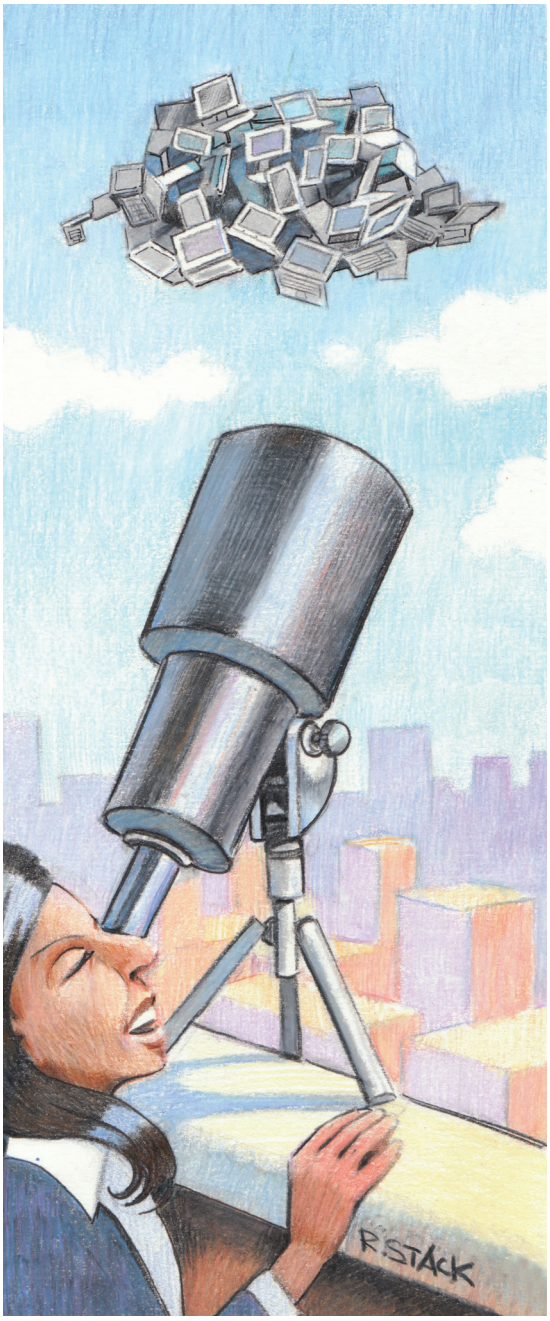
Editors: Mohamed Kaâniche, mohamed.kaaniche@laas.fr | Aad van Moorsel, aad.vanmoorsel@ncl.ac.uk

Building Reliable and Secure Virtual Machines Using Architectural Invariants

Cuong Pham, Zachary J. Estrada, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar K. Iyer |
University of Illinois at Urbana-Champaign



Reliability and security tend to be treated separately because they appear orthogonal: reliability focuses on accidental failures, security on intentional attacks. Because of the apparent dissimilarity between the two, tools to detect and recover from different classes of failures and attacks are usually designed and implemented differently. So, integrating support for reliability and security in a single framework is a significant challenge.

Here, we discuss how to address this challenge in the context of cloud computing, for which reliability and security are growing concerns. Because cloud deployments usually consist of commodity hardware and software, efficient monitoring is key to achieving resiliency. Although reliability and security monitoring might use different types of analytics, the same sensing infrastructure can provide inputs to monitoring modules.

We split monitoring into two phases: logging and auditing. Logging captures data or events; it constitutes the framework's core and is common to all monitors. Auditing analyzes data or events; it's implemented and operated independently by each monitor. To support a range of auditing policies, logging must capture a complete view, including both actions and states of target systems. It must also provide useful, trustworthy information regarding the captured view.

We applied these principles when designing HyperTap, a hypervisor-level monitoring framework for virtual machines (VMs). Unlike most VM-monitoring techniques, HyperTap employs *hardware architectural invariants* (hardware invariants, for short) to establish the root of trust for logging. Hardware invariants are properties defined and enforced by a hardware platform (for example, the x86 instruction set architecture). Additionally, HyperTap supports continuous, event-driven VM monitoring, which enables both capturing the system state and responding rapidly to actions of interest.

Continuous Monitoring

Traditional VM monitoring executes in a polling manner (periodically scanning the system) and captures only the target systems' state.[1,2] Furthermore, polling monitoring is vulnerable to transient attacks and intermittent failures that affect target systems between checks invoked by the monitor.

