International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

A SURVEY OF THE HOST HYPERVISOR SECURITY ISSUES PRESENTED IN PUBLIC IAAS ENVIRONMENTS AND THEIR SOLUTIONS

Paul Cullum
Cyber Security Master's Degree
Manchester Metropolitan University / HKU Space
Hong Kong

ABSTRACT - The use of virtualization can be attributed to the success of cloud computing. However, usage of a hypervisor in a shared environment among mistrusting users presents significant challenges. This paper surveys the works on host hypervisor security issues presented in cloud computing, performing a short review of current literature on the subject. Addressing several key topics, namely threats and known attacks against the hypervisor or a virtual machine (vm) that exist in a shared environment.

This paper also contains a thorough review and comparison of the current solutions and proposed mitigations for the known attacks and identifies any potential gaps. Aiming to uncover if a hypervisor can provide the level of confidentiality, integrity and availability expected by cloud consumers. Research is critically analyzed and consideration for each solutions suitability of implementation in an Infrastructure as a Service (IaaS) environment is applied, including the impact on performance, if any.

KEYWORDS - Hypervisor security, virtualization, Cloud computing Security, IaaS security.

I. INTRODUCTION

Virtualization provides the foundation for many Cloud service providers (CSP). The core advantages and selling points of Cloud computing are well documented as elasticity, reductions in operational overhead and capital expense and scalability of resources. All of which are fulfilled with the help of virtualization in a cloud computing environment.

It is observed that when shifting from an On-Prem private cloud model to IaaS public cloud, the management and responsibility for the virtualization and physical server layer shifts to the CSP as per figure 1. It is this layer that is not natively accessible or even visible to the cloud consumer.

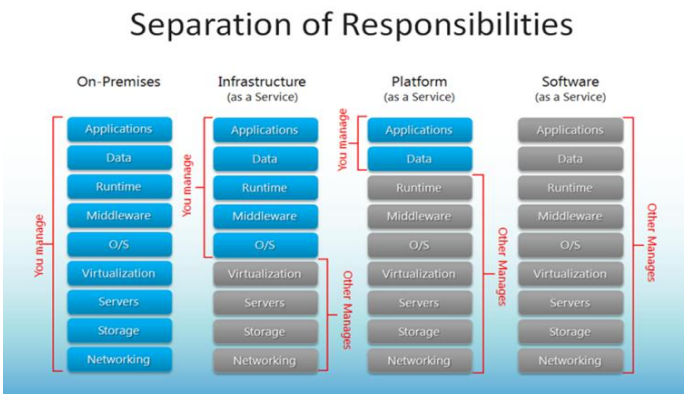

Figure 1. Shared responsibility model of cloud computing (Source **https://medium.com/@oscarpalaciosmontoya/cloud-models-and-the-shared-responsibility-in-public-cloud-c0a78e205369**)

Placing this level of trust in the CSP is often overlooked by cloud consumers and should be a key consideration when placing vm's that have sensitive data or critical workloads into IaaS public cloud environments. Cloud services by nature are always on and remotely accessible from the internet, the traditional network perimeter protection boundaries offered in an On Prem environment are either removed or operate in a reduced capacity.

1.1. Multi-tenancy

IaaS public cloud environments provide an economic, cost effective solution due to their multi-tenancy nature. Multi-tenancy refers to sharing the hosting infrastructure and the sharing of the address space across a number of cloud consumers. What this translates to in practice is that a cloud consumers vm, data or application could reside on the same physical machine or server as a malicious threat actor or even a malicious competitor. Consumers theoretically have

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

no visibility or insight into the precise location of their vm, data or application location nor do they have visibility into the other tenants sharing the physical host infrastructure alongside them. CSP's have a huge reliance on the hypervisor as the core security solution to isolate consumers from each other and ensure that data is not accessible between tenants. Due to this dependency on the security of the hypervisor a clear single point of attack is presented. Vulnerabilities do exist and will likely continue to exist, and so this expected, and often overlooked level of isolation, may not always be achieved or adhered to.

A large proportion of the cloud's value proposition is centered around the shared nature of cloud resourcing and the underlying hardware and compute such as memory, disk and CPU resources. This includes the capability to deliver resourcing both dynamically and on demand. Thus, empowering organizations to operate with a far greater level of flexibility than previously achieved with traditional environments. This transformation in which customers require flexibility whilst still maintaining a high level of isolation from malicious tenants or threats presents substantial challenges. Cyber criminals are presented with a unique opportunity due to the multi-tenancy that simply is not available in an On Prem private cloud [5].

1.2. Hypervisors

The issues and threats that pertain to hypervisors within public cloud computing also exist in private clouds, however the threats to virtualization are simply amplified in Cloud computing due to the shared infrastructure between untrusting customers.

The hypervisor is considered a lower level of the stack and cloud consumers do not have access or visibility at this level. However, the hypervisor is considered the layer of abstraction, providing logical separation across vm's for tenants. The hypervisor is used to manage and control all vm's, it is for this reason that the hypervisor becomes a single point of failure and provides a large attack surface within a cloud architecture. If a malicious actor were to compromise the hypervisor this would automatically result in the compromise of all the underlying vm's. Below figure 2 shows a typical hypervisor and vm relationship in a type 1 hypervisor often employed in IaaS architecture.

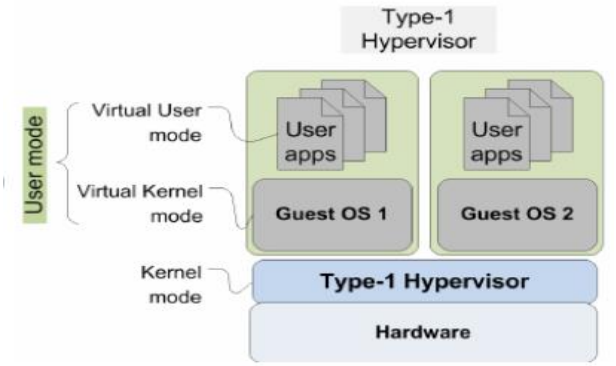

Figure 2: Type 1 Hypervisor high level view relative to user and kernel mode (https://www.researchgate.net/figure/Hypervisors-Type-1-and-Type-2_fig1_224202390)

1.3. Survey Aims

The aim of this survey paper is to collectively review the latest research into solutions that address known attacks against a hypervisor or known attacks leveraging virtualization presented in a cloud computing IaaS environment.

