Towards Multi-Layer Autonomic Isolation of Cloud Computing and Networking Resources

Aurélien Wailly, Marc Lacoste
Orange Labs, France
{aurelien.wailly, marc.lacoste}@
orange-ftgroup.com

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

Despite its many foreseen benefits, the main barrier to adoption of cloud computing remains security. Vulnerabilities introduced by virtualization of computing resources, and unclear effectiveness of traditional security architectures in fully virtualized networks raise many security challenges [5]. The most critical issue remains resource sharing in a multi-tenant environment, which creates new attack vectors. The question is thus how to guarantee strong resource isolation, both on the computing and networking side. System and network complexity make manual security maintenance impossible by human administrators. Computing and networking isolation over virtualized environments should thus be achieved and automated.

Unfortunately, current solutions fail to achieve that goal: hugely fragmented, they tackle the problem only from one side and at a given layer, thus without end-to-end guarantees. Moreover, they remain difficult to administer. A new integrated and more flexible approach is therefore needed.

This paper describes a unified autonomic management framework for IaaS resource isolation, at different layers, and from both computing and networking perspectives. A nested architecture is proposed to orchestrate multiple autonomic security loops, both over views and layers, resulting in very flexible self-managed cloud resource isolation. A first design for the corresponding framework is also specified for a simple IaaS infrastructure.

1. CLOUD RESOURCE ISOLATION

In the cloud, pooled networking and computing resources may be seen from two different but complementary views. The *networking view* abstracts the network resources, i.e., the successive protocol layers that encapsulate the data to be transmitted in the cloud. Orthogonally, the *computing view* captures the computational and storage resources of each machine at different abstraction levels (software and hardware), e.g., processor, memory, devices. Ensuring end-to-end isolation requires a fine-grained control of information manipulated in each layer crossed along the data path.

To simplify, we consider three main layers in an IaaS infrastructure: *physical*, *OS* (hypervisor), and *application and/or middleware* (VM-level), broadly corresponding to OSI network layers 1 and 2, 2 to 4, and 3 to 7.

Virtualization opens totally new attack vectors, as shown recently by many loopholes exploited in each layer. Compromising the hypervisor by a side-channel attack may leak information between VM sharing resources [10]. Layer spoofing is also possible as in the BluePill rootkit [11]. Another possibility is to fully bypass intermediate layer controls. For instance, the guest OS might directly access virtualized devices. Similarly, the applicative layer could attempt to spoof ARP requests to break physical isolation (e.g., MAC filtering). Next, we provide a brief overview of some existing isolation mechanisms in each layer.

Physical Layer. At this level, network isolation relies on dedicated network equipments like switches, routers, and firewalls. Besides evident physical separation, IEEE 802.1Q-compliant Virtual Local Area Networks (VLAN) enable to segregate virtual networks on the same physical infrastructure. It is also possible to create network overlays at link layer [13] or above [6]. Firewalls can filter desired packets in a fine-grained manner using Access Control Lists (ACLs). They can be configured from a console via a serial port, or by accessing a tiny Web server directly integrated into the equipment. The computing view is far more complex with hardware-virtualized resources such as Network Interface Controllers (VNICs) [14], Input/Output Memory Management Units (IOMMU) that map device-visible virtual addresses to physical ones, and special instruction sets to deliver virtualization in processors (Intel-VT, AMD-V).

Hypervisor Layer. One level up, the hypervisor provides control with software-virtualized interfaces and firewalls. The networking view is richer with kernel modules that extend physical switches into virtual ones. The two main competing solutions are Open vSwitch [9] and Cisco Nexus 1000v [4]. Both are very close, but Open vSwitch implements OpenFlow [7] to offer precise control of forward tables going well beyond ACL-based administration. The management of switches is simplified thanks to built-in commands accessible from the host OS. Further control on communications is possible thanks to the iptables rules which specify filtering rules for IP packets, the Linux-based bridging firewall ebtables establishing rules for the link layer, or solutions like VMware VShield Zones.

