From Nested Virtualization Architecture To 2D Security Management for Distributed Clouds

Alex Palesandro
Orange Labs, France
alex.palesandro@gmail.com

Aurélien Wailly
Orange Labs, France
aurelien.wailly@orange.com

Marc Lacoste
Orange Labs, France
marc.lacoste@orange.com

ABSTRACT

Distributed cloud computing is now moving user-centric. The promise of this evolution is: (1) unified resource management across IaaS providers, e.g., live migration independent from hypervisor technology; (2) increased customizability to choose the hypervisor services needed to realize fully à la carte, self-service clouds. However, a reality check shows that major interoperability and security barriers must be lifted, and complex trade-offs be found between functionalities provided and security assurance. Nested virtualization (NV) brings both a homogeneous interface for distributed hypervisor features and enhanced IaaS security, protecting VMs even in case of hypervisor compromise. However, practical NV adoption is still hindered by code additions to an already very large hypervisor TCB and by limited hardware support. New distributed IaaS architectures are therefore required.

In this paper, we argue that micro-hypervisor (MH) architectures are the key to overcome such NV limitations. From the previous challenges, we derive a set of design principles that a distributed IaaS architecture should satisfy. We compare possible architectural designs combining NV with micro-hypervisor features to find the best trade-off between user data privacy, modularity, TCB size, and compatibility with multiple platforms. We sketch how the resulting architecture may serve as a basis for 2D security management of a cloud of clouds: vertically, across layers for hardened IaaS security, and horizontally spanning cloud domains for end-to-end security.

1. INTRODUCTION

Distributed cloud computing is now becoming a reality. Modern cloud platforms are able to provide elastic use of resources, optimizing their allocation to meet user needs. However, currently deployed distributed cloud infrastructures present several limitations.

DCC does not handle heterogeneous infrastructures which secludes providers in non-evolving hardware. Also, security remains the last hurdle toward wide adoption of DCC. The user data are no more self contained but distributed among all parties without any proof of confidentiality, integrity or availability.

We classified problems related to DCC horizontally and vertically. The horizontal view gathers features related to interactions between IaaS stacks, such as live migrations. The vertical view addresses features close to the isolation of execution environment on the same physical machine. This classification provide a global coverage of problems to solve to deliver a secure, stable and robust architecture.

As a first element of answer, the nested virtualization [1] performed a breakthrough with enhanced isolation capabilities on the same physical machine. The hypervisor is now virtualized by a thin layer [21] to counter failures. Furthermore, the horizontal challenges were also included in this tiny layer to deliver an homogeneous interface for commodity hypervisors [17, 18]. However these classic approaches suffers from several shortcomings detailed in Section 2. Thus we propose a novel architectural approach to undertake remaining problems.

The previous works created both horizontal and vertical basic blocks from scratch. While being a necessary step, these approaches does not inherits the advances performed in terms of operating system security. The software architecture are mainly monolithic and yet-another-layer fends off security problems without figuring them out. Our approach learns from security issues fixed by OS architecure [] and apply these patterns to leverage nested virtualization. In this paper we show that component-based architecures are a promising design for the next generation of clouds.

This paper is organized as follows. Section 2 details the constraints of nested virtualization for the two dimension of our solution. In Section 3 we analyze the nested virtualization ecosystem and position existing solutions regarding our design principles. In Section 4, we details limitations of classical nested virtualization approach and in Section 5 we present the microvisor approach. In Section 6 we compare cloud architectures propositions, combining nested virtualization and microvisors, leveraging the design principles grid introduced before. In Section 8, we conclude the paper with future works and our vision of the SuperCloud.

2. CHALLENGES AND DESIGN PRINCIPLES

First, deploying cloud applications able to transparently leverage resources of several heterogeneous platforms is still difficult. The concept of super-cloud [17, 18] tries notably to overcome such issues. Multi-platform coordination is hard, primarily due to lock-in to different standards such as VM formats. Unfortunately, such lock-in is not only due to market strategic decisions, but also reflects heterogeneity of virtualization technologies. For example, each cloud provider may use different storage systems or different formats to store VMs images. Different hypervisors have nowadays a low degree of compatibility between them. This generally prevents leveraging some popular features like VM live migration between different platforms. Incompatibility is even greater if we

.

consider that each platform may adopt several types of virtualization techniques. This situation induces a tight dependency between the customer and the cloud provider. It may prevent guaranteeing service availability by relying on alternative providers, or to set up geographical optimizations due to cost or origin of traffic. This has resulted in some dramatic data server failures in recent years, causing prolonged outage of service for many applications. In what follows, we refer to this class of multi-platform issues as *horizontal* challenges.

