symmetry

MDPI

Article

Tenant-Oriented Monitoring for Customized Security Services in the Cloud

Huaizhe Zhou [1,†], Haihe Ba [1,†], Yongjun Wang [1,*], Zhiying Wang [1], Jun Ma [1], Yunshi Li [2] and Huidong Qiao [1,3]

1 College of Computer, National University of Defense Technology, Changsha 410073, China; huaizhezhou@nudt.edu.cn (H.Z.); haiheba@nudt.edu.cn (H.B.); zywang@nudt.edu.cn (Z.W.); majun@nudt.edu.cn (J.M.); qiaohuidong13@nudt.edu.cn (H.Q.)
2 Northwest Institute of Eco-Environment and Resources, Chinese Academy of Sciences, Lanzhou 730000, China; liyunshi@lzb.ac.cn
3 College of Computer and Communication, Hunan Institute of Engineering, Xiangtan 411100, China
* Correspondence: wangyongjun@nudt.edu.cn
† These authors contributed equally to this work.

Received: 30 December 2018; Accepted: 13 February 2019; Published: 18 February 2019

Abstract: The dramatic proliferation of cloud computing makes it an attractive target for malicious attacks. Increasing solutions resort to virtual machine introspection (VMI) to deal with security issues in the cloud environment. However, the existing works are not feasible to support tenants to customize individual security services based on their security requirements flexibly. Additionally, adoption of VMI-based security solutions makes tenants at the risk of exposing sensitive information to attackers. To alleviate the security and privacy anxieties of tenants, we present SECLOUD, a framework for monitoring VMs in the cloud for security analysis in this paper. By extending VMI techniques, SECLOUD provides remote tenants or their authorized security service providers with flexible interfaces for monitoring runtime information of guest virtual machines (VMs) in a non-intrusive manner. The proposed framework enhances effectiveness of monitoring by taking advantages of architectural symmetry of cloud environment. Moreover, we harden our framework with a privacy-preserving capacity for tenants. The flexibility and effectiveness of SECLOUD is demonstrated through a prototype implementation based on Xen hypervisor, which results in acceptable performance overhead.

Keywords: virtualization; cloud security; virtual machine introspection; cloud monitoring; privacy

1. Introduction

Despite the proliferation and popularity of cloud computing, security and privacy threats have been unfortunately endlessly emerging and have been an obstacle for further usage of cloud computing [1]. Due to the inherent deficiency of unsafe languages or implementation complexity, almost all large real-world services running in the cloud always come with its own set of vulnerabilities. They could be the target, as well as the source, of malicious attack to adversaries. A compromised application or virtual machine (VM) will be used to exploit other VMs on the same physical platform. Even worse, the openness of cloud environments exacerbates the problem. Protecting VMs from advanced, sophisticated attacks is a highly urgent task.

Unfortunately, conventional security solutions fail to fulfill the task since they reside inside the VMs they protect. They are inadequate for thwarting emerging advanced malicious attacks and susceptible to be circumvented or controlled by sophisticated adversaries. Baliga et al. [2] have demonstrated how easy to launch such an attack by manipulating Linux Netfilter to remove hook

functions to packet filtering. To address that, researchers and practitioners have proposed a stream of "out-of-the-box" solutions [3–8]. They leverage the ability provided by virtualization, called virtual machine introspection(VMI), to deploy monitoring and protection components in the hypervisor or a privileged VM, which makes the security solutions more robust. Despite the efforts existing research has made, security and privacy issues in the cloud remain challenges for these problems:

Inflexible support for customized security functionalities—Security requirements of tenants are often at odds with cloud providers' management routine. Typically, cloud providers offer unified security functionalities for all tenants due to the efficiency. However, an "one size fits all" security solution is unacceptable to all tenants in practice. For example, an IDS service that checks network packets for malicious content using simple signatures in a VM is not applicable to another VM that receives encrypted packets [9]. Under current provider-controlled service model, the cloud provider own the hypervisor and management VM, which are necessary for VMI techniques. The previous VMI-based security solutions have to rely on the cloud provider to develop and deploy in the cloud. Thus tenants have no access to the privileged VMI interfaces to customize individual security functionalities. To reconcile this conflict, it calls for a mechanism that provide tenants with access to the VMI interfaces to deploy customized VMI-based security tools underneath their VMs.