In the computing view, resource isolation can be performed through hardware-assisted instructions or emulation. Type 1 hypervisors seem most suitable for fine-grained con-

.

Layer	Networking View	Computing View
Application (VM)	Application-level firewall: WAF... Virtual firewall: VShield Zones/App... SSL/TLS VPN	Antivirus: VShield EndPoint VM introspection [3]
Hypervisor	Virtual switch [4, 9, 7] System-level firewall: iptables, ebtables L2/L3 VPN IP overlay network [6]	Security API: [8] Security modules: [1, 12]
Physical	Dedicated network equipment: firewall (VShield Edge), physical switch, router VLAN L1 VPN Link overlay network [13]	MMU/IOMMU Hardware-assisted virtualization: Intel-VT, AMD-V VNICs

Figure 1: Some Solutions for Cloud Resource Isolation.

trol over resources. Some generic hypervisors are actively maintained, with both open source and commercial solutions. VMware ESX is a mostly closed platform, control remaining limited to a fixed set of APIs, and thus difficult to extend. The Kernel-based Virtual Machine (KVM) and Xen use Intel-VT or AMD-V for instruction processing, IOMMU to separate resources, and deliver a device driver to manage virtualization hardware. KVM implements VMs as Linux processes, and therefore benefits from a wide variety of standard tools for process isolation and management. Security modules [1, 12] for isolation are available for Xen based on SELinux – thus directly supported by KVM with the advantage of built-in isolation of Linux kernel backend drivers. The libvirt library is also increasingly used to administer a wide range of hypervisors and modules. It provides a higher-level hypervisor-independent interface to control the virtualization environment. Security modules for this library are available, e.g., sVirt [8] providing MAC security schemes to isolate guest OSes and specify access permissions.

VM Layer. The VM essentially relies on the hypervisor capabilities for computing and networking isolation. However, a growing number of virtual appliances are already available to filter network data (virtual firewalls such as VShield Zone/App from VMware), to monitor the VM security status (VM introspection [3]), or to isolate a group of compromised VMs by defining a quarantine zone (antivirus suites). These solutions run into conventional user/group administration problems such permission management to determine the domains in which users, applications, and devices can be added. Some of those solutions provide little or no explicit interface to manage remotely the other VMs. Moreover, in the guest OS, security control usually remains limited to userland, while richer and stronger isolation could be possible by introducing kernel modules.

Overall, those solutions suffer from three main limitations. First, available mechanisms are highly heterogeneous, with lack of an overall architectural vision regarding their orchestration into an integrated security infrastructure. Second, the different security configurations of those mechanisms induce scalability and maintainability challenges: even if a cloud environment may be tuned to meet specific needs, nested dependencies between views and layers can become appalling and virtually impossible to solve, excluding the "by hand" approach to security management. Third, the extremely dynamic character of the cloud (e.g., with VM live migration between physical machines), and the short response times required to activate system defenses efficiently, make the problem even more complex. A flexible, dynamic, and automated security management of cloud isolation mechanisms is thus clearly lacking today.

2. APPROACH

The previous limitations may be overcome by adopting an *autonomic approach to security* of IaaS infrastructures [2], to get over fragmentation of security components providing resource isolation, automate security administration, and react rapidly to detected threats. Thus, security becomes self-managed through control loop patterns in the isolation mechanisms throughout the system. This means introducing *detection*, *decision*, and *reaction* security components that collaborate to select the adequate isolation policies matching the ambient estimated risk, and achieve an optimal level of protection of cloud resources. We propose to apply that approach for cloud resource isolation, both in terms of *views* and *layers*. To synchronize the different corresponding autonomic loops, we distinguish two types of *loop orchestration*, *horizontal* and *vertical*.