In contrast, HyperTap's continuous monitoring captures a complete view of relevant dynamic activity and the target system's state. It exploits the "trap-and-emulate" mechanism in *hardware-assisted virtualization* (HAV). Intel VT-x is an HAV extension to the x86 architecture that supports running an unmodified operating system in a VM. It defines guest mode and host mode execution. In guest mode, a

 Copublished by the IEEE Computer and Reliability Societies

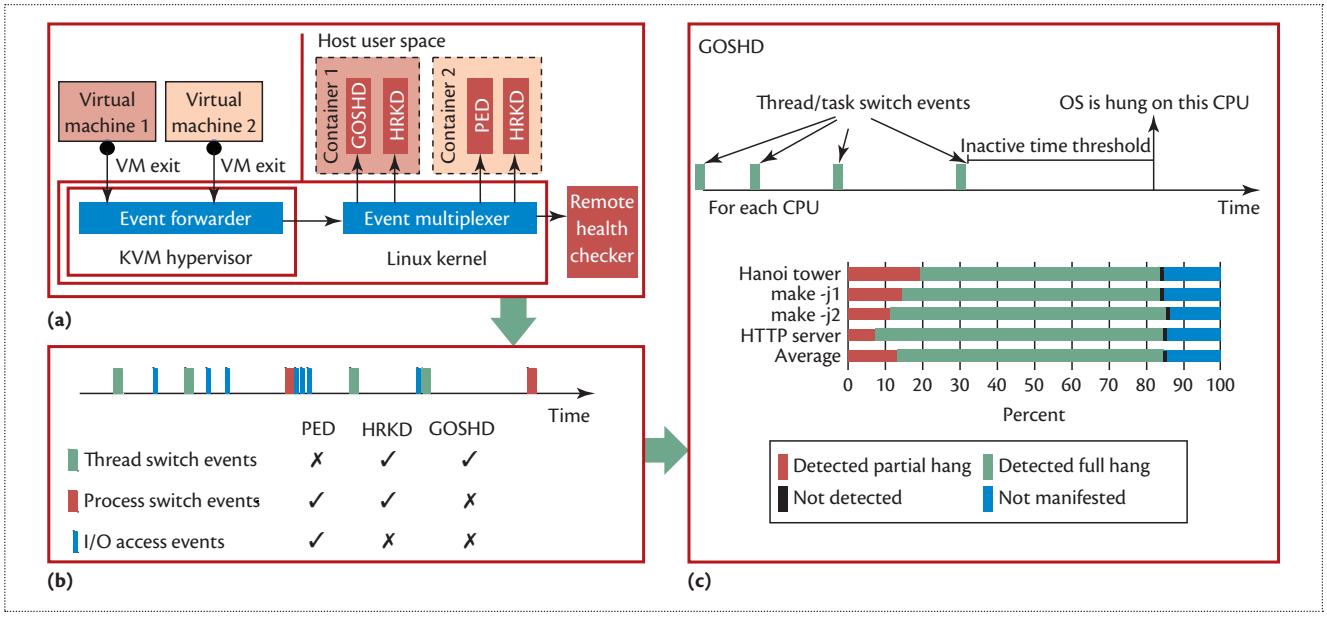


Figure 1. Implementing HyperTap. (a) A HyperTap prototype with the KVM (kernel-based virtual machine) hypervisor on a Linux platform. (b) Event types used by three auditors: guest operating system hang detection (GOSHD), hidden-rootkit detection (HRKD), and privilege escalation detection (PED). (c) An example of GOSHD. In Figure 1c, the timeline shows the principle of operation, and the graph shows the coverage results from fault injection experiments.

processor traps certain privileged operations (for example, access to processor control registers or I/O instructions). It then fires VM exit events to notify the hypervisor to emulate those operations. HyperTap intercepts VM exit events, records the related VM state, and passes that information to the auditor for detecting potential errors or malicious tampering.

Hardware Invariants

Hardware invariants must hold so that the entire software stack—for example, the hypervisor, OS, and user applications—can operate correctly.

We find that hardware invariants, particularly the ones defined by HAV, provide features that are desirable for VM monitoring. The behaviors enforced by HAV involve primitive building blocks of essential OS operations, such as process and application context switches, system calls, I/O accesses, and memory accesses. Also, you can use hardware invariants to derive OS-specific information—for example, user information and firewall rules. Details of how to intercept these events using HAV appear elsewhere.[3] Furthermore, strong isolation between VMs and the physical hardware ensures hardware invariants' integrity against failures and attacks originating in VMs.