Trustworthiness and privacy issues—In current cloud computing platforms like Xen (XEN. http://www.xen.org), KVM (KVM.http://www.linux-kvm.org/page/Main_Page), and VMware (VMware.http://www.vmware.com/), the Cloud Service Provider(CSP) holds the hypervisor-related operations necessary for VMI-based security tools. Given the possibility that malicious administrators or attackers could tamper or reveal sensitive data of guest systems by abusing VMI interfaces, privacy concern of tenants also arises for that the majority of security services need to access sensitive data of tenants. In the case of private information leakage, various mechanisms have been proposed to secure guest VMs in the cloud [10,11]. However, such mechanisms always prevent monitoring of VMs, which is imperative for security functionalities.

By analyzing the issues above further, we observed that the defects of the existing solutions mentioned above result from the lack of a feasible system to support tenant-oriented security monitoring. To this end, this paper introduces SECLOUD, a security monitoring framework to support tenant-oriented security functionality in the cloud environment. SECLOUD provides monitoring interfaces to tenants and deploys a proxy in the privilege VM to monitor the runtime information of guest VMs on behalf of tenants. By this way, tenants are able to have access to the runtime information of remote VMs and customize individual security functionalities assisted by the VMs on behalf of tenants or themselves further. Unlike the agent-based system for remote security monitoring, the proposed framework works in a non-intrusive way by extending VM introspection techniques, which makes it more robust. To preserve the privacy of tenants, SECLOUD is equipped with the capacity of keeping confidentiality from potential attackers. Additionally, the proposed framework improves effectiveness and gains some compelling benefits enabled by taking advantages of virtualization technique and centric location. We argue that SECLOUD throws light upon a better way of system monitoring to support customized security functionalities in the cloud, such as collaborating with the managed advanced security services.

To demonstrate the effectiveness and applicability of the SECLOUD framework, a proof-of-concept prototype was implemented of the proposed technique with Xen hypervisor. The implementation does not require fundamental changes to the cloud platform and the guest system, which makes it more general for use.

To summarize, this paper makes the following contributions:

- Proposing a novel monitoring framework to support tenant-oriented security functionality by exploiting the problem of existing security solutions in the cloud. The proposed framework enables tenants to obtain runtime information of their VMs for specific security analysis by extending virtual machine introspection techniques.

- Hardening the proposed framework by keeping confidentiality of obtained runtime information of the guest system with a cryptographic scheme. The encrypted introspection could prevent the privacy of tenants from inside or outside attackers during the whole procedure.
- Enhancing the proposed framework with a set of optimized features by taking advantages of architectural strengths of hypervisor-based security and centralized administration. The enhanced features enable tenants to gain a more comprehensive view of guest systems and improve security protection capabilities further.
- Implementing a prototype system of the proposed framework based on Xen hypervisor. The utility of the prototype implementation of SECLOUD was demonstrated by various evaluations.

The remainder of the paper is organized as follows. Section 2 explains the motivation for this work with a research background. Sections 3 and 4 show the details of the design and implementation of the proposed framework respectively. Section 5 presents the evaluation of the framework and Section 6 summarizes the previous work and compares with the work in this paper. Finally, Section 7 concludes the work in this paper.