The autonomic paradigm may be applied to different *views* of cloud resource isolation, both computing and networking. Indeed, earlier research on autonomics is usually separated between *autonomic computing*, providing self-management capabilities to overcome growing system complexity, but taking distribution of resources for granted, and *autonomic networking*, applying similar principles to communication environments to master rising network management complexity. An end-to-end cloud security framework should thus be able to reconcile both views. *Horizontal orchestration patterns* are thus introduced to coordinate autonomic computing and networking loops between the two views.

Autonomic behavior can also be considered at the different IaaS *layers*. Depending on the targeted mechanism, a great number of autonomic loops may thus be introduced, which may be increased due to cross-layering issues. In order to keep the self-management model consistent, those loops have to be synchronized to maintain a stable state. This is the role of the *vertical orchestration patterns* which coordinate the loops between the different layers.

Several patterns may be used for orchestrating autonomic loops. A *centralized* model is perhaps the simplest way to organize autonomic components: a single entity dictates the global behaviour to other autonomic managers. This pattern is extensible to a *hierarchy* of autonomic managers. Other patterns such as *peer-to-peer* are also possible.

Having views interacting with layers opens a totally new range of possibilities in terms of flexibilility for cloud isolation. The whole security management model is engineered by orchestrators that obey to generic patterns, specified by the administrator. Such a design makes possible the generation of high-level strategies for cloud resource isolation, enabling easy administration and high dynamicity.

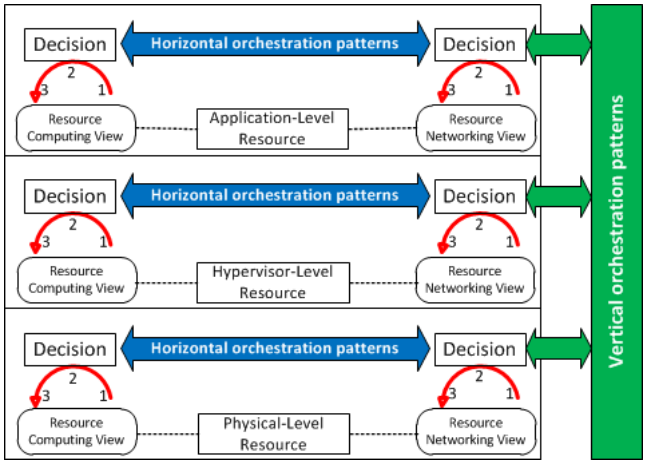

Figure 2: Isolation Framework Overview.

For instance, consider a scenario where a physical network equipment realizes isolation between applicative VM resources, also supervised by a virtual security appliance. When an attack is detected, in the physical layer, a first loop may decide to isolate a specific host in a VLAN. At the application level, the security appliance may also decide to update its database containing signatures of known viruses. However, if connection to the network is lost, the second loop will not work properly, a timeout being triggered. For the architecture to remain consistent, our approach enables to introduce rules for the physical layer loop to notify the application layer loop to stop network interactions.

We now give a brief overview of our proposition for an autonomic management framework for cloud resource isolation orchestrating feedback loops over views and layers based on the previous principles.

3. TOWARDS A FRAMEWORK

Overview. Figure 2 describes an abstract overview of our autonomic isolation framework, which makes the following assumptions on the IaaS infrastructure: (1) each cloud resource offers well-defined interfaces/hooks to capture its state (*detection phase*), and to perform actions on it (*reaction phase*), both from the system and the network perspectives; (2) in each layer, individual decision-making components (i.e., autonomic managers) provide the necessary interfaces to realize the collective layer management behavior (*horizontal orchestration*); and (3) each layer offers the needed management interfaces to perform synchronization between layers (*vertical orchestration*).

Two types of orchestrators guarantee overall consistency of self-management of cloud resource isolation:

- *Vertical Orchestrators (VO)* synchronize overall cloud resource isolation management behaviors between layers, and implements the vertical orchestration patterns. For instance, in case of inconsistency between layers, a VO may decide to enforce specific rules on well-chosen layers, according to administrator-defined policies.