Second, a number of issues have been raised about user data privacy and platform integrity. From the user perspective, users would like to preserve privacy of their data stored on the cloud platform. However, due to overprivileged hypervisors, a curious or malicious administrator could easily retrieve such data [21]. In a multi-tenant cloud platform, physical resource sharing between users may lead not only to data leakage or but also attacks against the IaaS [11, 13]. From the provider perspective, a key objective is to thus guarantee platform integrity against rogue VMs. However, commodity general-purpose hypervisors (GPHs) make it difficult to observe deeply all layers of a running IaaS platform. Indeed, running GPHs at the highest level of privilege hampers detection and reaction on the hypervisor itself in presence of faulty or malicious behavior – unless to cause interruption of service for all hosted VMs. Thus, both user-centric and provider-centric visions of the cloud have a problem with overprivileged hypervisors. A major challenge for new generations of cloud infrastructures is thus to find acceptable trade-offs between such conflicting requirements. In what follows, we refer to this class of layer-oriented issues as *vertical* challenges.

We identified five key design principles (DP) that a IaaS architecture should implement to meet the previous challenges. In what follows, those principles will serve as reading template to analyze different architectures.

DP1 User Data Privacy A curious or malicious administrator (or another platform user) should not be authorized to inspect and analyze user data and VM instances without explicit user consent relying on mutually trusted services. A trade-off should be found to allow providers to detect and react to potential threats, but also to prevent unlimited platform and user data access in such inappropriate operator behaviors. Thus, a platform administrator should not manage the infrastructure directly using the highest level of privilege. Instead, his actions on user data objects should be validated by a mutually-trusted kernel run in the most privileged mode. This kernel should also enforce memory page encryption to prevent snooping when pages are swapped out.

DP2 Fail-Safe Modular Architecture The architecture should be fail-safe: for instance, to enable automatically migration of instances or user-data in presence of a threat and to recover from software failures. To reach this goal, the amount of safety-critical components should be as low as possible to mimize the attack surface. Similarly, the overall design should be as modular as possible to enable platform reconfiguration.

DP3 Small TCB The number of potential failures and bugs in the platform is directly linked with the code size executed at the highest level of privilege. Thus, an architecture with a tiny TCB (Trusted Computing Base) will improve safety and integrity by design.

DP4 Inter-Platform Support The cloud architecture should allow supporting directly different platforms and service providers. The same user virtual instances should be allowed to be run (and potentially migrated) unmodified over different IaaS platforms.

Table 1: Design Principles for a Distributed IaaS Platform.

Id	Design Principle	Class of Requirement
DP1	User Data Privacy	Vertical
DP2	Fail-Safe Modular Architecture	Vertical
DP3	Small TCB	Vertical
DP4	Inter-Platform Support	Horizontal
DP5	Compatibility with Legacy	Horizontal
NDP1	Minimal Additional Code	–

DP5 Compatibility with Legacy Evolution in the IaaS architecture should avoiding disrupting existing third party applications. Thus, platform interfaces should be kept as close as possible to existing ones, to prevent code rewriting notably for applications.

In addition to the above functional design principles that specify which features have to be achieved by the platform architecture, another set of non-design principles (NDP) should be considered regarding effective feasibility of implementing the architecture. We just give an example in terms of platform usability:

NDP1 Minimal Additional Code A major issue could be represented by the amount of code to be added and maintained to an existing architecture to meet functional requirements. Despite not being a strictly a technical limitation of the architecture, it could become a show-stopper to its practical use.

In our analysis, we will consider not only the overall suitability of the architecture, but also the amount of effort needed to develop and deploy it. Table 1 summarizes the previous principles and which class of requirements they address.

