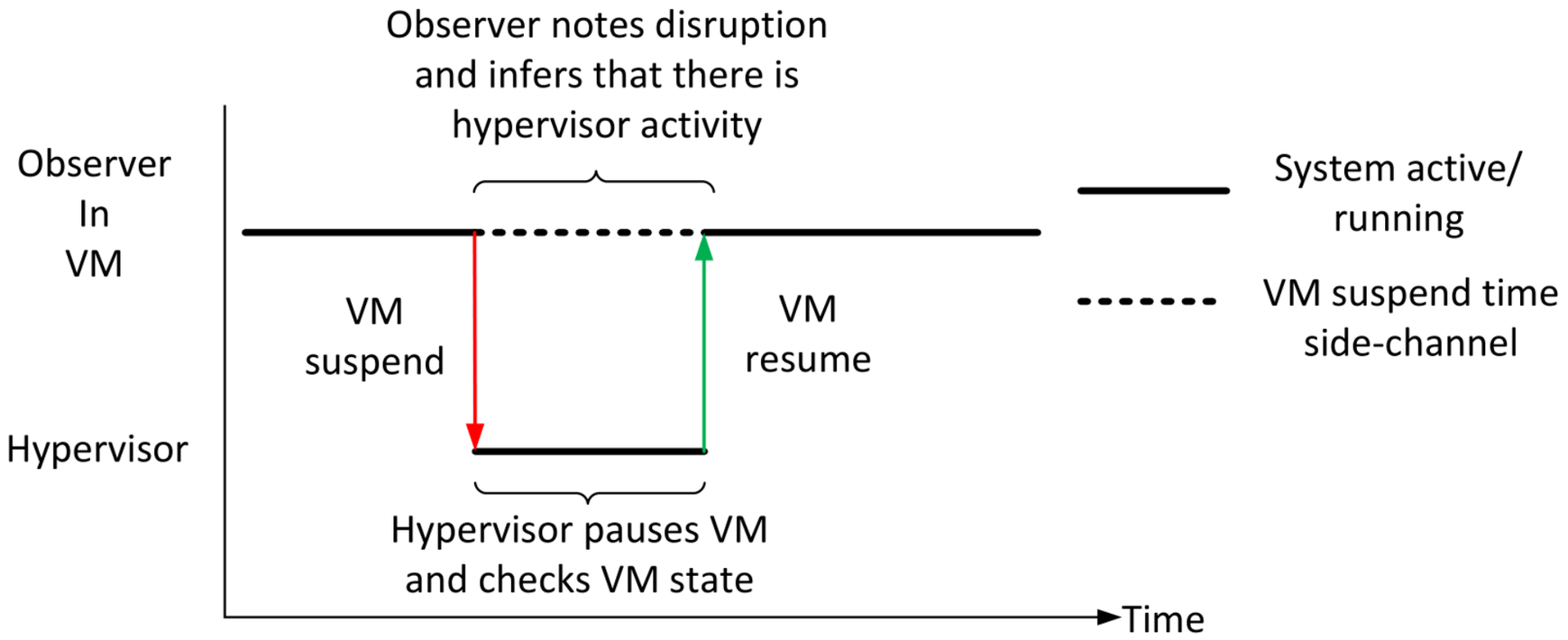
What we want to know

- Can we detect hypervisor-level monitoring from within the guest VM?
  - "Hypervisor introspection (HI)"
- If we can detect hypervisor-level monitoring, can we evade it?

Finding the side-channel

- Hypervisor activity suspends VM, which pauses all VM activity
  - Duration of VM suspend is duration of hypervisor activity
  - “VM suspend side-channel”

How do we detect VM suspends?