- *Horizontal Orchestrators (HO)* synchronize in each layer cloud resource isolation behaviors between computing and networking views of resources. Each view sends collected informations to the HO (1). The HO summarizes its knowledge and provides it to VO (2). When a local layer isolation policy has to be modified (or new policies chosen at the overall level), views are updated by the HO through hooks previously specified (3).

Thus, orchestrators implement a hierarchical and layered self-management model, distributed over layers and views, and defining a modular system that can be easily extended to fit particular cloud network architectures.

Cloud Framework Design. Figure 3 represents a simple implementation of the isolation framework on a sample IaaS infrastructure. Dedicated network equipments provide the *physical architecture*. Network traffic is segregated by a firewall by ACL rule-matching, by a switch through VLAN tables, and by routing tables. All of these security policies are modifiable by the physical autonomic loop. For our implementation, two VLANs are plugged between the firewall and the physical machine.

The *hypervisor* (KVM) contains Linux-specific policies, such as internal routing tables, or memory associations between physical and virtual devices. In the figure, `peth0` represents the physical Ethernet interface, `eth0` and `eth1` are physically virtualized interfaces, while `br0` and `br1` are the bridge abstractions needed by the hypervisor to switch on/off an interface. Bridge `br0` memory is simply associated with `eth0`, as `br1` with `eth1`. Each bridge will be the endpoint of an emulated interface for VMs. The libvirt API handles the remote access to establish the hypervisor- layer autonomic loop.

At the *VM layer*, we consider two types of VM: the *administrative VM (AVM)* and the *user VM (UVM)*. A UVM contains at least two components: a firewall, to isolate network flows, and an antivirus that take cares of data execution prevention, program isolation and kernel signals. In the antivirus kernel component, probes monitor the loaded images in the guest OS. UVMs communicate with the hypervisor through the `conn1` connection endpoint, connected to `br0` – a second VM would be connected through `br1` and so on. The UVM virtual memory is mapped to the physical machine memory. They run on the virtual CPU (vCPU) abstraction provided by the hypervisor.

The AVM collects probes from every layer and organizes framework decisions with orchestrators (vertical and horizontal). The AVM behaves as a security management interface, collecting threat information, and deploying countermeasures. In each layer, the autonomic managers (HO_X) negotiate with both a centralized VO, and with the layer management APIs.

4. NEXT STEPS

This paper described a flexible approach to manage autonomically cloud resource isolation between different layers, reconciling computing and network views. The corresponding framework overcomes fragmentation of security components and automates their administration by orchestrating different autonomic loops, vertically and horizontally.

Going beyond the presented architecture requires several modules in different views, currently under implementation. HTTP Web servers included into physical equipments need specific APIs and a wrapper in the AVM. At the hypervisor layer, the libvirt API uses a library named netcf to enforce new network rules via XML. Although the frontend is clearly

