KungFuVisor: Enabling Hypervisor Self-Defense

Aurélien Wailly Marc Lacoste
Orange Labs
firstname.lastname@orange.com

Hervé Debar
Télécom SudParis
herve.debar@telecom-sudparis.eu

ABSTRACT

Recently, some of the most potent attacks against cloud computing infrastructures target their very foundation: the hypervisor or Virtual Machine Monitor (VMM). In each case, the main attack vector is a poorly confined device driver in the virtualization layer, enabling to bypass resource isolation and take complete infrastructure control. Current architectures offer no protection against such attacks. At best, they attempt to contain but do not eradicate the detected threat, usually with static, hard-to-manage defense strategies. This paper proposes an altogether different approach by presenting KungFuVisor, a framework to build self-defending hypervisors. The framework regulates hypervisor protection through several coordinated autonomic security loops which supervise different VMM layers through well-defined hooks. Thus, interactions between a device driver and its VMM environment may be strictly monitored and controlled automatically. The result is a very flexible self-protection architecture, enabling to enforce dynamically a rich spectrum of remediation actions over different parts of the VMM, also facilitating defense strategy administration.

1. PROTECTING THE HYPERVISOR

The Problem. Despite many expected benefits, the Achilles heel of cloud computing remains security. The virtualization layer, foundation of a cloud infrastructure, is particularly vulnerable to potent attacks based on shared resources. Subverting a hosted virtual machine (VM) or the hypervisor may lead to breaking VM isolation, giving the attacker complete system control. So far, most of the attention has focused on protecting VMs. Unfortunately, the corresponding solutions become ineffective in case of hypervisor compromise, as they assume a trusted VMM. The true challenge lies therefore in protecting the hypervisor layer.
Several recent attacks [4,6] show that the main threat to hypervisor isolation breakout comes from buggy or malicious device drivers inside the hypervisor: kernel exploitation is enabled by poor driver confinement.

Limitations of Existing Solutions. A variety of techniques were proposed to attempt to solve the problem. For instance, driver virtualization achieves strong isolation, but does not address protection of the virtualizing layer underneath [8].
Trusted computing architectures provide strong guarantees regarding hypervisor code integrity [1]. Unfortunately, they usually only detect integrity violations, and do not include remediation operations. Integrity checking is also generally static – dynamic monitoring throughout the system life-time being much harder to achieve. Driver sandboxing has also been heavily explored: a reference monitor mediates access between driver and device, kernel, or userland [5]. However, solutions remain limited to simple confinement, proposing no actions to sanitize the kernel. Security policies are also often hardcoded in the interception mechanisms themselves. Dynamic, reactive protection strategies are thus difficult to set up, as policies must be configured and updated manually.
New designs towards componentized hypervisor security architectures also aim to strengthen driver isolation, and contribute to reduce the attack surface further. However, they often require extensive code rewriting, making them hard to apply to most legacy hypervisors [3,7].
Overall, current hypervisor architectures offer no – or at best rudimentary – protection for the VMM layer. Previous attempts suffer from: (1) static, hard-to-manage security policies, not well separated from enforcement mechanisms; and (2) no remediation against threats.

2. KUNGFUVISOR OVERVIEW

To overcome the previous limitations, we introduce KungFuVisor, an autonomic security management framework for building self-defending hypervisors. This framework allows to set up several control loops to regulate hypervisor protection, with detection, decision, and reaction steps.

Threat Model. The attacker may have arbitrary control over VMs. We assume tamper-resistant hardware and related firmwares (CPU, BIOS), and boot-time hypervisor integrity. However, VMM device drivers may be flawed, and thus tampered with to exploit a VMM vulnerability. A typical bounce attack scenario sourced from a VM might be: (1) perform VM isolation breakout through a buggy VMM driver; (2) alter and subvert the driver; (3) from there, compromise other parts of the VMM or co-located VMs. Such exploitation may for instance result into rootkit injection with inter-VM traffic sniffing over the hypervisor vSwitch.