3. RELATED WORK

A growing body of research has investigated building architectures and infrastructures integrating a subset of those design principles leveraging nested virtualization (NV) [1, 17, 21, 8, 4]. The distributed cloud feature set may be expanded using two levels of virtualization instead of a single one. This design allows to specialize hypervisor functions per layer, each layer addressing a subset of features, simplifying overall development compared to a general-purpose hypervisor. However, NV development on x86 architecture was initially hindered by severe performance issues [6]. However, several works have shown it possible to realize low-overhead NV using dedicated implementation architecture [1], or to recover performance loss exploiting multi-level consolidation [17]. NV slowdown can now be considered an acceptable price to achieve advanced features – hardware manufacturers are also developing new hardware primitives to reduce this overhead [10].

However, so far no architecture has currently addresses both vertical and horizontal set of challenges simultaneously. XenBlanket [17] does not investigate cloud architectural security concerns (DP2, DP3), limiting its focus to providing new features on a coordinated multi-provider cloud. Cloudvisor [21] proposed a NV-based security architecture, but cannot run more than one nested hypervisor at time, which is a criticial requirement for multi-provider cloud platform (DP4). In the super-cloud vision [18], each user runs its own L1 hypervisor instance to realize a blanket layer overcoming incompatibilities between different providers. However, security issues have not been addressed.

Other approaches based on a single layer of virtualization do not fulfill all principles either. Self-Service Clouds (SSC) [2] introduces a complete security architecture to overcome major security

and privacy issues of currently deployed cloud infrastructures. The aim is to reconcile user-centric and provider-centric requirements of the cloud. Despite being similar to the super-cloud approach (DP4), SSCs introduces new components that have to be directly managed by the user. This change of interface with user VMs prevents moving straightforwardly to the new platform without having to change its applications (DP5).

To reduce the TCB size, DeHype [19] presented a dissected version of KVM, moving a great part of the code to user space. DeHype satisfies all vertical design principles, re-interpreting the microkernel approach in a KVM setting. It increases a lot the robustness and security of a cloud platform leveraging KVM (DP1), but does not address horizontal challenges for a multi-provider service.

4. CURRENT NESTED VIRTUALIZATION

4.1 Principle

Nested virtualization (NV) refers to a system architecture with two layers of virtualization [1]. It consists in "virtualized" virtualization: the guest system virtualizes a nested guest. The concept may be generalized to an arbitrary number of nested guest layers, leading to recursive virtualization [6, 12]. (Full) virtualization is able to execute tasks in a deprivileged context that normally requires a privileged execution. With NV, this approach may be extended to virtualization itself: a guest hypervisor may be run with nested guest VMs without requiring any change to any of them.

Two layers should be distinguished when migrating to an NV architecture: (1) the hypervisor running above the hardware (*L0 hypervisor*); and (2) the nested hypervisor (*L1 hypervisor*). Nested guests are generally referred to as *L2 guests*. Unfortunately, virtualization comes with a price due to the extra layer of indirection introduced: the computational overhead is generally acceptable when dealing with single-layer virtualization, but grows exponentially when adding further levels [6]. This huge performance overhead prevented NV adoption during many years. However, Ben Yehuda et al. finally proposed an architecture tending towards more reasonable overheads [1].

NV feasibility opened plenty of new possibilities and avenues for research that have been largely discussed in the literature [1, 6, 17]. Such works all tend to diversify hypervisor functionalities, introducing new features but also keeping existing ones. It turns out that each layer addresses clearly different sets of issues.

L0 tackles vertical aspects of the platform. It aims to guarantee the best performance and strongest level of isolation possible. L0 represents the last line of defense of the platform and a privileged point for monitoring of its global status.

L1 is responsible for horizontal feature set. It has to provide the widest possible support for different platforms and the ability to virtualize using different techniques e.g., para-virtualization, hardware-assisted virtualization, dynamic binary translation. To address such requirements, Williams et al. [17] proposed a patched version of Xen, Xen Blanket, to realize an L1 virtualization layer designed to run over different hypervisors. This notably enables to run and to perform live migration of the same VM over different physical cloud infrastructures.

4.2 Limitations

Today, NV presents several limitations related to hardware support and software state of implementation.