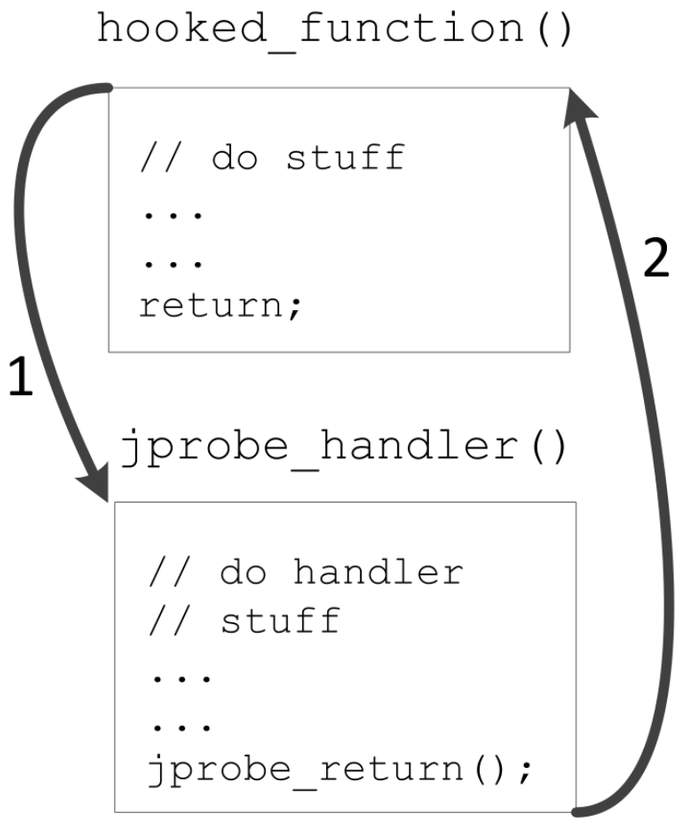
How do we detect VM suspends?

- Focus on host-based measurements
- Time frequently occurring events:
  - Disk reads
  - Process scheduling
  - Other possible events

How do we detect VM suspends?

- Leverage kernel probes (kprobes) to hook into relevant kernel functions
  - `sys_read`
  - `schedule`
- Utilize special kprobes (jprobes) to implement our timing functionality

What's a jprobe?



Our jprobe handler

- Calls `do_gettimeofday` and takes the difference with most recent timestamp
- Trigger alarm if difference is greater than threshold value of 5ms

HI kernel module output

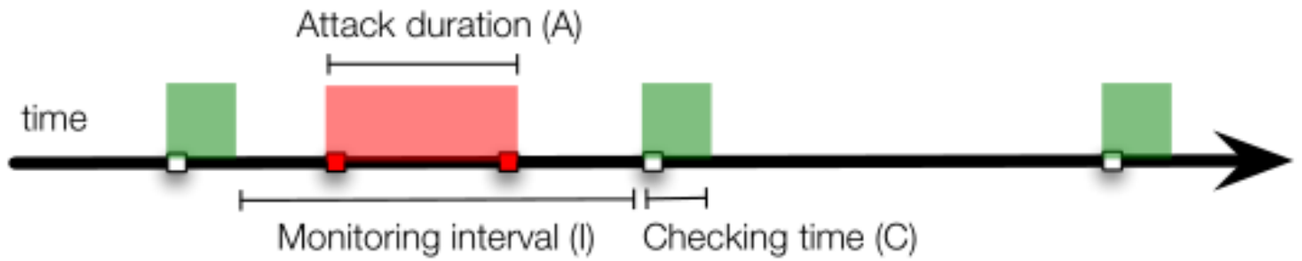
```
[ 2990.592923 <    0.005834>] scheduler_watcher: jprobe hooked into schedule() at addr c167b3a0
[ 2990.592925 <    0.000002>] scheduler_watcher: jprobe hooked into sys_read() at addr c1176740
[ 2990.820740 <    0.227815>] scheduler_watcher: pause > 5ms detected. TS: 1429246180828456
[ 2991.856721 <    1.035981>] scheduler_watcher: pause > 5ms detected. TS: 1429246181864436
[ 2992.908733 <    1.052012>] scheduler_watcher: pause > 5ms detected. TS: 1429246182916443
```

Limitations of HI

- Accuracy of measurements
  - Accurate up to polling rate of 10 Hz (check every 0.1s)
- Threshold for determining VM suspends
  - Requires empirical testing, but there is some leniency

How do we use HI?

