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

The problem of homogeneity over multiple IaaS providers is receiving considerable attention with the evolution toward nested virtualization. These new architectures unveil user-centric opportunities such as efficient resources management and VM migration. However the inherent drawbacks hinder its global adoption with a larger TCB and the complexity to monitor another virtualization layer. Thus, we argue that the promising architectural view of micro hypervisors approach is the key to the separation of hypervisor duties and reducing the attack surface.

Furthermore, our architecture mushrooms commodity and micro hypervisors to leverage nested virtualization in terms of user-data privacy, modularity, TCB size and compatibility. These design principles are evaluated for each architecture configuration and compared to establish a reference security architecture. The resulting combination is built over common hypervisors empowered with micro hypervisors features. A reference implementation proven the architecture to be viable and will be extended to meet broader hypervisor scopes, enabling migration over heterogeneous cloud servers.

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.

First, deploying cloud applications able to transparently leverage resources of several heterogeneous platforms is still difficult. The concept of super-cloud [13, 14] 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 [17]. 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 [9, 10]. 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 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 In order to recover soft failures, the archtecure is supposed to be fail-safe, being able of automatically migrate instances or user-data in presence of a menace, but also reactivating the correct functioning in presence of a

Id	Functional design Principle	Class
DP1	User-Data privacy	Vertical
DP2	Fail-Safe modular architecture	Vertical
DP3	Small TCB	Vertical
DP4	Inter-platform support	Horizontal
DP5	Backwards 3rd part compatibility	Horizontal
NDP1	Minimize code addition	

software failure. In order to satisfy this property, the amount of safety-critical components has to be as lowest as possible and the overall design has to be kept as modular as possible.

DP3 Small TCB The amount of lines of code executed at the highest level of privilege is normally proportional to the number of potential failures and bugs in the platform. According to this, an architecture presenting a tiny TCB will improve its safety and integrity properties by design.

DP4 Inter-Platform Support The cloud architecture has to be designed in order to support directly different platforms and service providers. The same user virtual instances have to be executed (and potentially migrated) unmodified over different platforms.

DP5 Compatibility with Legacy Avoid disruption of existing 3rd part applications compatibility could represent a major principle to consider. Platform interfaces has to be kept as close as possible to the existing one, preventing application rewrite.

Moreover, beside this functional design principles specifying which features have to be achieved by the platform, another set of non-design principles could be identified. Those principles does not concern directly features or aspects of the architecture of the platform, but the effective feasibility of the architecture.

NDP1 Minimize code addition a major issue could be represented by the amount of code lines that have to be added and maintained to an existing architecture to meet the functional requirement. Even it does not represent strictly a technical limitation to a certain proposal, it could become a "show-stopper" factor for its utilization.

In our analysis, it will be kept in consideration not only the overall suitability of the architecture, but also a quantification of the effort needed to develop and deploy it. In table 1 it is possible to observe a resume of previous cited principles.

This paper is organized as following. In Section 2 we analyze the nested virtualization ecosystem and position existing solutions regarding our design principles. In Section 3, we details limitations of classical nested virtualization approach and in Section 4 we present the microvisor approach. In Section 5 we compare cloud architectures propositions, combining nested virtualization and microvisors, leveraging the design principles grid introduced before. In Section 9, we conclude the paper with future works and our vision of the SuperCloud.

2. RELATED WORK

Several works presented in literature integrates these a subset of this design principles in their architecture, leveraging nested virtualization [1, 13, 17] Cloud computing feature set could be enlarged exploiting two layers of virtualization instead of just one. The presence of two different hypervisors allows the possibility to specialize hypervisors functions. Instead of having a general purpose hypervisor we could split the hypervisor role in two different layers, interconnected with a well defined interface. Each layer will have to care about a subset of features, simplifying overall development. However, nested virtualization on x86 architecture faced impeding performance issues [4].