First, to support NV, the hypervisor has to provide an exact copy of hardware virtualization primitives. Full virtualization with hardware assistance (HAV) is the most popular technique, but requires quite a complex dedicated support. HAV primitives assist instruction emulation, memory address translation, and management of devices. Both Xen and KVM propose nested VMX/SVN features that allow to efficiently handle nested hardware virtualization on single-layer hardware primitives. Full virtualization is currently the only virtualization technique supported for L1 virtualization due to its ability to run unmodified guest OSes – and thus hypervisors. Paravirtualization (PV) would on the contrary require an ad hoc modified hypervisor to act as L1 (thus violating the NDP1 principle). Despite still poor NV performance and not being yet stable on modern hypervisors [20], such feature is under heavy development and will hopefully be in a usable state soon.

Second, due to current monolithic GPH architectures, each feature directly introduced within the hypervisor such as security enhancements enlarges the TCB [21]. Therefore, a huge amount of code runs in highly privileged mode and must be trusted. Several works discuss and analyze the overall TCB of modern GPHs, always bigger than 100 KLoC [14, 16]. This approach directly violates the DP3 design principle. Moreover, the previous architecture does not improve overall stability of the infrastructure. The bottommost hypervisor L0 faces the same issues as single-layer GPH architectures: a large TCB implies also that a failure experienced in one of its critical components may trigger a generalized failure of the whole platform (violating the DP2 principle). Besides, in standard GPH architectures, the administrator has the entire control of the platform and could potentially snoop and inspect user VM status and data (violating the DP1 principle). Such designs notably do not take advantage of the possibility to use a platform compatibility layer, and more modular architectures with only a tiny security-critical kernel.

Overall, current NV architectures lack some of the dedicated features found in modern hypervisors and the impossibility to rely on a tiny TCB (violating principles DP3 and DP4). If the first one is currently being addressed by hypervisor developers, the second simply has not be achieved with modern hypervisors so far.

5. MICRO-HYPERVISORS

5.1 Principle

To solve the TCB size issue, the idea of *micro-hypervisor (MH)* architecture was introduced [14, 15, 16, 9]. Such hypervisor design draws inspiration from evolution in OS architecture (e.g., extensible, micro-, exo-, component-based kernels). Within the hypervisor are distinguished security-critical modules from non-sensitive code. The aim is to expel as much code as possible from the TCB make the hypervisor ultra-thin. Some hypervisor components (e.g., device drivers, VM address space management) may be virtualized outside the MH, isolated, and restarted in case of compromise [14]. The utmost MH removes the virtualization layer altogether [15].

MH architectures may be seen as convergence of traditional hypervisor and micro-kernels (MK). From the MK design principles legacy, the MH is guided by TCB minimization. Thus, it features a minimal kernel running in privileged mode, offering only a few fundamental services (e.g, scheduling, IPCs, memory management, VMX extensions) to other modules running in less privileged domains. All other components, such as device drivers, are moved out in user-space. The main benefit of such increased modularity is to easily define groups of user-space services, each group sharing a common a set of privileges. Trust relationships between groups may be arbitrarily defined, mirroring those between sets of principals or applications sharing the system. This allows MK to implement different "personalities": device drivers being run in user mode, such fine-grained control allows great flexibility to implement device sharing patterns. While the interest of realizing

such "personalities" in traditional MKs somewhat lessened as system services, drivers, and even applications had to be completely re-designed (hypervisors offering better primitives for abstraction with the notion of VM), such flexibility still remains a major benefit of coupling micro-kernel design with virtualization [5].

MHs are particularly well adapted to sastisfy vertical design principles. For instance, MH design increases reliability as services crashing in user space may be restarted without bringing down the entire system (DP2). The attack surface is also lowered by the small TCB size, the minimal kernel only containing security-critical components and forming a last line of defense (DP3). MH might also address some horizontal principles. "Personalities" are concretely a set of processes handling specific functions running in user-space. Implementing such features outside the kernel makes it easier for each user to rely on different instances of such personalities running concurrently, possibly in a distributed manner (DP4).