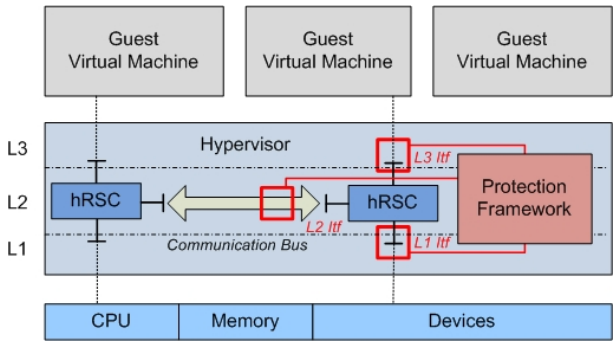

Figure 1: A 3-Layer Hypervisor Model.

Design. The framework operates through clearly-identified interception points (*hooks*) in the different hypervisor layers. KungFuVisor hooks enable to mediate interactions between device drivers, devices, VMs, and other hypervisor data structures. Thus, dynamic monitoring (*detection*) and access control enforcement (*reaction*) over communications between the driver and its environment may be achieved. It also enable easy integration into most hypervisors, provided that defined hooks are available. Note that containment is not limited to memory-based isolation (e.g., using processor-related mechanisms such as the IOMMU [7]). Enforced reaction policies may apply to other communication channels between the driver and its environment to cover a large spectrum of known exploitation techniques [2].

A security management plane provides a unified view of the *decision* logic. This plane contains orchestration facilities to realize elaborate detection and reaction patterns – both in each layer, and across layers, and between computing and networking views of VMM resources [9].
This design brings two main benefits: (1) self-managed hypervisor security automates policy administration, allowing dynamic enforcement of flexible driver isolation policies; and (2) coordination of multiple autonomic security loops enables to trigger a rich set of remediation actions over different parts of the hypervisor.

2.1 Hypervisor Model

A 3-Layered Model. We consider the generic 3-layered model shown in Figure 1 for the hypervisor architecture.
Layer 1 (L_1) contains the state of hardware computing and networking resources: CPU, physical memory, and devices (storage, network card). *Layer 2* (L_2) contains the hypervisor-level view of L_1 resources, known in KungFuVisor as *hRSCs (hypervisor ReSourCes)*: virtual CPU, host OS virtual memory, and device drivers. hRSCs are the weak point of hypervisor security, and should therefore be sandboxed and sanitized carefully. *Layer 3* (L_3) contains a number of services delivered by the hypervisor to VMs in the form of hypercalls, such as exposing or modifying the state of a given hRSC (e.g, a vNIC security configuration).

Interfaces. Each hRSC communicates with adjacent layers through 3 interfaces. The L_1 *interface* is used for instance to handle hardware interrupts. The L_2 *interface* allows the hRSC to interact with other hRSCs through an abstract *Communication Bus* capturing internal hypervisor (e.g., IPCs such as signals, shared memory, or sockets) or vSwitch-level communications. Finally, the L_3 *interface* connects the hRSC with VM resources through specific stubs in the hypervisor.

2.2 Protection Framework

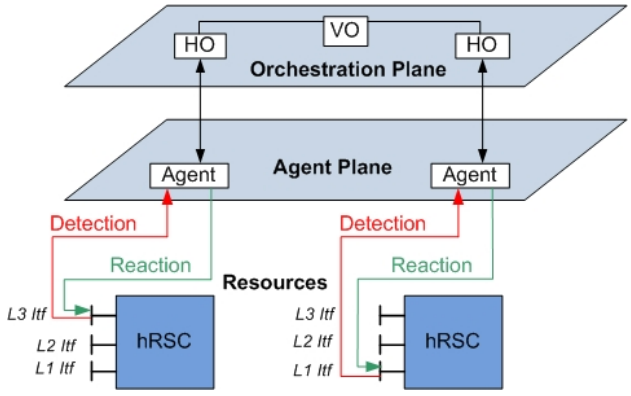

Figure 2: KungFuVisor Self-Protection Architecture.

Multiple Loops. Hypervisor self-defense is achieved through a set of autonomic loops operating over a number of components organized into 3 planes (see Figure 2). At the bottom, a *resource plane* contains the hRSCs to protect. Over it, a *management plane* containing a set of *agents* is defined for performing detection/reaction over hRSCs. At the top, an *orchestration plane* coordinates decision-making between self-protection loops.