This survey paper takes the hypothesis that there is no single solution to address all known hypervisor threats presented in public cloud IaaS computing environments, and that a hypervisor alone does not provide effective isolation.

1.4. Methodology

The methodology of the survey paper will review current solutions or mitigations and measure them for effectiveness in mitigating the mentioned known attacks, taking into consideration the solutions impact on performance, if any, and a brief view of the feasibility for CSP's adopting or implementing the proposed solutions. Papers have been selected from tier 1 sources based on their forward-thinking ideas and relevance to address the known attacks and ability to build upon previous research.

II. THREAT MODEL

The cloud security alliance report 2019 [3] lists abuse in the cloud as one of the top 11 threats to cloud computing.
This survey paper will focus on the known attacks that exist in a public cloud IaaS environment due to virtualization and sharing of physical host infrastructure, resulting in the co-residency of vm's and the solutions presented to address the attacks.
The threat model will focus on attacks from a malicious vm breaking out of the isolation boundary and compromising the hypervisor, or an unsuspecting tenant's vm or administering

resource exhaustion-based attacks against other cloud consumers sharing the same physical infrastructure. The paper will also examine attacks that allow a level of information leakage from vm's or the underlying host infrastructure.

To provide focus, the threat model adopted will assume that the CSP and its administrators are trusted.

We assume that one or more cloud consumers are not trusted and may be malicious in nature, and attempt to compromise the confidentiality, integrity or availability of another cloud consumers virtual machine or data or the hypervisor itself. The cloud environment in scope is IaaS, in which the host (CSP) has very little, if any, authority over the actions and operations of its guests vm's.

The list of attacks in section 2 is not exhaustive and instead focusses on the important known attacks concerning hypervisors and virtual machines in IaaS public cloud environments. The known attacks listed are present in IaaS due to the extensive usage of virtualization within cloud computing and the shared hardware required to support this model. Physical attacks are not in scope for this survey paper.

2.1 Known Attacks

Vm escape

In this attack an attacker leverages a vm and interacts directly with the hypervisor to escape from its control. In a vm escape the vm crafts an attack to bypass the isolation between the vm's and the hypervisor. The attacker can gain or elevate privileges to access the resources shared with other VMs. Examples of this are Venom CVE-2015-3456. Vendors take this vulnerability very seriously as it is clearly a threat to virtualization. Microsoft have offered up to 250, 000 US dollars as a bug bounty for any proof of concept code that can demonstrate a successful vm escape exploit in their Hyper-V virtualization platform.

Hyper-jacking

A Hyper-jacking attack inserts a root kit that allows an attacker to control the hypervisor and in turn the entire virtual environment. This is achieved by inserting a thin malicious hypervisor on-top of the legitimate hypervisor. This represents a single point of failure as a compromise of the hypervisor would provide access to all the vm's that reside under it. Previous research in SubVirt [21] demonstrated how this would be achieved.

Denial of Service

Whilst a Denial of service attack (DoS) can take many forms, in the context of this survey paper a DoS attack refers to a malicious VM exhausting the hypervisor resources in its entirety and impacting the performance of vm's running on the same physical host. This attack is also known as a resource exhaustion attack, which impacts one of the core principles for information security, availability. An example of this attack is CVE-2017-17566,

Side channel

In a side channel attack the attacker first aims to co-locate a malicious vm on the same physical host as a victim vm and achieve co-residency. The attacker then constructs covert side channels to obtain sensitive information from the victim's vm. A side channel attack leverages a communication method not originally intended for the transfer of data. This attack has caused serious concern since the announcement in 2018 of the **Spectre** and **Meltdown** vulnerabilities, found in nearly all Processors which leverage a side channel to access memory locations. Side channel attacks work by converting leakage into usable information. Due to the non-standard use of side channel attacks they often can go undetected from intrusion detection systems (IDS). There are different categories of the attack relating to the method in which the exploitation is achieved. Cache attacks and Timing attacks are the most prominent and will be included in this paper. In addition to the well-publicized Spectre and Meltdown vulnerabilities, additional examples of a side channel attack are the RowHammer attack [16] CVE-2015-0565, in which memory bits are flipped to a new location to alter outcomes.

III. LITERATURE REVIEW

Section 3 includes the literature reviewed for the survey paper including current solutions with some references to previous research.

3.1 Co-Residency

Achieving co-residency relates to the planned placement of a malicious vm onto the same physical host as a victim's target vm in which to leverage one of the known attacks to attack the target vm. Co-residency forms a large part and a core requirement to initiate the known attacks mentioned in section.

Achieving targeted co-residency was successfully demonstrated in previous research [18]. In which Ristenpart. et al were able to demonstrate an ability to achieve targeted co-residency in Amazon's EC2 Cloud service (AWS). This was successfully implemented by mapping the AWS internal cloud infrastructure using the IP address space and usage of simple heuristics to determine accurate location of vm's, with the view that subnets are logically assigned between locations for simple and ease of administration. After the mapping launch of a malicious vm and a probing technique using network latency round trip time variation was used to

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

determine when co-residence with a victim vm had been achieved. Using this strategic approach of understanding Amazon's vm placement algorithm, demonstrated up to a 40% success rate. During the same attack the keystrokes were recovered from a co-resident vm. This is very important as it allowed a malicious attacker to locate a target and then issue some of the known attacks against a specific target, demonstrating a non-trivial method and exposing novel risks within IaaS public cloud environments. Once co-residence has been achieved a side channel attack can be carried out.

3.2 Side Channel Attack Solutions

The mitigations in [18] listed steps towards side-channel attacks focus on blinding techniques used to minimize the information leakage.