Implementation

Figure 1a shows a HyperTap prototype coupled with the KVM (kernel-based virtual machine) hypervisor. (The same design principles are applicable to other HAV-based hypervisors.) Figure 1b lists events used to trigger auditors implemented as part of the HyperTap prototype. In this design, each VM can have multiple auditors running simultaneously. Each type of auditor can have multiple instances attached to different VMs. The core HyperTap components, including the event forwarder and event multiplexer, deliver VM exit events to the correct auditors. This design enables flexible deployment of auditors to meet target VMs' different demands.

Auditors are user processes in auditing containers (we use Linux containers; https://linuxcontainers.org) running on the host OS. Compared to the dedicated auditing VMs in previous research, this approach offers three main benefits. First, it provides lightweight attack and failure isolation among different VMs' auditors and between auditors and the host OS. Second, it simplifies implementation and reduces the performance overhead of event delivery from the event multiplexer. Finally, it allows integration of auditors into existing systems because the containers are robust and compatible with most Linux distributions.

Auditor Examples

We deployed and evaluated three auditors as parts of HyperTap:

- *guest operating system hang detection* (GOSHD),
- *hidden-rootkit detection* (HRKD), and
- *privilege escalation detection* (PED).

In our experiments, the auditors effectively detected the related attacks and failures, while causing less than 5 and 2 percent performance overhead for disk I/O and CPU-intensive workloads, respectively.

Guest OS Hang Detection

An OS is in a hang state if it ceases to schedule tasks. In multiprocessor systems, a partial hang occurs when the OS experiences a hang on a proper subset of the available CPUs. In a full hang state, the OS is hung on all CPUs. Distinguishing between partial and full OS hangs is important because typical OS hang detection approaches, such as heartbeats (in which a dedicated process or thread periodically sends an "I am alive" message to indicate the OS's liveness), are effective only against full hangs.

Detection. GOSHD tracks thread dispatches to monitor the VM's OS scheduler. If a VM's CPU (a virtual CPU or vCPU) doesn't generate thread switch events for a predefined time threshold, GOSHD declares the guest OS as hung on that vCPU. Because GOSHD monitors vCPUs independently of each other, it detects both partial and full hangs. The timeline in Figure 1c depicts the detection mechanism.

Results. To evaluate GOSHD, we injected errors in the locking mechanisms that Linux uses to synchronize access to shared data.[4] The graph in Figure 1c summarizes the results. Of approximately 18,000 injections, approximately 82 percent manifested as hangs, of which GOSHD detected 99.8 percent. What's more interesting, partial hangs were relatively common: 18 percent and 26 percent of the hangs were partial hangs on preemptible and nonpreemptible OSs, respectively. This result emphasizes the importance of partial-hang detection.

Hidden-Rootkit Detection

Rootkits are malicious computer programs that hide other programs from system administrators and security-monitoring tools. Rootkits can bypass autonomic security-scanning tools simply because their inspection lists don't contain the hidden programs.

A combination of continuous monitoring and HAV can provide a foundation for design and implementation of mechanisms for large virtualized computing systems.

Detection. HRKD monitors context switches to inspect every process and thread that uses CPUs, regardless of how kernel objects are manipulated. Each time a process or thread is scheduled to use a CPU, HRKD intercepts it for further inspection. This defeats hidden malware by putting malicious programs back on the inspection list.

Results. We tested HRKD with nine real-world rootkits in both Linux and Windows environments. HRKD always discovered the hidden applications, regardless of their hiding technique.

Privilege Escalation Detection

In a privilege escalation attack, a process gains higher privileges than originally assigned in order to obtain unauthorized access to system resources. Privilege escalation is essential to many real-world attacks.

Detection. Ninja is a real-world PED system that uses passive monitoring.[5] It's included in the mainline repository for major Linux distributions. It periodically scans the process list to determine whether a privileged (root-owned) process has a parent process that's not from an authorized user. If that process does, Ninja flags it as privilege-escalated.

We implemented two new versions of Ninja that operate at the hypervisor level. H-Ninja polls and decodes VM guest memory; HT-Ninja uses HyperTap.

To port the passive monitoring of the original Ninja (O-Ninja) to HyperTap's event-driven monitoring, we defined the events at which a process is checked:

- the first context switch of each process, and
- every I/O-related system call (for example, open, read, write, and lseek).