5.2 Limitations

The MH approach also showed several limitations. First, support for hardware is lacking, notably in terms of device drivers. Fortunately, this issue may be addressed leveraging I/O passthrough techniques, despite some resulting exclusive allocation challenges. Second, several cloud issues are simply not addressable with GPHs. However, GPHs are now widely adopted and foundation of existing cloud platforms. While a GPH exports mature APIs to upper software layers and is also tightly integrated with cloud toolkits, MHs lack such API stability and integration, requiring explicit dedicated code to be written – thus violating DP5. Third, for the sake of kernel size, MHs could lack several virtualization features commonly available on modern GPH hypervisors. For instance, XMHF does not support multiple guests. Such limitations applicable for MH architectures alone may be lifted partially or totally when applying MH design to NV architectures.

6. ARCHITECTURE COMPARISON

Adding an extra layer of virtualization enables to introduce a wide range of new features in the IaaS infrastructure, but does not reduce the TCB size. As already discussed, introducing MH design within a nested IaaS architecture could represent a new approach to improve NV benefits by overcoming such limitations. In what follows, we analyze the four possible architectures shown in Figure 1 to achieve nesting of GPH and MH, highlighting their potential, and assessing their benefits and weaknesses w.r.t. to the design principles of Section 2, as well as their cost of implementation.

6.1 GPH/GPH

The first architecture only features GPHs in both layers to serve as point of comparison. This type of architecture enables several remarkable features related to VM management flexibility such as live migration over different platforms, and transparent migration of virtualized clusters across data centers, fulfilling principle DP4. More generally, assuming complete and correct support of virtualization primitives for nested hardware in mainstream hypervisors, this architecture satisfies all horizontal design principles.

However, it fails to safisfy vertical design principles as TCB size is that of currently deployed NV architectures. The interest of other MH-based architectures will thus be to introduce isolation and privacy improvements by reducing the TCB, while still keeping horizontal features.

6.2 MH/MH

This architecture is considered only to highlight improvements brought by a MH design compared to a GPH/GPH architecture. An MH/MH implementation would meet the vertical requirements missing in the previous approach. As in single-layer MH architectures, hypervisor disaggregation and minimization overcomes issues posed by monolithic system architectures and over-privileged administrative VMs (DP1). The smaller resulting TCB also improves resilience of the platform against a large set of software failures (DP2) and reduces the surface of attack of the most privileged part of the IaaS infrastructure (DP3).

However, the interest this class of architecture remains limited as it cannot overcome any of the generally cited issues of single-layer MHs such as compatibility with legacy hypervisor architectures (DP5). While several horizontal properties are theoretically achievable, they would require a large development effort to implement the relevant features such as interoperable advanced resource management. Moreover, the lack of nested VMX support prevents testing in practice this type of architecture, as MH usually rely on VMX primitives to create nested HVM guests.

Overall, with this architecture, if vertical challenges are well addressed as in single-layer MHs, it does not provide any improvement for horizontal challenges, despite NV being used to bridge gaps between IaaS platform eco-systems.

6.3 MH/GPH

The third architecture aims to alleviate security concerns. It combines a GPH as L0 and a MH as L1. This approach is interesting as it may be easily deployed without requiring any extra code addition to the GPH. Hardware support is not a real barrier as existing GPH-based platforms could potentially export nested VMX features to guests. Even if this is not yet the case in modern cloud platforms, this lack may be considered temporary due to the experimental status of such features.

However, there is no TCB reduction as the code size running in L0 kernel space is as big as before. Besides, adopting a MH as L1 prevents using inter-platform enhanced features. As already seen for MH design, the price of having a thin hypervisor is normally paid with a lack of certain features. For example, XMHF only supports a single guest [16]. Thus, this architectural approach cannot provide any new horizontal features leveraging NV. Therefore, this approach can only provide the same features as those deployed today, with an increased but not sufficient level of isolation.

6.4 GPH/MH

This last approach achieves a smaller TCB than other architectures, due increased modularity brought by the micro-kernel-like design of the L0 layer – satisfying DP3. Leveraging a GPH like Xen or KVM for the L1 layer provides the user with an environment looking identical to those used in production today – satisfying DP5. The L1 GPH is normally integrated with modern cloud toolkits like OpenStack, and will thus support third-party applications – satisfying DP4. Therefore, this architecture achieves a stronger level of isolation due to vertical use of NV to introduce an additional hardware-protected security monitor. Thanks to the GPH, the MH layer remains transparent to the user and should not impact the functional properties of the IaaS platform.