The blinding techniques are expanded and further enhanced in paper [13] which improved on previous research in [14] in which a side channel attack is performed against memory deduplication employed by hypervisors and the mitigations against the attack are offered. Hypervisor memory deduplication is a memory optimization technique introduced to provide memory cost saving through techniques performed by the hypervisor to identify and merge identical memory pages across the vm infrastructure [4] to reduce the physical memory footprint of vm's, as depicted in figure 3. The described attack allows a co-resident malicious vm to perform a timing-based side channel attack by exploiting the timing difference between a binary that is already running in the shared memory pages between vm's versus the difference of a new binary. The difference in timing reveals information on which applications or even software versions are running on a victim vm. Allowing a malicious vm to fingerprint running applications on a victim vm. The mitigations offered against this particular side channel attack in [13] are to de-activate memory deduplication on the hypervisor altogether and lose the cost saving features or implement encryption of vm memory which

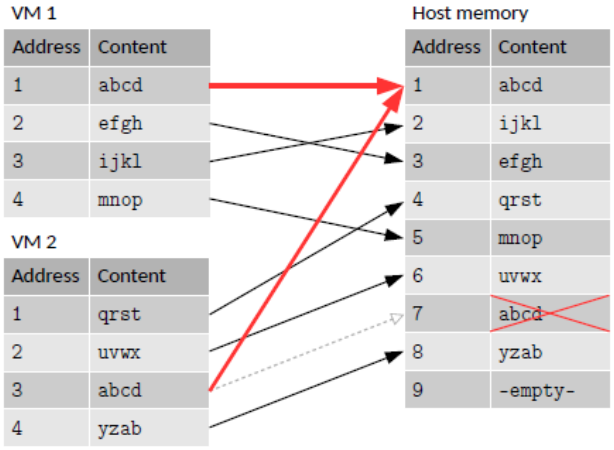

Figure 3 : Memory deduplication example

in turn would make it impossible for the hypervisor to perform any deduplication unless an encryption key is shared across the vm's. Whilst both suggestions are plausible this would present additional effort from CSP's. Requiring infrastructure changes and likely a large memory overhead to facilitate disabling memory deduplication, similarly enabling vm memory encryption would incur a larger memory overhead due to an inability to de-duplicate memory pages and would also require hardware infrastructure changes to perform the encryption all of which would need to be at the effort of the CSP potentially resulting in an increase in charge back to the cloud consumer.

[13] Also suggests an alternate counter measure is to obfuscate the vm memory by applying a randomizing technique, which would allow memory deduplication to be enabled, though using the fingerprinting technique implemented the victim vm's running applications could still occur, albeit at a much larger memory overhead with a requirement to probe ~5000 times as many signatures. Essentially slowing a malicious vm down in any fingerprinting activity but not eradicating the threat in its entirety.

An option not considered in [13] is to build upon previous research used in [2] and consider enabling deduplication on zero pages only. Applications and operating systems may zero out pages for usage in the future and this can be a frequent occurrence, although not for an extended amount of time. Whilst this provides less efficient memory optimization when compared with full memory deduplication, this method could reduce vm memory footprint without exposing a side channel.

Implementing the mitigations offered in [13] address a specific side channel timing attack in memory deduplication, but do not consider mitigating threats for other available side

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

channels that leverage the hardware design itself, such as an attack against the CPU cache or last level cache which is shared across all CPU cores to prevent memory bottle necks, figure 4 provides a visualization of the last level cache sharing between vm's. Research in [17] proposes an event driven solution that uses machine learning techniques to classify events and detect the probing techniques used within side channel attacks in a virtualized environment, specifically tested on the KVM hypervisor. Consideration for performance overhead and integration into KVM's existing architecture was applied and the system was built directly into the Linux kernel within KVM, reducing the complexity required and making good use of the default probes provided by the Linux distribution within KVM. The machine learning model trained for the monitoring system demonstrated in the results a clear capability to detect patterns of any side channel cache attack, casting a wide net on the ability for detection due to the persistent unique probing required in a side channel cache attack. The implementation can work directly within the host and further improved upon the research in [24] by negating the need for co-operation from the victim vm.

[17] mentions a small impact on the guest performance with almost zero impact to the host performance, though this has not been quantified in the paper. The model used in [17] leverages a trained class for known attacks and an untrained class, the latter is implemented to detect new side channel attacks that the model has not seen. The results demonstrate low false positives and low false negatives providing good future use to detect new unseen side channel attacks, examples of this could be any predecessors to Spectre and Meltdown. A clear draw-back is that a real-world implementation would only provide monitoring for a CSP and in its current state does not offer any prevention against a malicious vm from initiating an attack or ceasing an attack in operation, the CSP would need to implement corrective measures and the cloud consumers safety could depend on the CSP's ability to react promptly. No attempt was made to look at segmentation of the CPU last level cache between tenants and due to the threat model assumptions in [17], the solution could not cater for a compromised hypervisor.

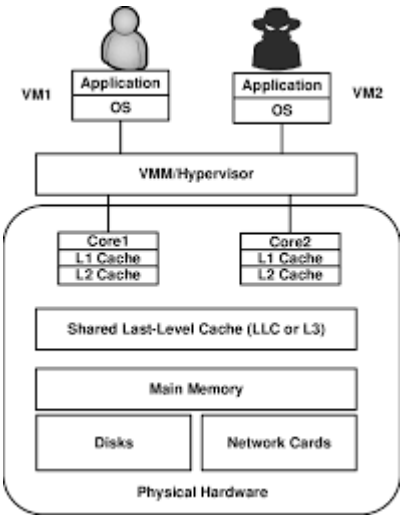

Figure 4: Last level cache sharing example (source https://www.researchgate.net/figure/Cross-VM-side-channel-attack-using-a-shared-last-level-cache_fig2_327314423)

3.3 Resource Monitoring

Cloud computing's always on nature is a key requirement for cloud consumers. It is therefore paramount that the availability of virtual machines is guaranteed and not negatively impacted by a malicious vm sharing the same physical host. In [25] a solution is implemented to detect and prevent a DoS attack initiated from a co-resident malicious vm. The solution consists of a set of testing programs to monitor a host's resource usage. [25] considers DoS attacks against memory, CPU, disk and network. The testing programs perform a set of repeated tasks for the hosts disk, memory and network to form a set of samples that are run on an offline host running the same hardware, this essentially creates a baseline. The testing programs then run on all hosts and a probability method is used to determine any deviations from the baseline that may indicate a DoS attack. After an attempted host DoS attack is detected the testing program will restrict resources from the suspected malicious vm for a short period, focusing on the area effected such as disk or memory, and if by performing the restriction the host performance improves then the identified vm is further confirmed as malicious and additional action is taken such as migrating the malicious vm to an alternate host or shutting the vm down.