2. Motivation and Background

2.1. Background

Computer system monitoring is a fundamental mechanism for maintaining systems security. In addition to dramatically changing the way the computer system operates, virtualization techniques have also pushed the system monitoring out of the VM in the cloud. Virtual Machine Introspection (VMI) [3], an emerging technique which empowered by the features of virtualization techniques, can inspect and interpose guest VMs outside them without guest interference or enforced guest VM cooperation. Compared with the conventional monitoring system which should hook or install agents in the guest system, VMI-based tools have advantages of strong isolation while keeping visibility to the guest system. However, VMI has an inherent problem, semantic gap, which has been the main motivation for a significant portion of research over recent decades [12]. LIVEWIRE [3] leverages debugging information generated by a modified `crash` tools to interpret the raw binary data. XenAccess [13] bridges the semantic gap by extracting knowledge of the monitored kernel manually. Process Implanting [14] narrows the semantic gap by replacing the code of a guest process with its own utilities. VMST [12] automatically bridge the semantic gap by an online binary code reuse approach. At present, with recent advances in forensics tools, the semantic gap problem can be considered as a solved engineering problem [15].

In view of its compelling advantages over agent-based monitoring, VMI has been exploited to support a wide range of use cases. A production-oriented IDS prototype was proposed in [16] by combining VMI with forensic memory analysis (FMA). HIMA [17] leverages VMI to build a hypervisor-based integrity measurement agent with both strong isolation and consistency capabilities. Srivastava et al. [18] propose a white-list based application-level firewall in a virtualized environment which performs introspection for each new connection to identify the process bound to the connection. Ahmed et al. [19] show HookLocator to real-time monitor the integrity of Windows kernel pools based entirely on virtual machine introspection. Stackdb [20] monitors and controls multiple targets in a system with VMI support. NFM [21] provides a non-intrusive, out-of-band and "always-on" cloud monitoring system with standard and straightforward interfaces by leveraging VMI. DRAKVUF [7] builds a dynamic malware analysis system which provides a VMI enhances its abilities on the latest hardware virtualization extensions and the Xen hypervisor. VMI strengthen it with a stealthy and in-depth view into the behavior of malware. VEDefender [22] uses a hardware-based approach to acquire the physical memory of the host machine. XScope [23] detects malicious applications with memory introspection. Proskurin et al. also [24] try to use stealthy memory introspection on the ARM platform.

Although there have been a variety of proposals based on VMI for security issues in the cloud, none of them could support diverse security requirements efficiently. Most of them are designed as a standalone system countering a series of specific threats. Lacking generality and scalability impair their utility gravely in practice.

2.2. Motivation

To mitigate security and privacy threats in the cloud, different types of security techniques have been proposed by academia, industry and open source communities. Most cloud providers have also developed an increasing array of protection products to meet the needs of tenants for securing their VMs. Despite this, the security features of the cloud platform remain limited in practice. A study [25] conducted in 2012 showed that none of the five cloud service providers had a response to DDoS attacks lasted 21 days. Furthermore, they always offer a uniform solution to a specific security issue, which varies very little for different tenants on the same cloud platform. What is expected by the tenant is to provide in the security as a service model, which similar to what is provided for performance management. However, in the current cloud platform, this is difficult to be realized. This situation could be attributed to the inflexibility of previous works in their design and implementation.

In the cloud, the separation between control and usage complicate the development of security services for securing guest VMs. In the current cloud computing environments, the cloud provider supplies tenants with necessary hardware resource and maintains their VMs. Their primary tasks are accountable for securing the underlying infrastructure and the management systems which guest VMs running on it, rather than securing software and data inside guest VMs. Additionally, the cloud provider does not and should not be aware of the internals of guest VMs in consideration of privacy issues. It would be an excellent barrier for the cloud provider to monitor and protect a guest system in fine granularity. On the other hand, the tenant has no saying on security services provided by the cloud provider for an absence of control of physical infrastructure. The tenant cannot claim his every security policy on the software stack running in his/her guest VM being enforced by the cloud provider. In addition, the multi-tenancy features of cloud makes matters worse.