This class of architecture may suffer from two types of limitations. First, nested VMX extensions must be exposed to the L1 hypervisor to let him "nest" another HVM guest. This requirement is implemented by GPHs like Xen and KVM, but usually not included in the key features to be provided by a MH: implementing nested VMX may be estimated to around 1KLoC, to be compared with the size of a typical MH (2-20 KLoC) [16]. More generally, the MH quest towards ever increasing minimalism could represent a serious limitation to preserving functions of existing platforms:

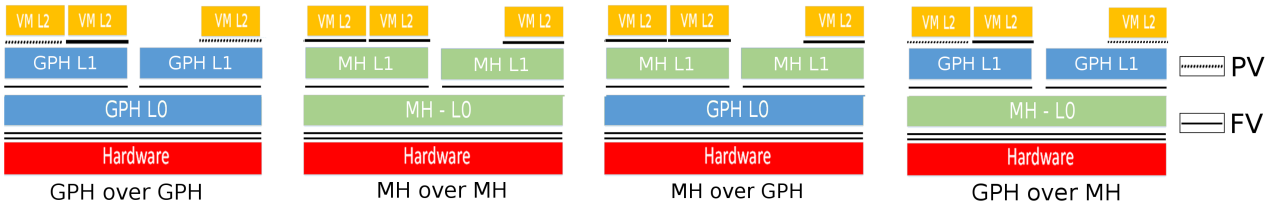

Figure 1: Different Types of Nested Architectures.

the constraint of privileged code size may force MH developpers to drop some features considered as non-essential. This may apply to NV-related primitives, but also to other types of features. For instance, XMHF may only virtualize a single L1 instance at a time. While this may be acceptable if the platform objective to enforce strict resource access control in a single-provider private cloud, it hinders having a platform model enabling the user to run its own hypervisor as L1 to set up a self-service cloud independently from the underlying provider – as in that case, multiple L1 hypervisors would need to run concurrently on the L0.

Second, device drivers should be correctly managed. Drivers are a major MH issue, due to costs of development or of adaptation to a new architecture. Such challenges may be addressed using device assignment: the L1 GPH hypervisor manage the physical device with the proper driver now running in a lower level of privilege. To be workable, this approach would require exclusive allocation to be simply implementable, e.g., using SR-IOV [3].

6.5 Discussion

Overall, how do the architectures above compare w.r.t. to the two main reported benefits of NV, i.e., vertical, through an increased level of isolation between hypervisor and customer VMs, and horizontal to easily achieve multiple provider platform interoperability at system level? Table 2 summarizes the previous analysis.

The GPH/MH architecture appears as the most interesting to expand the NV promised benefits. Through a solid and secure MH as L0, it is the only one able to reduce the platform TCB. This would bring several benefits towards a more secure infrastructure. First, a guarantee that the platform administrator could not lose control of the system thanks to an extra security layer. Second, certification perspectives: Klein et al. [7] showed that less than 10 KLoC kernels could be formally verified, under several conditions, which is far from be achievable today with standard GPH/GPH approach where the TCB is an order of magnitude bigger than MHs [14].

However, the effort to set up a GPH/MH is strongly dependent on the technique for L2 virtualization. As already seen, to support L2 guests with hardware-assisted virtualization which is becoming mainstream, the MH needs to export nVMX (nested VMX) primitives. On the other end, adopting paravirtualization or dynamic binary translation for L2 does not requires particular hardware support, and thus would not require MH patching to set up a running platform. However, PV remains limited by supporting only a subset of system features which could be an issue as a general solution.

Therefore, a MH-based architecture could increase the feature set of current NV architectures, mostly vertically. The main remaining barrier is to come up with an MH implementation supporting HAV nested guests. We are currently working on a GPH/MH implementation based on Xen over NOVA aiming to lift this limitation using PV.