The initial identification and pinpointing process indicates that thought has been given towards a method for managing false positives and not impacting an innocent vm that may have an irregular request for resources from the host. Evaluation on the impact of the testing programs performance overhead on the host has been applied and

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

results on monitoring memory, network or disk attacks does not incur any performance overhead. As part of the evaluation there has not been an attempt to see if multiple co-resident malicious vm's initiated simultaneous DoS attacks against the host how the testing program would handle this, as part of the threat model [25] assumes that only one vm per host is required to perform a DoS attack.

In [8] a solution is presented to detect DoS attacks even if the hypervisor has been compromised, secure resource allocation for the vm's is implemented by using a probing mechanism executed from the hosts CPU system management mode (SMM) which provides a higher privileged environment above that of the hypervisor ensuring protection and isolation against a compromised hypervisor. The SMM is normally used to handle privileged instructions outside of normal operation. The hypervisor is excluded from the trusted computing base (TCB) in this solution in which only the hardware and BIOS are included in the TCB. A sample-based approach performs random probing of the CPU and memory allocation, the probing program resides in System management RAM (SMRAM) installed within a customized BIOS on the host machine, though no hypervisor modifications are required. Near 100% accuracy is recorded for memory and CPU reporting.

The architecture also uses a dedicated proxy server to manage user requests to create new vm's , delete vm's and manage vCPU creation all of which are forwarded from the proxy to the hypervisor. The proxy is also used to check on resource usage and is connected via a serial device from proxy server to host. Architecture is shown in figure 5.

An out of band network channel is suggested as a further improvement option for bandwidth and scalability and to secure communication between the proxy and hypervisor, however beyond this no additional thought has been applied to further secure the proxy server which provides an attack surface if an attacker were to gain unauthorized access, whilst the hypervisor has been deprivileged in some aspects by leveraging the SMM, a proxy server has been introduced and further hardening of the proxy should be considered.

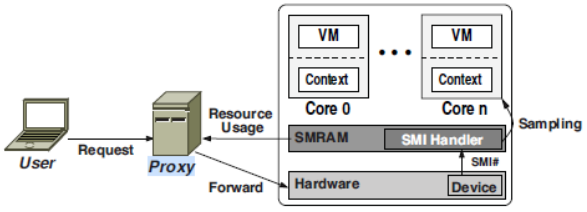

Figure 5: host resource accounting architecture

Performance overhead of the system is considered and normalized performance testing across a diverse range of applications running on the vm's was applied. For average workloads the performance overhead is between 1 – 2%, however for CPU intensive workloads the performance degradation is high with impact on the sampling.

It is also noted that unlike [25] there is no provision for the monitoring of disk or network usage within the solution and the focus is entirely on CPU and memory accounting. In the event the hypervisor was compromised in [8] the solution would still be able to continue probing for resource allocation, however based on the design it is probable that an attacker would be able to cause damage to the availability of the host or co-resident vm's in the event the hypervisor is compromised as this solution focuses on verifying CPU and memory resources under a vulnerable hypervisor rather than protection of vm's and the vm data as implemented in previous papers [15] .

3.4 Hypervisor modifications

Alluding to the simple idea that the hypervisor provides the attack surface and is the weakest link it has been suggested in NoHype [10] that removing the hypervisor from runtime is one approach that could be considered. NoHype architecture is proposed to modify and remove the hypervisor from runtime and therefore the requirement to defend the hypervisor. An attempt is made to preserve some of the connotations of virtualization whilst removing the interactions a vm can have with a hypervisor. Aiming to protect against vm escape and Hyper jacking known attacks.

In the NoHYpe architecture the codebase is minimized to reduce the attack surface. The view is that due to the multiple lines of code (LOC) in a standard hypervisor a larger attack surface is presented.

In the NoHYpe architecture each CPU core is directly allocated to a single vm, ensuring guest vm's cannot share CPU cores. Memory is also pre-allocated. A temporary hypervisor is used only to run at initialization, rather than the traditional vmm operating at run time.

Removing the hypervisor altogether whilst may seem plausible would require a drastic paradigm shift for CSP's. A clear draw back to this proposed architecture surrounds restricting the ability for guests to share CPU, which in turn is actually one of the major selling points of virtualization. Whilst performance is not measured in this paper, as resources are pre-allocated and this could result in underutilizing CPU and memory,

Flexibility and scalability of virtualization resources are lost due to the cost of a single core per virtual machine. The ability to over subscribe also is not possible with the NoHYpe architecture, as well as dynamic resource allocation. Some of the core advantages of cloud computing are removed, demonstrating a tradeoff between security and

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

economic value. In NoHYpe there is still a management vm that can provide an avenue to attack and impact the virtual environment.

An alternate idea in a solution named HypSec [12] is to modify and retro fit the existing KVM hypervisor rather than recreate a new hypervisor. Continuing with the theory in [10] and previous research in micro hypervisor, Nova, [19], that a large codebase within the hypervisor represents risk due to complexity and vulnerabilities. HypSec aims to reduce the trusted computing base (TCB) by leveraging microkernel principles. A trusted core, labelled corevisor, is created that provides access control to vm data and provides the CPU and memory virtualization. The hypervisor is partitioned into a non-trusted host, labelled hostvisor, performing hypervisor functionality with no access to vm data. Providing protection against a vm escape and hyper jacking attack.

Previous attempts in nested virtualization [22] that retro fit a hypervisor yet fail to provide full support of virtualization features are considered in HypSec. The viewpoint is taken that the hypervisors that are built on Linux (KVM and Xen) also inherit all of the vulnerabilities and bugs associated with the Linux kernel. HypSec' s design retains the Linux kernel and moves this to an untrusted hostvisor away from the corevisor. Isolation and protection of the trusted core is achieved using hardware virtualization, execution is at a higher level of privilege so that virtual machine exceptions are mediated and protect vm data is contained within CPU and Memory.