This ensures that checking occurs before any unauthorized action (for example, accessing a file or network).

Results. To compare the three implementations, we crafted transient attacks, which took a very small amount of time to avoid detection. We then improved those attacks by combining them with three other attacks:

- *Side-channel attacks* determined the exact monitoring interval so that we could strategically time transient attacks.
- *Spamming attacks* increased the monitor's workload to enlarge the window of vulnerability in which transient attacks could execute.
- *Attacks combining a privilege escalation attack with a rootkit* made transient attacks persistent by hiding them from the monitor.

Both O-Ninja and H-Ninja were highly vulnerable to transient attacks. For example, our side-channel attacks precisely predicted

O-Ninja's monitoring interval. Using the predicted values, we could launch transient attacks with an extremely low chance of detection. When an attack needed more time to execute, it could employ a spamming attack. For example, when we introduced 200 dummy processes, O-Ninja's detection coverage decreased to less than 2 percent. On the other hand, HT-Ninja wasn't vulnerable to any of those attacks because it used event-driven monitoring.

The HyperTap prototype shows that a smart combination of continuous monitoring and HAV can provide a foundation for design and implementation of low-overhead, highly efficient resiliency mechanisms for large virtualized computing systems, including the cloud. HyperTap's logging capabilities can be used to implement other reliability and security auditors. Examples include

- security tools that depend on system call interception[6–8] and
- failure detection mechanisms based on machine learning[9] in which the logged events and states provide inputs to anomaly detection algorithms.

This research has exemplified how to achieve reliability and security in the context of virtualized environments. It also presents an interesting research space to continue identifying similarities of these two areas in a broader context. This could lead to a common framework that facilitates solutions for problems coming from both sides. ■

Acknowledgments

This research was supported partly by the US National Science Foundation under grant CNS 10-18503 CISE, the US Army Research Office under award W911NF-13-1-0086, the US National Security Agency under award H98230-14-C-0141, the US Air Force Research Laboratory and Air Force Office of Scientific Research under agreement FA8750-11-2-0084, an IBM faculty award, and Infosys Corporation. Any opinions, findings, and conclusions or recommendations expressed in this article are the authors' and don't necessarily reflect the views of the National Science Foundation or other organizations.

References

1. B.D. Payne, M. de Carbone, and W. Lee, "Secure and Flexible Monitoring of Virtual Machines," *Proc. 23rd Ann. Computer Security Applications Conf.* (ACSAC 07), IEEE, 2007, pp. 385–397.
2. T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," *Proc. Network and Distributed Systems Security Symp.*, 2003, pp. 191–206.
3. C. Pham et al., "Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants," *Proc. 44th IEEE/IFIP Int'l Conf. Dependable Systems and Networks*, 2014, pp. 13–24.
4. D. Cotroneo, R. Natella, and S. Russo, "Assessment and Improvement of Hang Detection in the Linux Operating System," *Proc. 28th IEEE Int'l Symp. Reliable Distributed Systems* (SRDS 09), 2009, pp. 288–294.
5. T.R. Flo, "Ninja: Privilege Escalation Detection System for GNU/Linux," *Ubuntu Manuals*, 2005; http://manpages.ubuntu.com/manpages/lucid/man8/ninja.8.html.
6. N. Provos, "Improving Host Security with System Call Policies," *Proc. 12th Usenix Security Symp.*, 2003, p. 10.
7. A.P. Kosoresow and S. Hofmeyer, "Intrusion Detection via System Call Traces," *IEEE Software*, vol. 14, no. 5, 1997, pp. 35–42.
8. T. Garfinkel, "Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools," *Proc. Network and Distributed Systems Security Symp.*, 2003, pp. 163–176.
9. D. Pelleg et al., "Vigilant: Out-of-Band Detection of Failures in Virtual Machines," *Operating Systems Rev.*, vol. 42, no. 1, 2008, p. 26.

Cuong Pham is a graduate research assistant at the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. Contact him at pham9@illinois.edu.

Zachary J. Estrada is a graduate research assistant at the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. Contact him at zestrad2@illinois.edu.

Phuong Cao is a graduate research assistant at the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. Contact him at pcao3@illinois.edu.

Zbigniew Kalbarczyk is a research professor at the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. Contact him at kalbarcz@illinois.edu.

Ravishankar K. Iyer is a George and Ann Fisher Distinguished Professor of Engineering at the University of Illinois at Urbana-Champaign. Contact him at rkiyer@illinois.edu.