- Additional reconnaissance for attackers
- Hide malicious activity from a passive VMI system



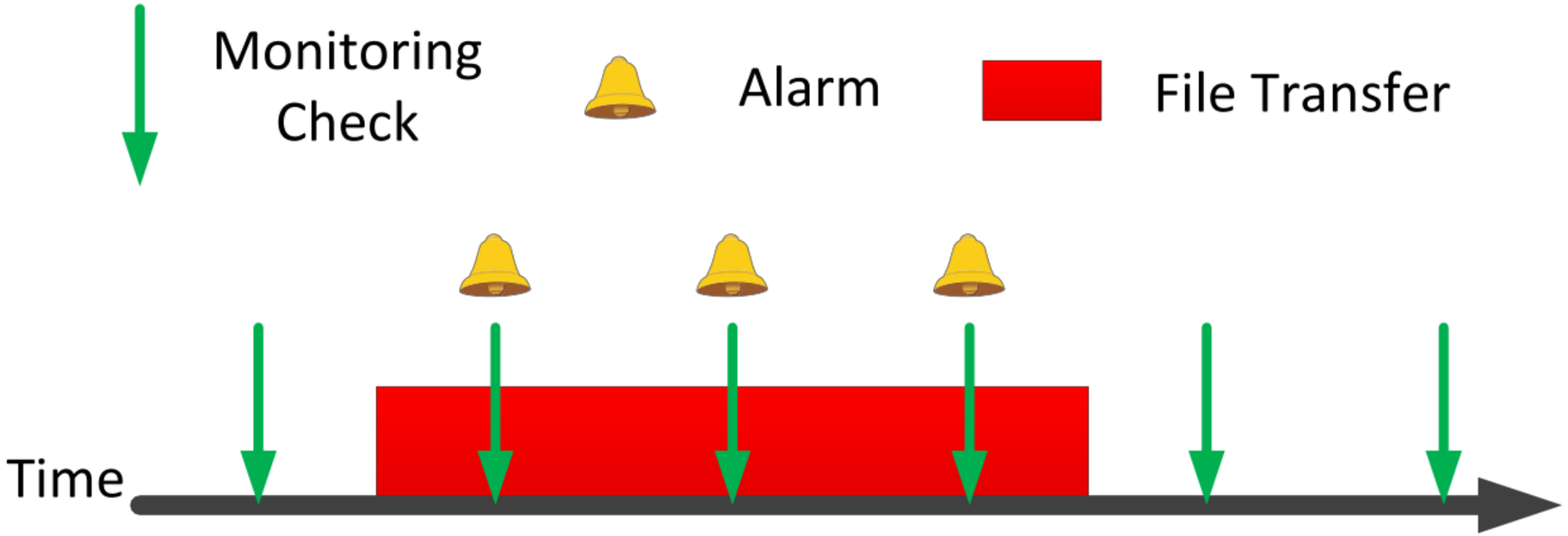
Attack model and assumptions

- Insider threat model
  - Attacker has root, but wishes to maintain presence on virtual systems after leaving
- VMI system checks for bad network connections
- No other monitoring systems