In the interest of protecting VM Data the corevisor intervenes privileged instructions, such as VM EXIT and the exit reason determines if the corevisor handles the exit directly or if there should be a switch to the hostvisor to handle. This step will provide a performance overhead when deciding to hand off or handle the instruction directly.

The transition from non-root mode to root mode is called VM EXIT and the opposite is called VM ENTRY. In the architecture for Hypsec the hostvisor is deprivileged and must call VM ENTER for the corevisor to execute a vm. Performance testing reveals up to a 19% performance overhead on the number of CPU cycles when HypSec is compared to a standard KVM hypervisor, this is due to the corevisor interposing on the trap and emulation and potential transfer to the hostvisor. Whilst this handoff to the hostvisor is considered to simplify the TCB there is a clear detriment to performance. Evaluation of practical attacks is also completed and compared with KVM, providing further protection against privileged escalation attacks that KVM are vulnerable to.

Whilst the privileged corevisor is deemed to be secure there is little evidence of further hardening to support this.

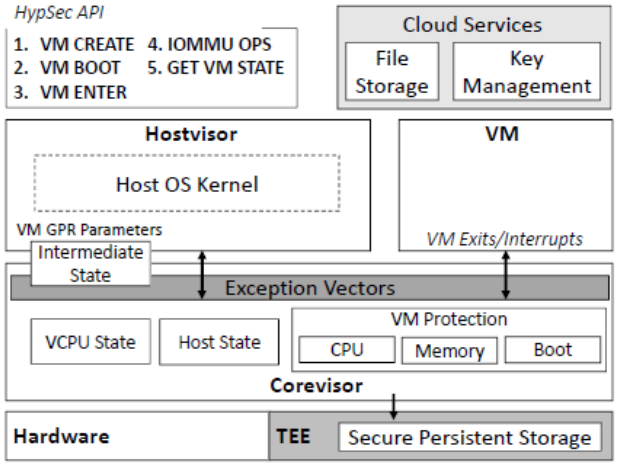

Figure 6: HypSec architecture

Similarly, to [12] control of privileged instructions in FWinst [7] the focus is on hardening the hypervisor to effectively secure all instruction emulation and the interaction of privilege instructions from an application on the vm in the ring 3 user space to the vmm in ring 0. This emulation is required as part of the standard trap and emulate behavior of virtualization in which sensitive actions are attempted by a vm and allow a vm to operate as if it is a physical machine. Actions such as direct access to the hardware, are trapped by the vmm to take control and then interact with the hardware as necessary and return the appropriate information to the vm. This emulation process is complicated and can be prone to errors which results in vulnerabilities that can lead to a vm escaping the isolation boundary and compromise of the hypervisor facilitating a vm escape or a host-based DoS, which FWinst can protect against. The emulator should validate privileged instructions and ensure they are executed from the correct ring code segment and not directly executed from a lesser privileged code segment such as ring 1-3.

Unlike Hypsec which selectively hands off to the corevisor for privileged instructions, FWinst uses an instruction filter, positioned between the VM Exit handlers and the instruction emulator (Figure 7). Based on a list of pre-programmed legitimate instructions for each context taken from the VM Exit reason. FWinst provides firewall like properties for privileged instructions from vm to instruction emulator. An inability to provide validation of privilege and prevent emulation of sensitive instructions from ring 3 has appeared as a common vulnerability in the Xen hypervisor as seen in CVE-2014-7155. From this list FWinst is able to filter out illegitimate instructions from a malicious vm and reduce the attack surface.

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

The overall performance overhead at runtime is stated as less than 2.5% using standard benchmarking tests. FWinst is implemented in KVM and performance tested with Windows and Linux vm's. It was proven in research from Amit et. Al [1] that an attacker can force any instruction, including non-privileged, to be emulated and so an approved filtered instruction list certainly has a place in further securing a hypervisor.

Challenges for the CSP would be in maintaining the list of legitimate instructions within FWinst if ever there were changes to architecture, similarly if unauthorized access to the hypervisor was gained there is an avenue to amend the list of legitimate instructions potentially undetected. FWinst uses only 279 LOC and it is also lightweight as per [12 and 10]. Whilst FWinst clearly addresses the known threat of a vm escape, the coverage of preventing a DoS attack against vm's in its entirety is questionable, if the hypervisor was compromised via a method other than vm escape this would allow a malicious actor to impact the availability of all vm's on the compromised host.

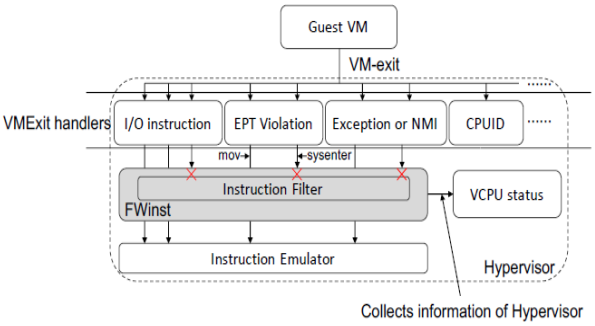

Figure 7: FWInst instruction filter architecture

In blind hypervison [6], the hypervisor is deprivileged, similarly to HypSec [12], so that in the event of compromise via a vm escape attack the underlying vm's data is inaccessible to the hypervisor. Focusing on the privacy concerns if a hypervisor was compromised that a consumers vm data would be safe. In this architecture a master server and one or more host servers are implemented. The master controls the overall global system state and the hosts have the deprivileged hypervisor installed. The master acts as an authentication authority responsible for the deployment of new vm's and ensuring confidentiality between the vm and hypervisor.
The master server can run on standard hardware, however the host requires specific hardware components that are used to encrypt and decrypt the vm data and loading them into memory, storing the private key. The hosts also require an extended memory management unit to further enforce isolation and creation of specific execution modes customized on the host for protection from unauthorized access. Symmetric keys are used to encrypt and decrypt memory between the master and the host to guarantee confidentiality of the vm data. Essentially by modifications in the hypervisor and the addition of the host memory protection of the vm's is achieved.
Implications into the performance overhead on the host and underlying vm's running in this architecture have not been evaluated. The design and research is focused towards protecting vm migration between hosts and communication across the network as all vm migrates can only be initiated from the master. There is no consideration into the hardening of the master which could provide a single point of failure within a cloud IaaS deployment. The additional hardware requirement in each host and software modifications may also present challenges for CSP's, coupled with administrative overhead to the manage and maintain the master servers. As per [12 and 7] there is focus towards control of privileged instructions, however the introduction of non-standard instructions into the hosts may also present supportability and compatibility challenges.