7. TOWARDS 2D SECURITY MANAGEMENT

We now sketch how the previous enhanced-NV design could enable to build a security architecture for a distributed cloud infrastructure (DCI) satisfying both vertical and horizontal requirements. Such DCIs will face 2D threats, vertical spanning layers, and horizontal, spanning domains DCIs being multi-layer, multi-domain distributed cloud infrastructures. Vertically, attacks currently target several infrastructure layers (e.g., VMs, hypervisor). Unfortunately, most existing defenses are generally for single layers only. This makes it difficult to grasp the overall extent of an attack. Horizontally, threats may propagate through the network from cloud to cloud. Unfortunately, most existing defenses are for single clouds only. Lack of interoperability in security policies and mechanisms is thus a major barrier to unified hybrid cloud security. DCI security thus requires 2D protection.

We consider the security architecture shown in Figure 2.

Supercloud SUPERCLOUD Layer built SS cloud Also security management both vertically and horizontally.

OS:

Provider MH: Assumption min V, XenDisagregated... Above GPH: Achieve Vertical security Cross Layer security

GPH allows also to layer of interoperability horizontal blanket layer above the different MH to enalbe self-service security.

On the top are the VM built self-service security where security is personalized/customized. Automation. Cross-Layer Framewokr: VESPA Cross-Domain Framework: 360 Atuomation

This means: providing vertically a cross-layer integrated vision of security of the cloud and SUPERCLOUD infrastructure; defining horizontally a security abstraction layer to achieve a homogeneous level of security across clouds independently from underlying provider mechanisms; and integrating both dimensions smoothly. This approach is key to providing a unified view of security of the SUPERCLOUD. Our ambition is to define a security supervision architecture and framework for the SUPERCLOUD enabling this 360 view of security management with a high level of automation. Thus, the SUPERCLOUD will bring unified, simple management of security for cloud of clouds.

Our vision is that the security of the SUPERCLOUD should be both self-managed and 360.

The second challenge means that although many defense mechanisms may be leveraged in this 2D landscape, they remain difficult to configure and to reconcile. The current “by hand” approach becomes a nightmare for security administrators. Despite some automated solutions for managing security of single clouds, there is currently no solution truly applicable to federations of clouds in practice.

two major security challenges:

In terms of architecture: The northbound, user-centric supercloud API will implement self service security and dependability, allowing to build self-service clouds (SSC) where VM, net-

Table 2: Architecture Comparison.

Id	Functional Design Principle	Class of Requirement	GPH/GPH	MH/MH	MH/GPH	GPH/MH
DP1	User-Data Privacy	Vertical	×	✓	×	✓
DP2	Fail-Safe modular architecture	Vertical	×	✓	×	✓
DP3	Small TCB	Vertical	×	✓	×	✓
DP4	Inter-Platform support	Horizontal	✓	×	×	✓
DP5	Compatibility with Legacy	Horizontal	✓	×	×	✓
NDP1	Minimal Additional Code	–	✓	×	×	✓

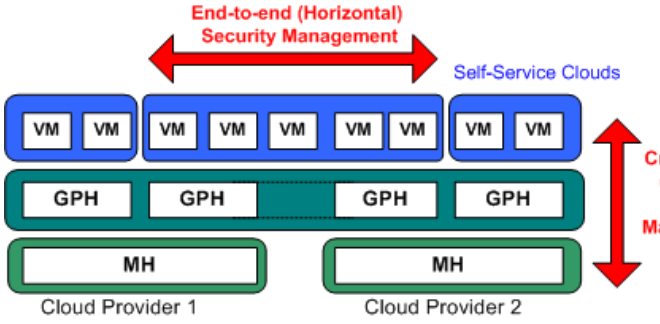

Figure 2: Security Architecture.

work, and storage security may be personalized independently from the underlying provider. Such clouds may be defined at the service level to the full IaaS, relationships between SSC being flat or nested, also with complex trust relationships between sets of SSC according to the ecosystem relationships of the super-cloud providers providing the SSC. The southbound, provider-centric super-cloud API will define a unified security and dependenability management plane of the compute, data, and network resources independently from providers. This may apply for B2B use cases to management of clouds of clouds but also to device-to-cloud scenarios if devices are considered, with possible extensions to the personal cloud for device-to-device scenarios, the SSC taking the form of personal cloud-based digital infospheres where security and dependability may be totally user-controlled. ... The SUPERCLOUD will fulfill that vision with a two-level infrastructure: • The lower part of the SUPERCLOUD is be a distributed infrastructure will realize the distribute resource abstraction layer federating distributed cloud resources and implementing the self-service security to build SSC. It will implement the northbound, user-centric super-cloud API. • Above, the upper part of the SUPERCLOUD will be a supervision infrastructure providing a set of security and resilience services to guarantee the security of the overall SUPERCLOUD architecture. It will realizing self-service security, end-to-end security, and resilience. It will implement the southbound, provider-centric super-cloud API.