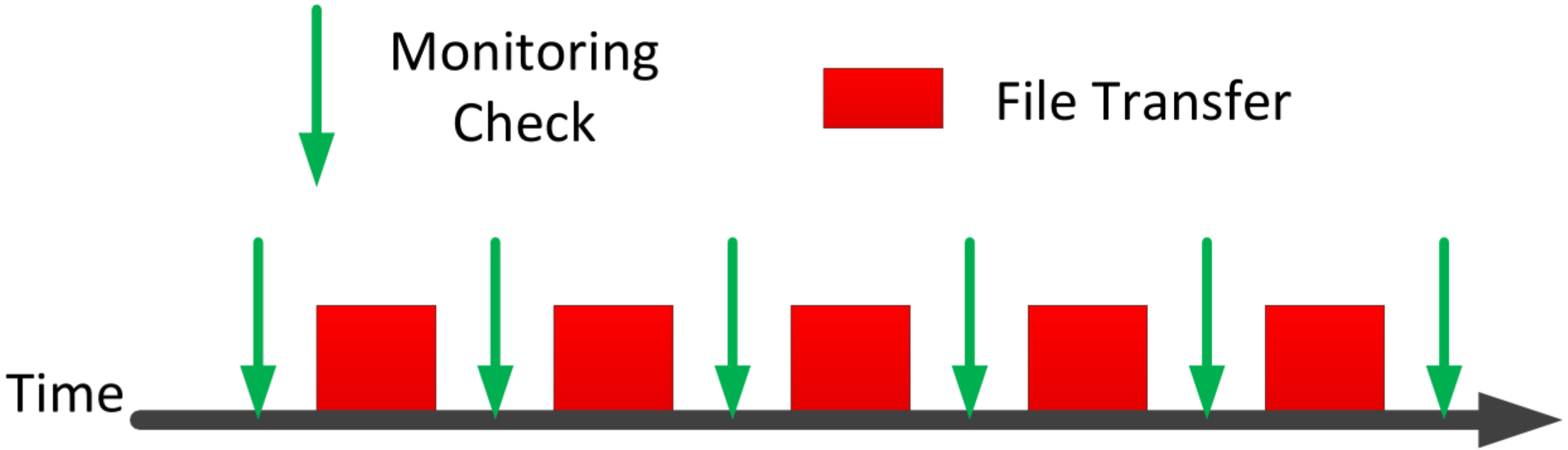
Large file transfer

- Attackers typically exfiltrate data out of the network
- Large files being exfiltrated will be detected by VMI
- Evade VMI by splitting the file up into smaller chunks

File transfer without HI



File transfer with HI



Large file transfer

- Python script transfers a chunk of the file each time it's called
- Script called from kernel using `call_usermodehelper`
- Use HI to time calls to occur right after VMI check