HyperPS [23] aims to implement a hypervisor monitoring solution that provides event monitoring of interactions between a vm and a hypervisor. An isolated space is created by separating privileges in which HyperPS monitors the hypervisor at runtime and is able to operate on privileges no higher than the hypervisor. The approach tackles the idea that when the kernel and hypervisor address space is shared additional security is required to protect vm's from a compromised hypervisor.
The standard hypervisor has the control of any security-sensitive system resources removed such as a VM Exit hypercall or manipulation of page tables or vm memory mapping. Hooks are used to forward privileged instructions such as VM EXIT from the hypervisor directly to HyperPS, essentially transferring the control flow and adopting similar focus on control of privileged instructions as observed in [6, 7, 12]. HyperPS uses an additional page table to achieve an isolated address space upon which to run, ensuring there is no mapping of virtual to physical in the hypervisor address range, preventing compromise of a hypervisor with known attacks vm escape and hyper jacking.

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

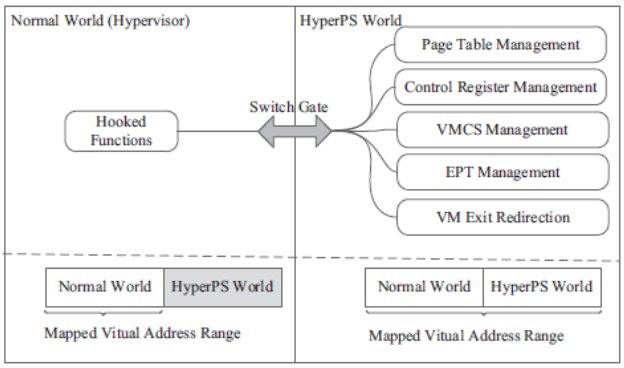

Figure 8. Hyper PS architecture

An impact on performance was evaluated and due to the requirement of the creation of a new memory page table for HyperPS and allocating memory pages an impact on the speed vm boot times is seen up to 1.09 times slower when compared with KVM. Relative performance testing on the whole system was achieved using application and micro benchmarks, up to a 4% performance overhead was observed. Additional hardening of the HyperPS world is not considered in the research and would also present an additional attack vector for a malicious tenant.

3.5 Hypervisor removal and vm modifications

Previous research in [15] proposed to modify the vm and implement a unikernel in which the user space and kernel space share a single address space. Unikernels, whilst running on traditional hypervisors such as KVM, restructure vm's into secure and flexible components by adopting a minimalist approach and provide the minimum number of libraries needed to run. The standard TCB is modified and the user space and security kernel space share a single address space on a lightweight vm. Within the single address space, the typical drivers or support library functions expected within a standard OS are excluded and only the required libraries and functions needed to run the application are included, and nothing else. A small monitor process is required to interact and manage privileged instructions or hypercalls between the unikernel and hypervisor and hardware for I/O. This was considered an improvement to the security of vm's by reducing the attack surface when compared with a traditional vm due to the absence of drivers or unnecessary I/O commands and non-essential libraries. Unikernel supporters also argue that due to the single address space applications no longer run in ring 3 and now operate in ring 0. Negating the need to switch contexts between the traditional user and operating system boundary seen with a standard TCB and is recognized as allowing operations to run faster providing a performance increase.

[21] aims to build upon the Unikernel principles in [15] and implement unikernels to run as processes whilst retaining the isolation benefits from their vm- like properties. In [21] the solution proposes to remove the hypervisor completely. This further enforces the principles used in NoHype [10] by completely removing any trace of the hypervisor, instead of simply at runtime. If the hypervisor does not exist then the vulnerabilities pertaining to a hypervisor should be removed, resulting in mitigation against Hyper jacking and vm escape. [21] proposes to implement a tender process, depicted in figure 9 which builds upon the ukvm used in [15]. This tender process also contains the unikernel and maintains exit handling for interaction with I/O devices directly and setup of the unikernel. Multiple copies of unikernels can run on a single host and the tender process is able to ensure that the unikernel binary is shared in memory by the host for memory efficiency across unikernels. A whitelist of system calls in the kernel is used to maintain isolation. This aims to improve on the considered limitations of hardware virtualization and the use of a hypervisor with techniques such as memory ballooning which although provide memory efficiencies, operates at the expense and trade off of extra CPU cycles.

[21] also aims to improve on the design in [15] that requires a switch between the context of the vm and monitor, an increase in hypercalls to perform the context switching was found to more than double the CPU cycles when compared with a direct function call as introduced by implementing the unikernel into the tender process and removal of the hypervisor. Ultimately removing the context switching results in better performance and is demonstrated in the comparison tests which show a faster startup and more efficient utilization of memory and CPU when compared with traditional unikernels. In [21] thought was considered to evaluate isolation capabilities of unikernels as processes when compared with traditional unikernels. The metrics adopted for evaluation check how much of the host kernel access is needed to function correctly and how much of the kernel a unikernel *could* access.

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

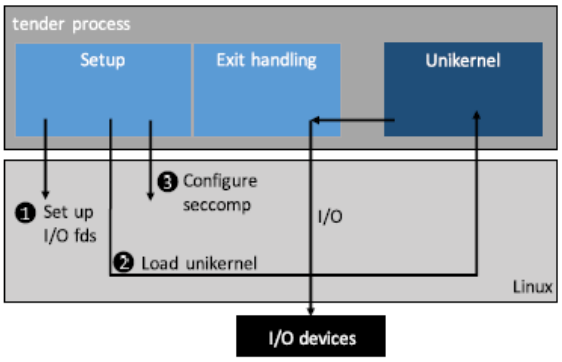

Figure 9: Unikernels as processes architecture

Results are reported as unikernel running as processes requiring half as much access to kernel functions as standard unikernels.