With the ubiquitous and convenient network, managed security services have become more and more popular to be provided in the security as a service model. They could enhance cloud users' experience by offering more secure, flexible, and automated security management for applications deployed on cloud infrastructures. These security services provide users the ability to monitor the health and protection of their systems. They can analyze their virtual networks, memory, and applications for vulnerabilities; continuously monitor for attacks, intrusions, viruses, and application integrity. Such a model not only provides organizations of all sizes with access to the technology services that they need, but it can also be a much more cost-effective way of accessing services than performing functions in-house. By outsourcing security protections to a security service provider, users can gain consistent and cost-effective protections regardless of device types, users locations or operating systems [26,27].

Enlightened by this, tenants resort to managed security services to find a way out of their dilemma of VMs protection in the cloud [28]. The tenants could subscribe to different security services based on their security requirements. However, such techniques can be problematic as they require the installation of agents on the monitored system. The agents are at risk of being subverted by malware even though they run at the system's highest privilege level. The tenant could not strengthen the agents unless a higher privileged method is available, e.g., virtualization-based method. Unfortunately, it is neither supported by the modern hypervisor nor supported by the existing VMI-based works to strengthen the agents at a lower layer. Existing VMI-based security solutions are always designed specifically for a single functionality [3,6,17]. Moreover, it is a pain point that provisioning and maintaining multiple agents for a single tenant.

The work carried out in this paper is designed to support flexible security services for tenants efficiently, meanwhile, alleviate privacy concern of tenants about misusage of VM information. More specifically, the study aims to achieve the following goals.

- **Flexible monitoring under tenant control:** The proposed technique should enable tenants or their authorized security providers to have fine-grained access to the runtime information of their VMs in the cloud. It depends on the tenant rather than the cloud provider to decide who can access and what can be accessed.
- **Tamper resistance:** The proposed technique should be free from compromising or evading by sophisticated attackers. Our monitoring process should be in a non-intrusive way which does not involve guest cooperation. We can offer security services the genuine state of a guest system.
- **Privacy preservation:** The proposed technique should keep tenant's sensitive data from being snooped on by both malicious administrators and outside attackers. We should preserve the privacy of a tenant through all the procedures, including data obtaining and transmitting.
- **High-efficiency:** The proposed technique should be capable of handling multi-tenancy of the modern cloud environment and increasing security functionalities.
- **Generality of usage:** The proposed technique should not couple with the specific cloud platform deeply. The hypervisor does not need to be modified in our approach, nor does the guest system. It would be a portable architecture that can, with little effort, be reused on other cloud platforms.

3. Design of SeCloud

This section presents the concepts used to design SeCloud, as well as the architecture for this system. First, we provide an overview of SeCloud and its deployment scenario in the cloud. Second, we detail how the proposed framework supports flexible security service for tenants. Lastly, we enhance our framework by taking advantages of virtualization-based approaches and adding a mechanism for privacy preservation.

3.1. Threat Model and Assumptions

SeCloud aims to provide remote tenants with flexible system monitoring so that they can deploy individual security services to protect their VMs hosted in the cloud. In such a way, we assume an adversary could gain the highest privilege of guest VM by exploiting a weakness in userspace software and escalating to root via a kernel vulnerability. The adversary could execute arbitrary malicious code in the VM and evade or disable the protection mechanism in the VM. Similar to the assumption in [9], we also distinguish cloud service providers from cloud administrators. We consider that a cloud service provider is trusted and has a vested interest in protecting guest VMs hosted on his platform from being compromised. He is willing to equip his infrastructures with necessary hardware such as the Trusted Platform Module(TPM) chip, which would be used to guarantee the integrity of the hypervisor via various techniques [29]. On the other hand, we do not trust the cloud administrators who could be malicious for financial profit or honest but curious. They could abuse the privileges of the management VM to steal sensitive data related to the privacy of tenants.