Polling rate effects

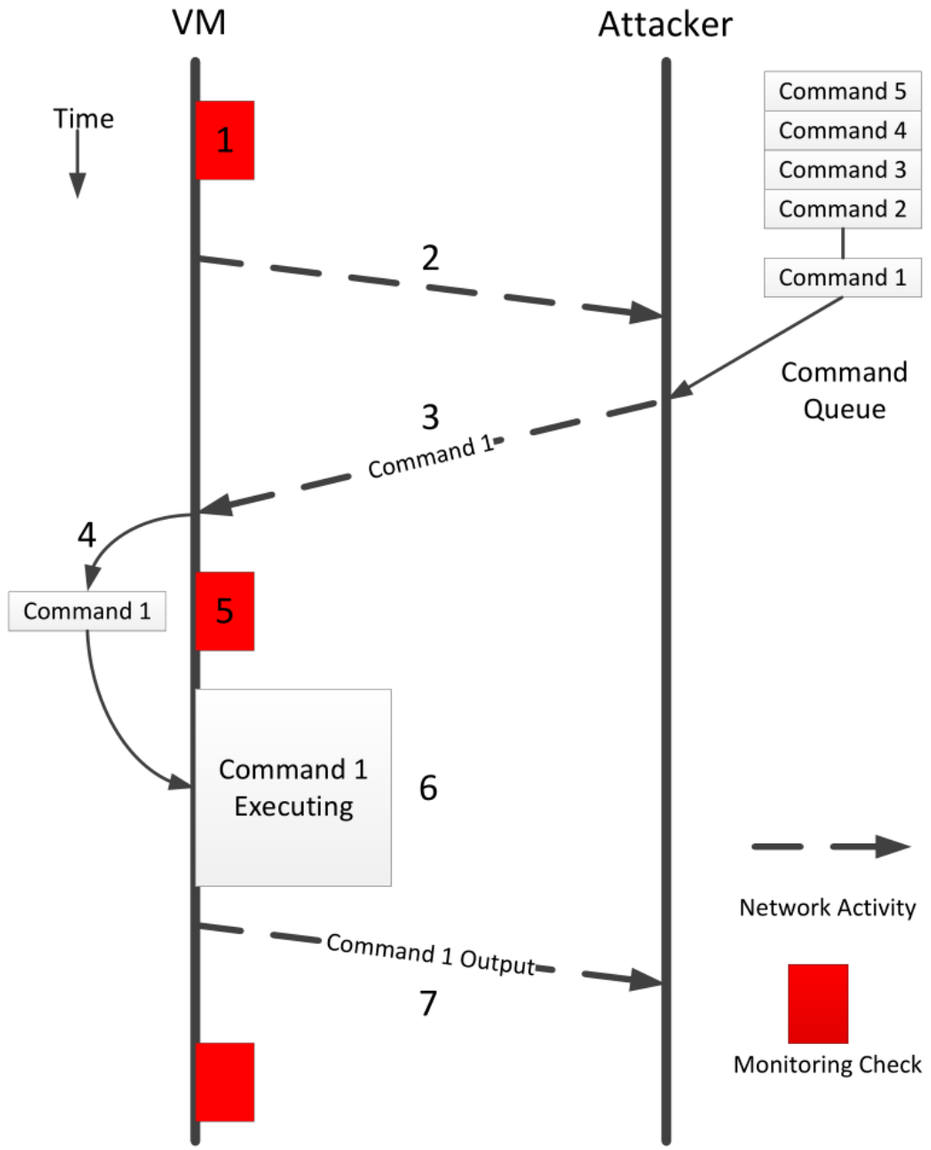
- File transfer time depends on polling rate
- One chunk is transferred after each monitoring check
  - But can change chunk size
- Chunk size/transfer time versus detection tradeoff

Backdoor shell

- Maintain access to system with backdoor
- Naïve backdoor that listens for connections is detected by VMI
- Evade VMI by having VM connect back to attacker between monitoring checks

Command cycle

1. First monitoring check
2. Backdoor client connects to attacker
3. Retrieve command to be run
4. Save command and terminate client
5. Next monitoring check
6. Runs command
7. Send output of command back to attacker



Command output latency

- Depends on polling frequency in similar manner to file transfer attack
- Command cycle steps occur between monitoring checks, so fewer monitoring checks lead to increased latency
- Attacker may batch commands to mitigate this somewhat

Possible defenses against HI

- Add noise to VM clocks
  - Previous work has used virtual clock to hide activity
  - Real time requirements may not allow this
- Randomized monitoring interval
  - Pretty good at detecting file transfer (70% success)
  - Does not catch backdoor shell

Possible defenses against HI

- Force processes to run for certain amount of time before scheduling
  - Can observe other OS operations (network activity or memory allocation)
- Non-blocking monitoring check
  - VMI may not always provide correct information

Proposed defenses against HI

- Virtual clock that only skews during VMI check
  - Hide the VM suspend with skew/catchup
  - Maintain real time requirement
- Use active VMI!
  - Hard for HI to predict when checks occur, and monitoring is more targeted

Conclusions

- Virtualization cannot perfectly isolate everything
- Passive VMI has some inherent weaknesses that can be avoided with active monitoring

Acknowledgments

- Advisors:
  - Professor Zbigniew Kalbarczyk and Professor Ravi Iyer
- Colleagues:
  - Zak Estrada and Cuong Pham
- Funding:
  - Illinois Cyber Security Scholars Program
  - Air Force Research Laboratory
  - MIT Lincoln Laboratory

Questions?