Monitoring and Reaction. Agents are wrappers around hRSCs which mediate communications over specific hRSC interfaces via the framework hooks, to monitor activity (e.g., detect malicious invocations), or to perform reactions (e.g., forbid access to an interface). The framework is agnostic with respect to detection and reaction components: such dedicated compoents can be plugged-in to mitigate specific attacks. For instance, any type of lightweight IDS (signature-, anomaly-, or classifier-based) monitoring the L_x interface functions and parameters may be used for detection. Similarly, mechanisms for firewalling outgoing L_x calls, or cleansing the driver by internal state modification are applicable for reaction.

Decision-Making. The self-protection decision logic is split between two types of *orchestrators*. Each hypervisor layer L_x contains a *Horizontal Orchestrator (HO)* providing a layer-view of security management. The HO is a simple autonomic security manager performing a reflex, local response to threats targetted at a specific set of hRSCs incoming and propagating through the L_x layer interface. The HO supervises agents attached to the L_x interface of monitored hRSCs, aggregating collected information, and dispatching chosen reactions.
A *Vertical Orchestrator (VO)* realizes higher-level, wider spectrum security reactions. By evaluating information provided by HOs in each layer, the VO coordinates layer-level decisions in order to provide a consistent, cross-layer response to detected threats.
Orchestrator interplay results in a very flexible self-protection model allowing to enforce a rich continuum of remediation strategies, both within and across layers, and between computing and networking views of resources. For instance, a L_3-level reaction over a networking hRSC (e.g., disable a vNIC) may be triggered by L_1-level detection of anomalous behavior on a computing hRSC (e.g., physical memory tampering).

Implementation. The protection framework is easily mapped to the hypervisor model by setting all entities of management and orchestration planes directly into the hypervisor. Specific hooks then connect agents to the relevant hRSC interfaces. This design limits the attack surface, as all framework entities are in the hypervisor itself, without interfaces presented to the outside (i.e., no backdoors). Moreover, it reduces the impact on legacy hypervisor code, as agents have the same external interfaces as hRSCs. The performance overhead is also expected to be minimal, as the framework code is interfaced to hRSCs using simple function calls.

3. ONGOING WORK

A first version of the KungFuVisor protection framework has been specified and implemented to protect the KVM hypervisor. The framework-to-hypervisor mapping is currently under implementation. We now focus on adding framework components into the `qemu-kvm` part of KVM which contains the majority of device drivers.
In the future, we plan to introduce KungFuVisor components in the KVM `kvm.ko` kernel module, which also contains drivers which could lead to ring-0 exploitation. We also intend to explore how to use the hardware layer to make the framework components tamperproof. Future work includes offloading some framework components such as the orchestrators into one or more admnistrative VMs [9]. This option should improve modularity and simplify security policy administration by reducing dependence on hypervisor code. The VMM size should besides be smaller. However, it may weaken security and induce performance penalties. Finding the right balance is thus the next big challenge.

4. REFERENCES

[1] A. Azab et al. HyperSentry: Enabling Stealthy In-Context Measurement of Hypervisor Integrity. In *CCS*, 2010.
[2] T. Ball et al. Thorough Static Analysis of Device Drivers. In *EUROSYS*, 2006.
[3] P. Colp et al. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. In *SOSP*, 2011.
[4] N. Elhage. Virtunoid: Breaking out of KVM. In *DEFCON*, 2011.
[5] V. Ganapathy et al. The Design and Implementation of Microdrivers. In *ASPLOS*, 2008.
[6] K. Kortchinsky. CloudBurst: A VMware Guest to Host Escape Story. In *BLACKHAT*, 2009.
[7] U. Steinberg and B. Kauer. NOVA: A Microhypervisor-Based Secure Virtualization Architecture. In *EUROSYS*, 2010.
[8] L. Tan et al. iKernel: Isolating Buggy and Malicious Device Drivers Using Hardware Virtualization Support. In *DASC*, 2007.
[9] A. Wailly, M. Lacoste, and H. Debar. Towards Multi-Layer Autonomic Isolation of Cloud Computing and Networking Resources. In *Workshop on Crytography and Security in Clouds (CSC)*, 2011.