However, several works demonstrate that it is possible to put in place low overhead nested virtualization using special approaches [1] . Moreover, other works try to recover performance loss exploiting multi-level consolidation, introduced in nested virtualization [13]. In conclusion, nested virtualization slowdown could be considered nowadays an acceptable price to achieve advanced features and, in addition, hardware manifacturers are developing and proposing new hardware facilities to reduce this overhead [8].

To sum up, no architecture had been presented to address vertical and horizontal class of problems at the same time until now. XenBlanket does not investigate cloud architectural security concerns (DP3,DP2), limiting the focus on new potential feature of a coordinated multi-provider cloud. Cloudvisor proposed a security architecture that leverages nested virtualization without allowing the possibility of executing more than one nested hypervisor at time, that is a crucial requirements of a multi-provider cloud platform (DP4). In a SuperCloud scenario, each user has to run its own L1 hypervisor instance to enforce a compatibility layer, overcoming incompatibilities between different providers.

Other approaches leveraging only one layer of virtualization does not achieve the whole design-principle satisfaction. Self-service cloud approach (SSC) [3] design a complete security architecture to overcome major security and privacy issues of currently deployed cloud infrastructures, in order to alleviate the different needing user-centric and provider-centric vision of cloud computing presented in 1. Even if this architecture reproduces an approach similar to the SuperCloud without leveraging an extra layer of virtualization (DP4), the approach changes the behavior where the user acts introducing new components that have to be directly managed by the user. Therefore, the main drawback of this solution is to change the interface with the user, preventing the possibility to straightforwardly move to the new platform without having to change its applications (DP5).

Furthermore, DeHype [15] presented a dissected version of KVM hypervisor, trying to reduce the TCB of this popular hypervisor. The approach consisted in demoting a great part of the hypervisor codebase to a user-level execution. DeHype satisfies all vertical design principles listed before (DS1,DS2,DS3), reinterpreting the microkernel approach defining the greatest part of code to userspace. DeHype increases a lot the robustness of a cloud platform leveraging KVM, limiting for example the possibility of attack in multi-tenancy platform, but it does not address horizontal design principles about a multi-provider service.

3. CURRENT NESTED VIRTUALIZATION

Nested virtualization refers to a system architecture with two layers of virtualization. It consists in a "virtualized" virtualization, where the guest system virtualize a nested guest. The concept could be generalized to an arbitrary number of nested guest layers, introducing recursive virtualization. Virtualization is able to execute in a deprivileged context that normally requires a privileged execution. With nested virtualization we could apply this approach even to virtualization itself, executing a deprivileged hypervisor and leveraging code reutilization.

Introducing nested virtualization, we have to identify the two main actors involved in the virtualization process: bare-metal hypervisor (L0) and nested hypervisor (L1). In addition, we will refer to the nested guest as L2 guest. However, virtualization introduces a computational overhead. It is normally acceptable when we are

dealing with single layer virtualization, but it grows exponentially when we try to add further levels [4]. This huge performance overhead has prevented nested virtualization adoption during last years and had been finally addressed by Ben Yehuda et Al [1] proposing an architecture with acceptable overhead.

Nested virtualization feasibility opened a plenty of new possibilities and perspectives, that have been largely discussed in literature [4, 1, 13]. The first aspect in common among different works proposed in literature is to diversify hypervisor functions, introducing new features but keeping the existing ones. In a general analysis of existing works, it could be possible to identify the general role of each different layer. On one hand, L0 is charged of the vertical aspects of the platform. It has to achieve the best possible performances and enforcing the strongest possible level of isolation. L0 represents the last line of defense of the platform and a privileged point of global status observation.

On the other hand, L1 is responsible for horizontal feature set. It has to provide the widest possible support to different platforms and the capacity to virtualize adopting different techniques (paravirtualization, hardware-assisted, dynamic binary translation). According to these requirements, Williams et al. [13] propose a patched version of Xen, Xen Blanket, to accomplish a L1 virtualization layer, designed to run over different platforms. This concretely allows to run over different physical platform the same VM, and live-migrate it at the occurrence.