Whilst implementing unikernels as processes and omitting the hypervisor sounds promising, the paper excludes how supportable this would be. Unikernels are essentially black box implementations and the ability to support and debug would present CSP's challenges. Removal of the hypervisor in an IaaS public cloud would require a huge architecture overhaul for CSP's.

Implementing unikernels as processes and removing the hypervisor would also not provide protection against side channel attacks such as the Spectre and meltdown vulnerability, which even in the Unikernel [15] or unikernel as processes [21] architecture could be exploited.

The unikernel design is in direct conflict with the principle of least privilege and prevents application developers from running the standard user protection boundary due to the single address space sharing, which takes an application from the standard ring 3 and essentially moves it into ring 0. The notion that a unikernel is more secure due to the limited codebase whilst may be correct for the unikernels does not factor in the Linux machine kernel vulnerabilities and the attack surface this presents.

The reduced codebase in Unikernels was found in [20] to be missing basic security features, such as an ability to update whilst running and input validation failures leading to buffer overflow vulnerabilities. Raising concerns that whilst the hypervisor is removed additional vulnerabilities are introduced, coupled with a lack of supportability from CSP's and a complete infrastructure overhaul requirement.

3.6 Survey summary

Figure 10 is used to better visualize the coverage of the presented papers, and each solutions ability to address all the known threats presented in section 2 in their entirety.

	Paper Number									
Known threat addressed	[13]	[17]	[25]	[8]	[10]	[12]	[7]	[6]	[23]	[21]
Vm Escape					√	√	√	√	√	√
DoS			√	√			√			
Hyper-Jacking					√	√		√	√	√
Side Channel	√	√								

Figure 10: Table of all surveyed papers solutions and their threat coverage.

A full table summarizing the work is shown in appendix A at the end of this paper.

IV. FUTURE WORK

In this survey paper, we describe the known attacks against a hypervisor and the relevance of this in the context of a public IaaS environment.

Based on the survey's findings the gaps and opportunities and valuable future work could include:

The Inclusion of I/O encryption built into the hypervisor and included as a standard among all vm's, this coupled with an ability to share memory pages. Research into slicing CPU last level cache to ensure this cannot be used in a side channel attack.

There are clear gaps in the full coverage of the proposed solutions. Combining some of the proposed solutions into hypervisors as standards to provide full coverage against the known attacks presented in this paper.

Continued usage of machine learning techniques to future proof against unknown attacks based on classification and anomaly as seen in solution [17].

Transparency from CSP's into the tenants that occupy a physical host so that the decision and ultimately the risk is offloaded to the cloud consumer to decide if they wish to decline sharing with a particular tenant due to competitive reasons or a conflict of interests.

V. CONCLUSION

Ensuring the security of cloud consumers is paramount. Threats to hypervisor security exist in all environments that leverage virtualization, including private clouds, however this threat is exacerbated in multi-tenancy environments due to the shared nature of the hosting infrastructure.

The prevalence of side channel attacks demonstrates that isolation is not achieved at the hardware or hypervisor level.

Mechanisms that have been implemented to improve performance across hardware and scalability across shared

resources are often exploitable. The survey paper demonstrates that there is often a trade-off between performance and security.

For guaranteed security against attacks from malicious vm's cloud consumers should resort to avoiding co-residence. Customers with strong privacy requirements could consider the economical trade off against security and request dedicated hosts in an IaaS environment or consider if the workload is more suitable to a private cloud.

Solutions that aim to remove the hypervisor as a threat by various methods of deprivileging, often introduce an alternate to the hypervisor and in turn introduce an additional attack vector.

Clear trends towards controlling the flow of privileged instructions using variations of trap and emulate between the vm and hypervisor were employed in the surveyed papers, often at the expense of extra CPU cycles and ultimately resulting in degraded performance for the host or vm's.

As demonstrated in the literature review there is no single solution that can address all known attacks within a virtual environment. The hypervisor should not be trusted as a level of isolation by cloud consumers, who should take effort to further scrutinize CSP hypervisor security in more detail when deciding to move workloads to an IaaS environment.

As with many items in Information Security an approach that is layered and applies a defense in depth methodology, leveraging a multitude of solutions will likely yield the best results.

VI. REFERENCES

[1] Amit N., Tsafrir D., Schuster A., Ayoub A., and Shlomo E.
(2015). *Virtual CPU Validation*. In Proceedings of the 25th Symposium on Operating Systems Principles , (Pg311–327). https://doi.org/10.1145/2815400.2815420

[2] Barker S., Wood T., Prashant S., and Sitaraman R. (2012). *An Empirical Study of Memory Sharing in Virtual Machines.* https://www.usenix.org/system/files/conference/atc12/atc12-final226.pdf

[3] Brook Jon-Michael C. et al. (2019). *CSA Releases New Research - Top Threats to*. Cloud Security Alliance: https://cloudsecurityalliance.org/articles/csa-releases-new-research-top-threats-to-cloud-computing-egregious-eleven/

[4] Chang C., Wu J., and Liu P. (2011). *An empirical study on memory sharing of virtual machines for server consolidation.* In IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2011, (Pg 244-249).

[5] Chen L., Nhien-An Le-Khac and Takabi H. (2019). *Security, Privacy, and Digital Forensics in the Cloud*. Wiley, New York, NY.

[6] Dubrulle P., Sirdey R., Dore P., Aichouch M., and Ohayon E.. (2015). *Blind Hypervision To Protect Virtual Machine Privacy Against Hypervisor Escape Vulnerabilities* https://ieeexplore.ieee.org/document/7281938/

[7] Ishiguro K. and Kono K. (2018). *Hardening Hypervisors against Vulnerabilities in instruction emulation.* DOI: https://dl.acm.org/doi/10.1145/3193111.3193118

[8] Jin S., Seol J., Huh J., and Maeng S. (2015). *Hardware-Assisted Secure Resource Accounting under a Vulnerable Hypervisor.* DOI: https://dl.acm.org/doi/10.1145/2731186.2731203

[9] Jin S., Ahn J., Cha S., and Huh J. (2011). *Architectural Support for Secure Virtualization under a Vulnerable Hypervisor* ACM: https://jeongseob.github.io/papers/jin_micro11.pdf