and security management complexity. Th

A DC

Autonomous

Supercloud architecture

Proposition of Architecture

Base Layer

Supercloud layer

VESPA.

In the previous sections we proposed a functional analysis of cloud architecture in order to respect design principles that we identified in the introduction. In this section, we propose a GPH over MH implementation, based on .

Inter-platform support Executed as L1, Xen would virtualize its nested guest initially using PV delivering the compatibility layer.

Tiny TCB The NOVA kernel will represent the only part of the platform executed at the highest level of privilege. Even the VMM, Vancouver, will be executed in the L0 user-space. The microkernel in L0 enforces a strictly modular approach to L0. Therefore, the VMM will present distinct processes instances for each L1 guest, enforcing a context division.

3rd part compatibility From a user point of view the architecture still exports the same interfaces provided by the Xen hypervisor, allowing the user to keep its applications unchanged.

Fail-safe components architecture Moreover, the tiny core is the only safety-critical software component. The system could potentially be able to recover any crash in different user-space components.

8. CONCLUSION AND FUTURE WORK

In previous sections we provided a global analysis of modern challenges of cloud computing and how those could be globally addressed addressed leveraging nested virtualization and different hypervisor architectures.

Future works will be focused on two complementary parts. Firstly, they will concern the completion of the architecture of security in order to concretely meet horizontal design principles, identifying key components, similarly as done for the vertical properties. At the same time, a major effort will be directed on enlarging the initial implementation.

9. REFERENCES

[1] M. Ben-Yehuda, M. 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 *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[2] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-Service Cloud Computing. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[3] Y. Dong, Z. Yu, and G. Rose. SR-IOV Networking in Xen: Architecture, Design and Implementation. In *USENIX Workshop on I/O Virtualization (WIOV)*, 2008.

[4] A. Fishman, M. Rapoport, E. Budilovsky, and I. Eidus. HVX: Virtualizing the Cloud. *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2013.

[5] G. Heiser and B. Leslie. The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors. In *ACM SIGCOMM Asia-Pacific Workshop on Systems (APSys)*, 2010.

[6] B. Kauer, P. Verissimo, and A. Bessani. Recursive Virtual Machines for Advanced Security Mechanisms. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) Workshops*, 2011.

[7] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[8] C. Liu and Y. Mao. Inception: Towards a Nested Cloud Architecture. In *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2013.

[9] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. D. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Symposium on Security and Privacy*, 2010.

[10] J. Nakajima. Making Nested Virtualization Real by Using Hardware Virtualization Features, 2013.

[11] D. Perez-Botero, J. Szefer, and R. B. Lee. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *International Workshop on Security in Cloud Computing*, 2013.

[12] G. J. Popek and R. P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[13] 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.

[14] U. Steinberg and B. Kauer. NOVA: A Microhypervisor Based Secure Virtualization Architecture. In *ACM European Conference on Computer Systems (EUROSYS)*, 2010.

[15] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[16] A. Vasudevan, S. Chaki, L. Jia, J. McCune, J. Newsome, and A. Datta. Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework. *IEEE Symposium on Security and Privacy*, 2013.

[17] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize Once, Run Everywhere. In *ACM European Conference on Computer Systems (EUROSYS)*, 2012.

[18] D. Williams, H. Jamjoom, and H. Weatherspoon. Plug into the Supercloud. *IEEE Internet Computing, Special Issue on Virtualization*, 17(2), 2013.

[19] C. Wu, Z. Wang, and X. Jiang. Taming Hosted Hypervisors with (Mostly) Deprivileged Execution. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[20] R. Yongjie. Nested Virtualization Test Report for Xen 4.3-RC1.

[21] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.