3.1 Limits

Nowadays, nested virtualization presents several limits related to the hardware support and software state of implementation. At first, to provide nested virtualization, hypervisors has to provide an exact copy of hardware virtualization facilities. The most exploited technique, full virtualization with hardware assistance (HV), relies on a quite complex dedicated support. This set of facilities assists instruction emulation, memory address translation and device management. Both Xen and KVM propose nested VMX/SVN features, that consists in the possibility to efficiently handle nested hardware virtualization, in presence of a single-layer hardware facilities. Although nested virtualization performance are still not good and the feature is not yet stable on modern hypervisor [16], the feature is under heavy development and it will be hopefully in a usable state soon.

Secondly, due to the monolithic architecture, each feature introduced in GPH enlarges the TCB. Therefore, leveraging GPH, a huge amount of code is running in the most privileged mode and has to be trust. Several works discuss and analyze the overall TCB of modern GPH [11, 12], considering GPH TCB always bigger than 100 KLoC. This approach directly collides with one design principles introduced in the introduction paragraph (DP3),

Finally, the above architecture does not improve the overall stability of the infrastructure. The underlying core, L0, has the same problem of single-layer GPH architecture: the large TCB implies also that a failure experienced in one of the critical components will simply produce a generalized failure of the whole platform, not satisfying DP2. Moreover, in GPH standard architecture, the administrator has the entire control of the platform and could potentially snoop and inspect user VM status and data.

This kind of architecture, in particular, does not leverage completely the possibility to hide behind a compatible layer, a really modular architecture with only a tiny critical core. To sum up, current nested virtualization limits are represented by a certain lack of dedicated feature in modern hypervisor and the impossibility to rely on a tiny TCB (DP3, DP4). The first one is being addressed by hypervisor developers. The second one could simply not be achieved with modern hypervisors.

4. MICRO-HYPERVISORS

Consequentially, several works focused this problem leveraging a micro-hypervisor architecture, with a traditional microkernel-like design that tries to consistently reduce the amount of code running with the highest privilege level. A micro-kernel traditionally consists in a tiny core running in Kernel Mode, offering only few fundamental services to other modules running (scheduling, IPC, memory management) in a less privileged domain. All other components, in particular device drivers, are moved out in user-space. Microkernels had been questioned because of the wasteful performance of first generations, due to IPC intrinsic cost. This argument had been completed confuted by Liedtke in 1993: [6, 7].

Microkernels have the capacity of easily redesign the hierarchy of privileges. A user application has normally to trust the tiny micro-kernel and only the set of user-space software it uses. However, this set of trusted set of drivers and tools have not to be trusted by every application executed in the system. Normally, this allows microkernels to implement different "personalities". User applications shares the system but could have to deal with completely different system services in a transparent way. However, "personalities" interest had been hugely reduced by the fact that system services, device drivers and even application had to be completely re-conceived to fit this new design pattern. Therefore, with the introduction of VMs, microkernels lost the battle in favor of hypervisors, in terms of popularity. However, the great advantage of tiny TCB, provided by microkernel, was not provided also by hypervisors. As we analyzed before, they present normally a monolithic architecture, like the great part of modern OS. This feature kept alive the interest in microkernels, in particular trying to couple microkernel benefits with virtualization.

Therefore, leveraging this concept of microkernels, a micro hypervisor (MH) is composed from a small kernel providing only essential functions, like scheduling, IPC, memory management and VMX extension. Several micro-hypervisor have been proposed in literature [12, 11]. They rely on the micro-kernel paradigm, extending it to virtualization. MH naturally satisfy vertical design principles (DP1,DP3, DP4). The tiny core could be considered the only system-critical componente of the architecture and, due to its relative small dimension and the consequent minor attitude to vulnerabilities,it could represent the "last line of defense".