We assume an adversary without the ability to tamper with any hardware resources of the cloud platform physically, and the physical attacks such as cold boot attacks are out of the scope of this paper. We assume that regular administrators do not have `root` authorization on compute nodes. We also expect the integrity of our monitoring component residing in dom0 though it is not advisable. However, it is not a fatal defect of our framework because we can harden it by leveraging existing techniques such as XSM [30]. Our design is described under the assumption that the information gathered by VMI is genuine. We acknowledge the possibility that a sophisticated adversary can manipulate kernel data to make VMI invalid. Previous works have already addressed this problem (e.g., [7]), and such attacks are out of the scope of this paper.

3.2. Overview

The primary goal of the work is to enable tenants or their designated security service providers to monitor corresponding guest VMs flexibly. With SECLOUD, we achieve this goal by building a set of well-defined security-related VMI operations and exposing them to tenants and their designated security service providers in the form of interfaces. With the assistance of the interfaces, security services designated by the corresponding tenant could inspect the guest system without any agent installed inside guest VMs.

Architecturally, the overall framework, depicted in Figure 1, can be viewed as three inter-associated components locating in different places across the tenant and cloud side. Each of these components is detailed in the context of the Xen VMM-based environment next.

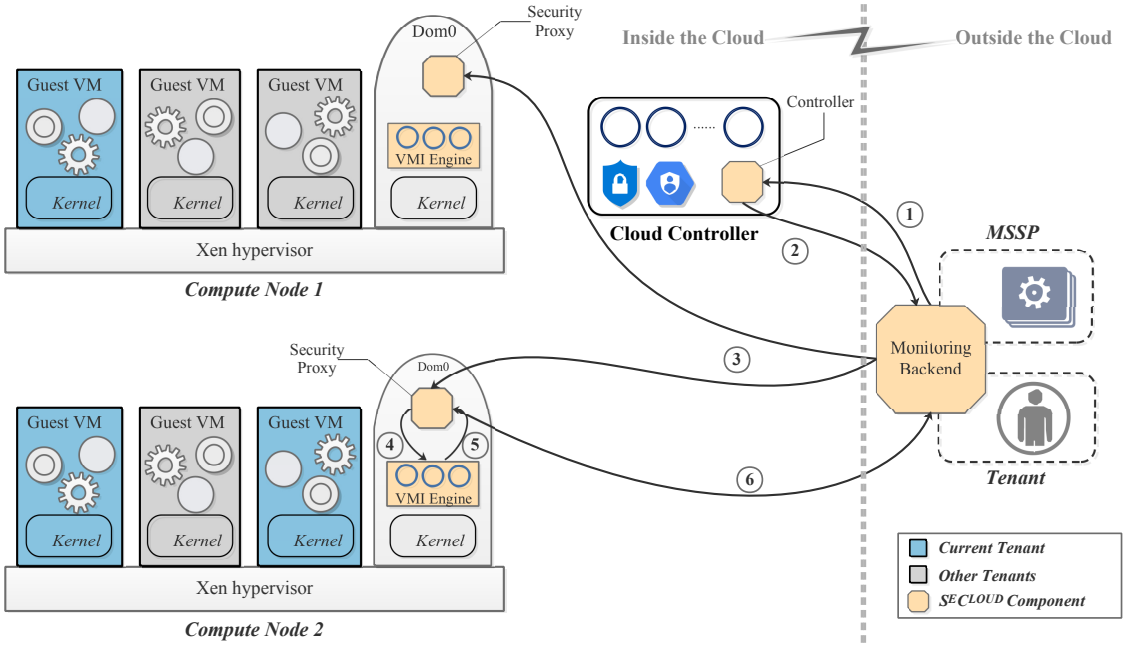

Figure 1. Overview of SECLOUD architecture. We use MSSP as the abbreviation of managed security service provider.

