**Self-Stabilizing Hypervisor Architecture for Resilient Cloud**

Alexander Binun, Mark Bloch, Shlomi Dolev, Martin Ramzi Kahil, Boaz Menuhin, Reuven Yagel+

Dept. of Computer Science, Ben-Gurion University of the Negev. + and Azrieli – Jerusalem College of Engineering, Israel

Thierry Coupaye, Marc Lacoste, Aurélien Wailly, Orange Labs, France

**Motivation**. The core entity used for facilitating virtualization in cloud computing infrastructures is the *hypervisor*. Being a basic part in the virtualization infrastructure, the hypervisor is the most attractive target for attackers. The (steadily rising) complexity of hypervisors and unlimited privileges of hypervisors over virtual machines ([2]) aggravate the situation: a successful attack against the hypervisor almost certainly brings the whole system down. In the meantime, threats evolve faster than hypervisor defense mechanisms. Thus a significant part of attacks against the hypervisor succeed. This happens because proactive detection of malicious activities often requires the logic of security modules to be guided by threat-specific behavioral patterns. Detecting threats following the symptoms of (already compromised and malfunctioning) system is a more generic task as in [3].

Trying to keep the situation under control, designers augment hypervisors with additional security functions that are intended to fight new threats. This functionality can set an additional security layer threats attempt to compromise (for example, sandboxing individual drivers [4]). Another way to fight malicious activities is sanitizing the internal structures of the (possibly already compromised) hypervisor as in [2], thus mitigating the malicious actions. In any case, once a threat captures the system, the latter faces severe danger, possibly crashing.

**Novelty**. Admitting that threats evolve faster than the corresponding defense mechanisms, we shift the focus from preventing/mitigating of malicious activities to building a system that is capable of recovering gracefully after threat attacks and regaining stable behavior. We refer to recovery as the existence of (at least) the possibility for restarting from an initial (stable) state. Graceful recovery is a recovery that tries to keep the system requirements during convergence and converge fast to a stable state, possibly by rolling back (or forward) restarting the system from the stable state that is nearest in the execution history (which can be built by snapshotting and consistency checks prior to reloading). Our approach is guided by a few behavioral patterns shared by a number of threats. Exploiting the modularity of our architecture we can adapt the system to new threats.

We also leverage on the solid background in the area of self-stabilizing systems. Following [1] a system is *self-stabilizing* if every of its executions always ends up in a stable state after a finite number of steps no matter what state it is initialized in. *Stable* (also called *safe*) states are distinguished from *unstable* ones solely through the concrete application logic, namely, any system execution that starts in a stable state exhibits the desired application (also called task) behavior.

**Main ideas**. Materializing the above idea, we suggest a novel self-stabilizing hypervisor architecture. Once in a time period, a special routine, the *stabilization manager (SM)* examines the hypervisor, checking whether the latter is in a stable state. If needed, the system is set into a safe (stable) state. The corresponding enforcement actions range from simple, coarse-grained ones (like restarting suspicious VMs or even the entire hypervisor) to fine-grained ones; for example, stopping individual guest applications with suspicious behavior). Upon success, the SM notifies the *system watchdog* by sending the *AmAlive* message. Even if the SM itself is corrupted (e.g., due to a successful attack) and thus does not fulfill its duties, the watchdog ultimately reveals this (either by the absence of *AmAlive* messages or following a system integrity check) and restarts the system. The watchdog module is tamper-resilient because it is write-protected by hardware means and is triggered by a hardware non-modifiable timer. The watchdog thus forms the *trusted computing base* of our system. We provide a conceptual description of the architecture and argue that it is self-stabilizing.

As a proof of concept, we implement the SM as a separate Linux kernel module collaborating with the KVM infrastructure ([www.**linux**-**kvm**.org/](http://www.linux-kvm.org/)) and the watchdog hardware. We use KVM as a minimalistic hypervisor to illustrate the application of our concept. We chose KVM because it is widely used, compact, simple and open-sourced. Our implementation was tested with individual Denial of Service and rootkit attacks and exhibits low overhead in consistency monitoring and guarded commands execution. We plan to expand experiments using known test workloads.

##### References

1. S. Dolev. Self-Stabilization. Journal of Aerospace Computing, Information, and Communication, 1:253-255, June 2004.
2. A. Wailly, M. Lacoste, and H. Debar. VESPA: Multi-Layered Self-Protection for Cloud Resources. In Proceedings of the 9th International Conference on Autonomic Computing, ICAC 12, pages 155-160, New York, NY, USA, 2012. ACM.
3. Z. Wang, X. Jiang, W. Cui, and P. Ning. Countering Kernel Rootkits with Lightweight Hook Protection. In Proceedings of the 16th ACM Conference on Computer and Communications Security, pages 545-554, New York, NY, USA, 2009. ACM.
4. M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The TURTLES project: Design and Implementation of Nested Virtualization. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation OSDI 10, pages 1-6, Berkeley, CA, USA, 2010.