In addition, MH inherits the concept of micro-kernel to implement different "personalities". Those "personalities" are concretely a set of "processes" handling specific functions running in user-space. As those functions are implemented outside the kernel, each user could rely on a different instances of the hypervisor user-space process, that could coexist at the same time. This process separation between user could enforce a straightforward mechanism to neutralize VM-to-VM attack from a malicious user. Moreover, user-space processes are easier to debug with commodity tools (GDB, valgrind). Finally, this extraction process drives to a more solid platform, considering that user-space crashes are generally not fatal for the entire system integrity. A possible example, that will be analyzed in next section, is the user-space VMM component of Nova, Vancouver, that resides in user-space [11]. In addition, this approach is not far from the one proposed in DeHype [15], where a consistent part of the KVM hypervisor has been demoted in user-space.

4.1 MH General Issues

Despite the positives, MH approaches had demonstrated some limitations. Firstly, the straightforward one is the initial lack of

hardware support, lack of device drivers are still a relevant Achilles' heel for this solutions, inherited from its ancestors microkernels. Fortunately, this kind of issue could be addressed leveraging I/O passthrough techniques, although it will cause exclusive allocation issues.

Moreover, as reported in previous sections, several cloud computing issues are simply not addressable with general purpose hypervisor (GPH). However, they are commonly widely adopted and represent the bedrock of existing platform. Therefore, GPH exports wide and solid interfaces leveraged by a plenty of different software that is able to properly communicate and act through them on the virtualization layer. On the contrary, MH lack of cloud toolkit integration, requiring an explicit support to be written, not satysfing 3rd part compatibility requirements (DP5). GPH are tightly connected with cloud toolkit and could present solid interfaces to communicate with different tools.

Finally, microkernel modularization introduce a major challenge in architecture design. Due the increased complexity, for sake of kernel size, MH could generally lack of several virtualization features, commonly available on modern GPH hypervisor. For instance, XMHF does not support multiple guests. This kind of drawbacks could limits the potential interest in MH, even they are able to introduce several important features.

5. ARCHITECTURE COMPARISON

The adoption of an extra layer of virtualization is capable to introduce a remarkable set of new features but it does not reduce the dimension of Trusted Code Base (TCB). MH could represent a new approach to improve the potential benefits provided by the adoption of nested virtualization. In the following subsections, it will presented the four possible architectures achievable "nesting" of GPH and MH, underlying potentialities, drawbacks and potential cost of implementation.

5.1 GPH Over GPH

The first analyzed architecture concerns only GPH. At first, it is important to recall the most interesting achievable by such architectures. In particular, several remarkable features are live migration over different platform, transparent migration of virtualized clusters between different data-centers (horizontal requirements DP1). Supposing the completeness and the correctness of nested virtualization hardware facilities in widespread hypervisor, GPH over GPH satisfies all horizontal design principles, but it lacks in vertical DPs as the size of TCB is the same of currently deployed architectures.

To sum up, it could be possible to achieve live-migration between different platform and other features related to VM management flexibility but it could not still be possible to reduce the TCB. Analyzing MH approaches, the aim is try at most to retain to features, trying to introduce isolation and privacy improvements due to the lower TCB promised.

5.2 MH Over MH

MH on MH in reported only to see the orthogonality between this approach and GPH over GPH. A theoretical implementation of MH over MH will provide the vertical requirements that are missing in the previous approach. The tiny core could be able to overcome monolithic problems due to privilege level of administration tasks (DP1), the possibility to improve the resilience of the platform to a large set of software failures (DP2) and finally reduce the possibility of the vulnerabilities of the most privilege part of the software (DP3). However, the interest on such architecture is limited because it is not able to overcome any cited issues of single-layer MH. Moreover, several horizontal properties are theoretically achievable but they will require a huge effort in developing functions of interest. Moreover, the lack of nested VMX functionalities prevents to concretely test this kind of architecture, because MH normally rely on VMX facilities in order to create a nested HVM guests. To conclude, nested virtualization is exploited in order to fill eco-systems gaps with GPH but MH over MH does not provide any improvement in this sense.

5.3 MH Over GPH

The third architecture, proposed in order to reduce security concerns, is built combining an GPH at L0 and a MH at L1. An interesting point of this approach is the fact of being easily be deployed, without requiring any extra code addition to the GPH, L0. Existing GPH-based platforms could potentially exports nested VMX features to those guests. Even if this is not so usual in modern cloud platform, this lack is considered temporary and related only to the experimental status of those features.