Monitoring backend. The monitoring backend is a tenant-controlled client to interact with the remote server. It is used by the tenant or his authorized security service provider to acquire runtime information of guest VMs. ① When a security service needs to inspect the guest system for malicious activities checking, the monitoring backend initiates a request to the cloud controller. Once the cloud accepts the request, the monitoring backend will gain necessary information about the target VM. ② Then the backend sends commands to the security proxy in the dom0 of the compute node where the target VM locates. Finally, the result will be returned by the security proxy through a secure channel.

Controller. The controller is responsible for security service authentication and authorization with the help of a cloud management system. ③ When receiving a request for inspecting VMs on its platform, the controller depends on the management tools of the cloud platform to locate the computer node of the target VM. In addition, it checks the permission of the security service against an access control list based on the credential pre-distributed. If the authentication succeeds, the controller will authorize the access to the security service.

Security proxy. The security proxy acts as the frontend of VM monitoring on behalf of the remote security service. ④ The security proxy parses the commands from remote security service and redirects them to the VMI engine. ⑤ Then the VMI engine executes corresponding operations requested by the remote security service. After the desired results are acquired, the VMI engine returns them to the

security proxy. ⑥ In addition the security proxy send the results to the remote security services with integrity checking padded.

3.3. Supporting Customized Security Service Efficiently with SECLOUD

We revisit the motivation of the work here to illuminate how SECLOUD could be used to address existing problems of securing guest VMs in the current cloud environment. As depicted in Figure 1, our framework is comprised of cloud side and tenant side components. A two-way connection is established between components for information exchange. A tenant can inspect his VMs by invoking interfaces exposed by SECLOUD, just as what he does with "in guest" agents. Thus the tenant could develop his customized security service and make it work properly by interacting with a monitoring backend of our framework. The self-build security service supported by our SECLOUD would neither rely upon the cloud provider nor upon the "in gues" assistance, which endows it with the quality of flexibility and tampers resistance. Furthermore, the tenant could delegate his privilege to a trusted third party, such as a managed security service provider (MSSP). It will enable some heavy-handed, complex security analytics, such as malware detection based on deep learning. Compared with previous solutions for securing guest VMs, we could provide more flexible and powerful protection capabilities over guest VMs with underlying support of SECLOUD. As for the efficiency issue, we consolidate similar operations for inspection and provide unified interfaces to all the security services. By this way could we eliminate redundancy between multiple security services which is an inherent defect of standalone VMI-based solutions and agent-based solutions.

Case study. To clarify how our framework could be used in modern cloud environment more intuitively, we illustrate it with the use case we mentioned in Section 1. Suppose two tenants host several VMs on the same cloud platform. Tenant *A* deploys web services in his VMs for the public. Tenant *B* deploys some corporate services in his VMs for the staffs in his company only. The corporate services are accessed via encrypted communication channels, e.g., TLS-based channels, for the reason of trade secrets. However, there are still potential attacks in case of account theft. Both of them need to secure their VMs with IDS services. We can accomplish this by implementing different monitoring strategies for different tenants. For tenant *A*, we need to inspect network packets for analysis of malicious activities. For tenant *B*, we need to extract the master key of a TLS connection at runtime using memory introspection first [31]. Afterward, we can decrypt the TLS channel and analyze packets. As a result, both security requirements of different tenants are satisfied.

3.4. Privacy Preservation with Trustworthiness

Security service always needs to access sensitive data of guest systems which acquired by introspection utils in our framework. As all the related operations proceed in the cloud environment, where distrusted administrators and outside attackers could exist, privacy concern arise inevitably from tenants. On the other hand, given the possibility of the gathered data being tampered with by malicious administrators or attackers, the tenant also concern about the credibility of collected data of runtime information. To alleviate the concern of tenants, we endow our framework with the ability to protect the privacy of tenants and the credibility of gathered data.