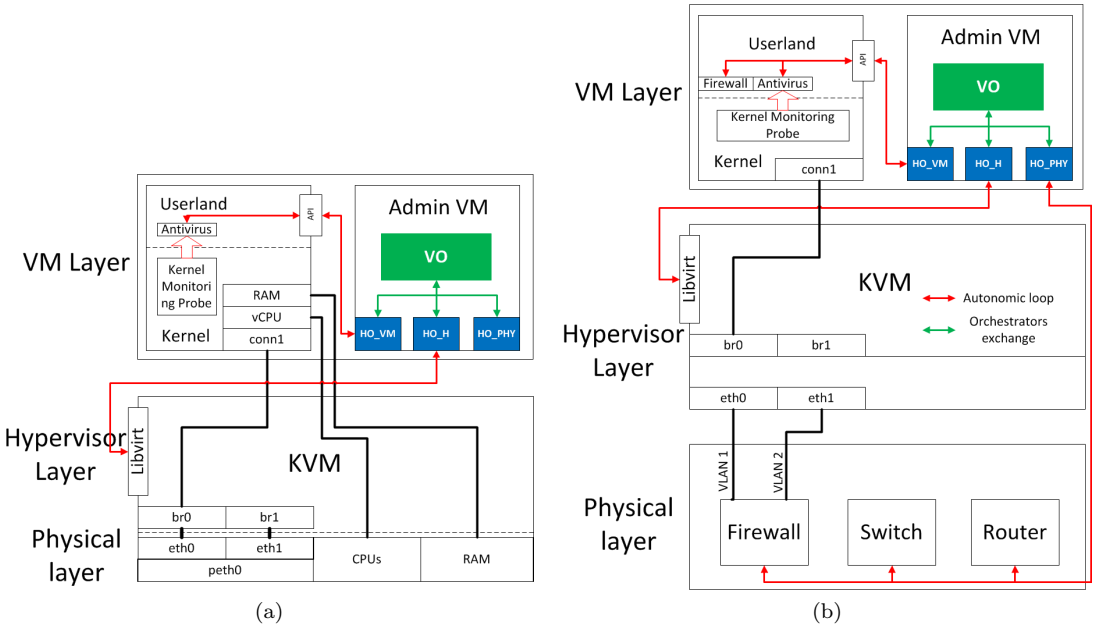

Figure 3: Framework Design: (a) Computing View; (b) Networking View.

defined, the backend is OS-dependent: we thus have to fully translate netcf's XML configuration files and to implement commands for interface creation, modification and deletion. In the VM layer, we have chosen to use ClamAV as a flexible antivirus with Python support for remote control as source code is available. Real-time protection is missing but we are implementing a kernel module to scan files when they are loaded in memory and control their execution. This example underlines what can be acheived in the VM layer: specific functions such as filtering socket creation to ban a range of compromised VMs can also be hooked.

This particular implementation is actually under deployment as security infrastructure for the French government-funded SelfXL project, aiming at self-management of large scale systems such as cloud computing infrastructures. It allows the realization of dynamic quarantine zones to isolate and clean potentially compromised VMs.

A preliminary use case of this infrastructure is the following: a virus is launched on a VM and catched by the detection engine. The VO then sends a set of orders: disable bridge connection for this VM at the hypervisor level, migrate the VM to a special physical machine, clean the VM, and migrate it back to its original location. More complex use cases will be investigated in the coming months.

Acknowledgments

This work has been funded by the ANR SelfXL project.

5. REFERENCES

[1] S. Berger, R. Careces, D. Pendarakis, R. Sailer, and E. Valdez. TVDc: Managing Security in the Trusted Virtual Datacenter. *ACM SIGOPS Operating Systems Review*, 42(1), 2008.

[2] D. Chess, C. Palmer, and S. White. Security in an Autonomic Computing Environment. *IBM Systems Journal*, 42(1):107–118, 2003.

[3] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, and D. Zamboni. Cloud Security is not (just) Virtualization Security. In *ACM Workshop on Cloud Computing Security (CCSW)*, 2009.

[4] Cisco. Nexus 1000v. `www.cisco.com/web/go/nexus1000v`.

[5] Cloud Security Alliance. Top Threats To Cloud Computing. `http://www.cloudsecurityalliance.org/topthreats.html`.

[6] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on OverLay INfrastructure. In *International Symposium on Parallel and Distributed Processing and Applications*, 2004.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communication Review*, 38:69–74, March 2008.

[8] J. Morris. sVirt: Hardening Linux Virtualization with Mandatory Access Control. In *Linux.conf.au Conference*, 2009.

[9] Open vSwitch. `openvswitch.org`.

[10] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud! Exploring Information Leakage in Third-Party Compute Clouds. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[11] J. Rutkowska and A. Tereshkin. Bluepilling the Xen Hypervisor. In *BlackHat Technical Security Conference (BLACKHAT)*, 2008.

[12] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. van Doorn. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *Annual Computer Security Applications Conference (ACSAC)*, 2005.

[13] A. Sundararaj and P. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *USENIX Virtual Machine Research and Technology Symposium (VM)*, 2004.

[14] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaied. Crossbow: From Hardware Virtualized NICs to Virtualized Networks. In *ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009.