However, the amount of code running in L0 kernel space is still as big as before and, therefore, no TCB reduction. Finally, adopting a MH as L1 prevents to use inter-platform enhanced features. As analyzed when dealing with MH design, the price of having a tiny core is normally paid with a lack of certain functionalities. For example, XMHF only supports one single guest. It could be easily noticed that this approach is not architecturally able to provide any new "horizontal" features, leveraging on nested virtualization. Therefore, the architecture is only able to provide the same features that are provided by today deployed ones, with an enhanced but not sufficient level of isolation.

5.4 GPH Over MH

In this configuration, the TCB is smaller than other architectures, due to the modular structure of micro-kernels., executed at L0. Leveraging as L1 a GPH, like Xen or KVM, provides to the user an environment apparently identical to the one used in production today, satisfying DP5. The L1 GPH is normally integrated with modern cloud toolkits like OpenStack and will support 3rd part applications. Therefore, the architecture presents a stronger isolation leveraging on nested virtualization, providing in particular a strong tiny core(DP3). The MH layer will be transparent from the user and it will not not any change in the functioning of the platform.

The first class of possible drawbacks of this architecture is represented by the necessity to expose to the L1 nested hypervisor the nested VMX extension to let him "nest" another HVM guest. This is experimentally done by GPH like Xen and KVM, but it is generally not included in the essential features provided by a minimal micro-hypervisor. As reported in XMHF paper [12], the amount of lines of code for nested VMX is around 1KLOC. More generally, from an architectural point of view, the general MH minimalist attitude could represent a limitation to general preservation of existing platform functions. The constraint of privileged code size forces normally MH developers to drop some features, considered not essential. Therefore, several limitations could arise not only for nested virtualization proper features, but also for other class of features. For instance, XHMF is able only to virtualize one L1 instance at time. This is acceptable if the platform objective is limited to be provided of a secure core, but dramatically limits the possibility of having a platform model able to give the user the possibility to run its proper hypervisor as L1.

Another important class is the necessity to explicit hardware devices support. Device drivers represents a general Achilles' heel of MH, due the general cost of development. However, this class of problem could be addressed leveraging device assignment. L1

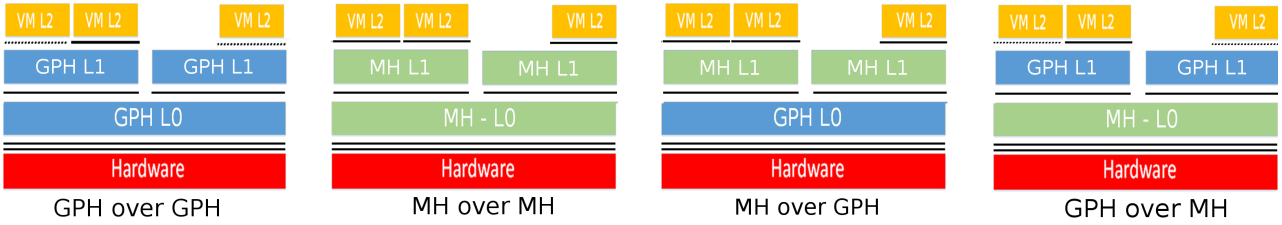

Figure 1:

Id	Functional design Principle	Class	GPH over GPH	MH over MH	MH over GPH	GPH over MH
DP1	User-Data privacy	Vertical	×	✓	×	✓
DP2	Fail-Safe modular architecture	Vertical	×	✓	×	✓
DP3	Small TCB	Vertical	×	✓	×	✓
DP4	Inter-platform support	Horizontal	✓	×	×	✓
DP5	3rd part compatibility	Horizontal	✓	×	×	✓
NDP1	Minimize code addition	-	✓	×	×	✓

GPH hypervisor will handle the real device, using the proper driver now demoted to run in a lower level of privilege. It has to be underlined that this solution could represent a feasible approach only if it possible to overcome the exclusive allocation trouble (SR-IOV).