In our framework, we propose a cryptography-based scheme concerning protecting the privacy of tenants. In the process of inspecting guest VM, we encrypt all the obtained data. When the security proxy obtains runtime information data request by remote security service, it will encrypt them with a negotiated key. Such a method could preserve the confidentiality of information data when they store in our framework and transmit to remote security services. Thus we can prevent both malicious administrators and attackers from stealing private information of tenants. As for the credibility compromising of obtained data, we add an integrity verification mechanism on the basis of scheme above. Our whole scheme is illustrated with the protocol in Figure 2.

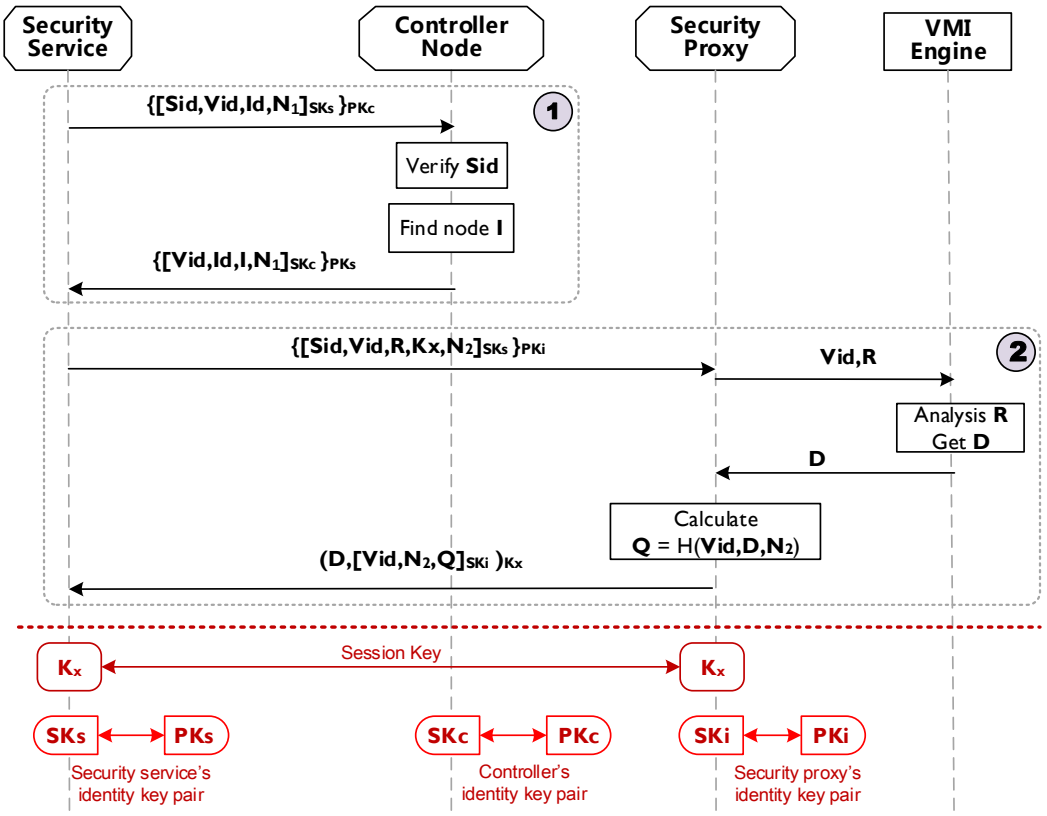

Figure 2. Protocol of privacy preservation and tamper-resistance for security relatd data. We use the notation $[M]_K$ for a private key operation with key K, M_K for a public key operation with key K, and $(M)_K$ for a symmetric key operation with symmetric key K.

The protocol is comprised of two parts, which are marked with ❶ and ❷ in Figure 2. The first part is the procedure of authentication and authorization of remote security service by cloud controller. Initially the security service sends cloud controller the access request, which including the security service identifier **Sid**, the VM identifier **Vid**, the tenant identifier **Id**, and a nonce $\mathbf{N_1}$. Then the cloud controller checks this information to verify if the service **Sid** has access to the tenant **Id** and if there is a VM **Vid** associated with this tenant. Once the verification succeeds, the cloud controller will authorize privilege to security service and send the address of the computer node where VM **Vid** located. In the second part, security service sent request **R** of runtime information access to the security proxy in the compute node **I**. Then the security service invokes the VMI engine to obtain runtime state data of VM **Vid** requested by **R**. At last the security proxy attach a signature to obtained data **D** by calculating the hash value of (**Vid**, **D**, $\mathbf{N_1}$) and return the encrypted result to the security service.