[10] Keller E., Szefer J., Rexford J., and Lee R.B. (2014). *Eliminating the Hypervisor Attack Surface for a More Secure Cloud* https://eric-keller.github.io/papers/2011/ekeller_nohype_ccs11.pdf

[11] King S. T. (2006). SubVirt: *Implementing malware with virtual machines.* IEEE Symposium on Security and Privacy 2006: (Pg 314-327) http://web.eecs.umich.edu/~pmchen/papers/king06.pdf

[12] Li Shi-Wei, Koh John S., and Nieh J. (2019). *Protecting Cloud Virtual Machines from Hypervisor and Host System Exploits* https://www.usenix.org/system/files/sec19-li-shih-wei.pdf

[13] Lindeman J and Fischer M. (2018). *On the detection of applications in co-resident virtual machines via a memory deduplication side-channel* https://dl.acm.org/doi/10.1145/3307624.3307628

[14] Lindemann J. and Fischer M. (2018) Efficient identification of applications in co-resident vms via a memory side-channel. In ICT Systems Security and Privacy Protection - IFIP TC 11 International Conference (SEC) 2018, (Pg 245-259).

[15] Madhavapeddy A. et al. (2013). Unikernels: Library Operating Systems for the Cloud. ASPLOS Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems http://unikernel.org/files/2013-asplos-mirage.pdf

[16] Mutlu O. and Kim Jeremie S. (2019). *RowHammer: A Retrospective*. https://arxiv.org/pdf/1904.09724.pdf

[17] Paundu Ady Wahyudi, Fall D., Miyamoto D. and Kadobayashi Y. (2018). *Leveraging KVM Events to Detect Cache-Based Side Channel Attacks in a Virtualization Environment*. https://doi.org/10.1155/2018/4216240

[18] Ristenpart T., Tromer E., Shacham H., and Savage S. (2009). *Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds*. https://doi.org/10.1145/1653662.1653687

[19] Steinberg U. and Kauer B. (2010). *NOVA: a micro hypervisor-based secure virtualization architecture*. https://dl.acm.org/doi/10.1145/1755913.1755935

[20] Talbot J. et al. (2019). *A Security Perspective on Unikernels*. https://arxiv.org/pdf/1911.06260.pdf

[21] Williams D., Koller R., Lucina, M. and Prakash N. (2018). *Unikernels as Processes*. SoCC '18: Proceedings of the ACM Symposium on Cloud Computing October 2018 DOI: https://doi.org/10.1145/3267809.3267845

[22] Zhang F., Chen J., Chen H., and Zang B. (2011). *CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization*. https://dl.acm.org/doi/10.1145/2043556.2043576

[23] Zhang K., Liu W., Lin K., and Tu B. (2019). *HyperPS: A Hypervisor Monitoring Approach Based on Privilege Separation*. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00141

[24] Zhang T., Zhang Y., and Lee R. B. (2014). CloudRadar: *A Real-Time Side-Channel Attack Detection System in Clouds*. http://palms.ee.princeton.edu/system/files/cloud_detect.pdf

[25] Zhang T and Lee R.B. (2017). *Host-based DoS Attacks and Defense in the Cloud.*. http://palms.princeton.edu/system/files/cloud_destroy_v6_0.pdf

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 8, ISSN No. 2455-2143, Pages 36-48
Published Online December 2020 in IJEAST (http://www.ijeast.com)

APPENDIX

A Full table summarizing literature review.

Threat addressed	Solution Name and reference	Solution	Performance Overhead	CSP Impact
Memory Deduplication Side Channel	**On the Detection of Applications in Co-Resident Virtual**	Disable Memory deduplication	Memory cost savings from deduplication.	Large infrastructure changes required.
		Memory obfuscation		
	Paper [13]	Encrypt vm memory		
Side channel cache	**Leveraging KVM events to detect cache based side channel attacks.**	Monitoring using events in KVM leveraging machine learning to detect known and unknown side channel attacks.	Zero impact to host performance, small impact to guest	Process change for monitoring events and responding to incidents.
	Paper [17]			
DoS	**Host-based DoS Attacks and Defence in the cloud 2017**	Set of testing programs used	No performance over head	Compatibility checks into real usage of hypervisors
		Able to monitor CPU, disk and network.		
	Paper [25]			
DoS	**Hardware-Assisted Secure Resource**	Leverage SMM internal hardware and a proxy.	High performance degradation on high CPU processes and workloads	Infrastructure changes to faciliate
		Deprivilege the hypervisor and control.		
	[Paper [8]			
Vm escape and hyper jacking	**No Hype**	Hypervisor removed from run time to reduce the attack surface.	Not measured	No dynamic resourcing available
				Direct vm ti CPU core mapping required
	Paper [10]			Memory is also pre allocated. Benefits of virtualization are lost
Vm escape	**HypSec**	Retrofit a hypervisor, reduce the TCB. Split between a privileged corevisor and a deprivileged hypervisor known as hostvisor.	Up to 19% due to corevisor trap and emulation.	Hypervisor changes required to retrofit KVM
Hyper jacking				
	Paper [12]			
Vm escape	**Fwinst - Hardening the hypervisor**	FWInst Use a privileged instruction filter to handle VM Exit from vm to hypervisor.	Less than 2.5%	Maintenance and supportability of the instruction filtered list.
DoS				
	Paper [7]			
Vm escape hyper jacking	**Blind Hypervisor**	Uses a master and a host. Requires additional hardware components to encrypt and create non-standard privilege instructions.	No check on performance in paper.	Non-standard instructions. Master server and additional specific hardware required for encryption on the host server.
	Paper [6]			
Vm escape	**Hyper PS**	Creates a new address space for a privilegd world, known as HyperPS world. Uses hooks and a gateto forward privileged instructions from the hypervisor to HyperPS world.	1.09 times slower vm start-up.	New virtual memory address space to manage per host.
Hyper jacking			Up to 4% performance overhead.	
	Paper [23]			
Vm escape	**Unikernels as processes**	Complete removal of the Hypervisor	Improvements in performance	Removal of all virtualisation, significant challenges in supportability and complete architecture overhaul
Hyperjacking		Running of Unikernels as processes using a tender process that interacts direclty with the Linux OS.		
	Paper [21]			