5.5 Lessons Learned

To conclude, this section provides a short comparison between different platforms, contrasting architecture features above the two main different types of benefits given by nested virtualization, as reported widely in literature [13, 2, 17, 1]. The first could be identified in the augmented degree of isolation the architecture could enforce between host OS and the "customer" point of access, allowing to redesign the security architecture of the whole platform. The second one is represented by the possibility to leverage multiple virtualization layers to easily define inter-platform policy and operations between different platforms. In table 5.4, it is possible a summary of the previous analysis. To conclude, GPH over MH is the most interesting one in order to enlarge nested virtualization promised benefit. This architecture with a solid and secured MH as L0 is the only able to reduce the platform TCB and this could be a key point to obtain more protected infrastructure, . Firstly, adopting a MH as architecture trusted core, we will have a secure layer guaranteeing that administrator could not lose the control of the system. Klein et Al. [5] have demonstrated that a kernel with less than 10KLoC could be formally verified, under several conditions. This is far from be achievable today with standard GPH over GPH approach where the core is not tiny and the TCB is an order of magnitude bigger than MH [11].

However, the effort to set up a GPH over MH is strongly dependent on which technique leveraging for virtualizing L2. As stated in paragraph 5.4, with L2 guest virtualized with the hardware assitance technique, the most popular one, the MH needs to export nVMX at least. On the contrary, adopting paravirtualization/Dynamic binary translation for L2 does not necessitate a particular hardware support and so it could not be necessary patch the MH to set up a platform. However, it has not be underestimated, however, that para-virtualization supports only a subset of OS, and could not be considered a general solution and DBT has shown important overhead.

To sum up, the adoption of MH could increase feature set introduced by nested virtualization adoption. The first impeding problem is the implementation of MH, able to propose to guest to virtualize with hardware assistance. In next section, we present an implementation of GPH over MH, that try to circumvent this limitation leveraging paravirtualization.

6. SECURITY ARCHITECTURE

7. VERTICAL PROPERTIES

8. IMPLEMENTATION

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 Xen over NOVA [11].

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.

9. 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.

10. 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] O. Berghmans. Nesting Virtual Machines in Virtualization Test Frameworks, 2010.

[3] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-service cloud computing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 253–264, New York, NY, USA, 2012. ACM.

[4] B. Kauer, P. Verissimo, and A. Bessani. Recursive virtual machines for advanced security mechanisms. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, DSNW '11, pages 117–122, Washington, DC, USA, 2011. IEEE Computer Society.

[5] 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 *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[6] J. Liedtke. Improving IPC by Kernel Design. In *SOSP*, pages 175–188, 1993.

[7] J. Liedtke. On micro-Kernel Construction. In *SOSP*, pages 237–250, 1995.

[8] J. Nakajima. Making nested virtualization real by using hardware virtualization features, 2013.

[9] D. Perez-Botero, J. Szefer, and R. B. Lee. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *Proceedings of the Workshop on Security in Cloud Computing*, SCC, May 2013.

[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, 2009.

[11] U. Steinberg and B. Kauer. NOVA: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 209–222, New York, NY, USA, 2010. ACM.

[12] A. Vasudevan, S. Chaki, L. Jia, J. McCune, J. Newsome, and A. Datta. Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework. *2013 IEEE Symposium on Security and Privacy*, 0:430–444, 2013.

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

[14] D. Williams, H. Jamjoom, and H. Weatherspoon. Plug into the Supercloud, 2013.

[15] C. Wu, Z. Wang, and X. Jiang. Taming Hosted Hypervisors with (Mostly) Deprivileged Execution. In *Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA*, 2013.

[16] R. Yongjie. nested virtualizaiton test report for Xen 4.3-RC1.

[17] F. Zhang, J. Chen, H. Chen, and B. Zang. 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*, SOSP '11, pages 203–216, New York, NY, USA, 2011. ACM.