RetroVisor: Nested Virtualization for Multi IaaS VM Availability

Aurélien Wailly
Orange Labs
aurelien.wailly@orange.com

Marc Lacoste
Orange Labs
marc.lacoste@orange.com

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

Nested virtualization [1] provides an extra layer of virtualization to enhance security with fairly reasonable performance impact. User-centric vision of cloud computing gives a high-level of control on the whole infrastructure [2], such as untrusted dom0 [3, 4].

This paper introduces RetroVisor, a security architecture to seamlessly run a virtual machine (VM) on multiple hypervisors simultaneously. We argue that this approach delivers high-availability and provides strong guarantees on multi IaaS infrastructures. The user can perform detection and remediation against potential hypervisors weaknesses, unexpected behaviors and exploits.

1. ARCHITECTURE

The physical host is running a minimal hypervisor (L0) with nested virtualization enabled. L0 hypervisor controls, monitors and runs multiple guest hypervisors (L1) in isolated environments. Each L1 hypervisor deals with its own VM image and VM execution. The original VM is cloned and distributed to all L1 hypervisors at the exact same state. Further VM state evolution is filtered upstream, when the user wants to modify VM (through VNC here). It is then propagated by a dispatcher to each L1 hypervisor separately (1). The dispatcher is part of an autonomous manager. The latter detects input incoherences (2) and react appropriately using the L0 hypervisor to kill or restart L1 hypervisor and VM (3).

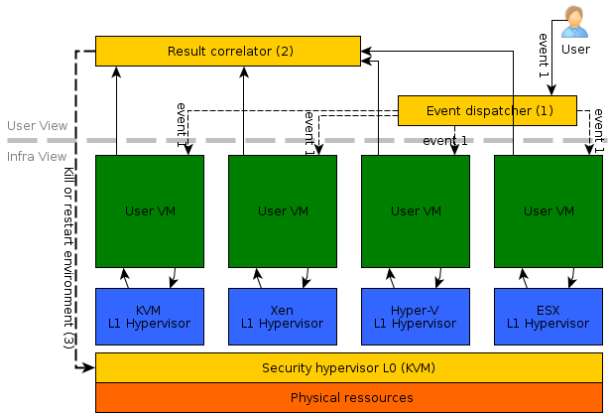

Figure 1: A Simple Use Case.

2. IMPLEMENTATION

We compare three solutions to deploy the dispatcher component: end-user client modification, network replication of packets and hypervisor modification to clone commands. We evaluate four important aspects for each of them: ease of implementation, which reflects the investment needed by programmers to integrate the solution; error tolerance of the architecture security against buggy code; genericity to estimate reusability of the current code on similar endpoint; and security of the solution in terms of the skill level required to perform an attack.

End user. The user is in charge of handling the multiple connections to hypervisor VNC interfaces. He sends mouse moves and keystrokes to each one in parallel. These concept increase the size of the client display program, according to client language. Some VNC clients are available in python, easy and fast to modify in order to match our needs. The major downside is that the end-user manages the solution entirely and thus controls global security.

Router. All packets received on the master VNC port are replicated to another port. The well-known netfilter interface provides TEE operation, which duplicates a packet and sends it to another machine. Thus we set up a VM that handles the replicated packet, modifies the destination port and forwards it back to the original machine. However it is hard to implement and poorly reusable. VNC relies on a connection-oriented protocol (TCP). First machine is connected to the client, while others drop replicated packets as there is a non-existing TCP connection. Indeed, we can deliver UDP packets but we still need to modify VNC to use this transport protocol. Either porting RFB [5] or using UDP tunnel requires modification of other components. The end-user cannot manage router, thus enhancing solution security.

Hypervisor. Only one port is listening for all VM images and can be seen as a VNC proxy, and therefore modifying the hypervisor VNC handler. This approach is error prone as each bug severely threatens the infrastructure security. Furthermore, it adds an extra layer of processing, where the hypervisor is usually under heavy load with a large number of VM. End-user is unable to use more than the usual VNC interface, increasing security.

Method	Evaluation?			
	Easiness	Fault tolerence	Genericity	Security
Client	high	high	high	low
Router	high	medium	medium	high
Hypervisor	low	low	low	high

Table 1: Solutions evaluation (higher is better)

The client-oriented solution is the less invasive and less complex of all three to set up. We modified a python VNC client for detection and reaction. Client shared screen buffer is updated by all servers, and provides a visual way to distinguish different execution of the same inputs. Client compares Incoming buffers to automatically detect display incoherences. Management is available through L0 hypervisor APIs, libvirt in our example, to kill and restart L1 hypervisors.

3. NEXT STEPS

We presented RetroVisor, an architecture design to enhance and guarantee VM execution. RetroVisor uses nested virtualization to detect hypervisor failures and recover to a safe state. We are currently supporting hypervisors appearing on Figure 1. Advanced threat detection and remediation are under development through the VESPA framework [6].

4. REFERENCES

[1] Ben-Yehuda, M. et al. The Turtles project: Design and implementation of nested virtualization. In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association. 2010, 1–6.

[2] Williams, D., Jamjoom, H., and Weatherspoon, H. The Xen-Blanket: virtualize once, run everywhere (2012).

[3] Zhang, F., Chen, J., Chen, H., and Zang, B. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM. 2011, 203–216.

[4] Butt, S., Lagar-Cavilla, H., Srivastava, A., and Ganapathy, V. Self-service Cloud Computing. In: *ACM CCS*. 2012.

[5] Richardson, T. and Levine, J. *The Remote Framebuffer Protocol*. RFC 6143. Internet Engineering Task Force, Mar. 2011.

[6] Wailly, A., Lacoste, M., and Debar, H. VESPA: multi-layered self-protection for cloud resources. In: *Proceedings of the 9th international conference on Autonomic computing*. ACM. 2012, 155–160.