In our designed protocol, the sensitive data of guest systems are encrypted by a symmetric key $\mathbf{K_x}$ which generated by security service randomly. Neither malicious administrators nor outside attackers could inspect sensitive information. By calculating the hash value of obtained data and signing them, we could guarantee the integrity. Additionally, two different nonces $\mathbf{N_1}$, and $\mathbf{N_2}$ are used to prevent replay attacks over channels between security service and different cloud nodes. A combination of the two schemes above could prevent obtained data being tampered with and thus improve their credibility.

3.5. Enhanced Functionalities

In addition to supporting tenant-oriented monitoring in the cloud, our design builds up some compelling features by taking advantages of architectural strengths of hypervisor-based security and centralized administration. These features open new opportunities for supporting enhanced functionalities and bring tenant more powerful security protection.

3.5.1. Cross-VM Visibility

Compared to the conventional way of integrating third security services with installing agents in the VMs, SECLOUD locate in a place beneath all guest VMs. The work of monitoring and inspecting guest VMs is consolidated and executed centrally at VMM layer. The previous VMI-based solutions do not share acquired state of a VM with each other. With increasing requirements of tenants for security functionality, the complexity of management and maintenance will burden the cloud administrators severely. Furthermore, security services for different guest VMs are isolated, and there is no mechanism for interacting or sharing threat intelligence. In our framework, the monitor component could collect system data from multiple guest VMs instead of individual VMs separately. We can enable across-VM analytics by aggregating collected state data, which will benefit any security service for detecting distributed attack mode.

There are two critical insights behind our design of this new feature. First, malicious activities are becoming more sophisticated in the cloud environment. There are typically more than one VM involved in an attack incident. For instance, Distributed Denial of Service (DDoS) attack, a large-scale coordinated attack, is supported by botnets consist of numerous infected VMs. It is difficult to detect the distributed malicious operations in the individual VMs separately. To identify these types of attacks efficiently, it is necessary to combine the evidence of suspicious activities from multiple distributed VMs. The second one is the homogeneity of the cloud environment. More specifically, a group of VMs is assigned the same set of tasks in a particular tenant deployment, such as database queries in big data processing. This task specialization nature of the cloud makes these VMs exhibit the same behavior model. If one of these VMs is compromised by an unknown attack, it will be detected by cross-VM analysis effectively since its behavior diverse from the rest of the VMs.

3.5.2. Multilayer Visibility

The security services rely on monitoring tools to infer the behavior of guest VM and then decide the legality of it. The effectiveness of them depends on how accurately the behavior is described. Benefiting from virtualization technology, SECLOUD has not only the fine-grained visibility into the guest systems, but also the access to network level activities information. Thus we can get a comprehensive runtime state of guest VM, which can be used to profile complicated guest VM activities more accurately.

Compared to previous work [32], our design for this feature is more than a simple collection of various information from different layers. We combine all the knowledge of system state and integrate them into a new dataset with information correlation. Our method of information correlation is depicted in the Algorithm 1.

When a new network connection is found, the IP address is extracted to find the target VM whose IP address matches the extracted one. Then, SECLOUD identifies the socket associated with the network connection in the target VM. Finally, SECLOUD obtain the information of the process by iterating over the list of active processes in the target VM and checking each process to see if it holds the socket.

Our proposed approach could increase the dimensions of system state data rather than data volume only. We can precisely portray the behavior of a suspicious entity in the VM to allow further analysis for various security services.

Algorithm 1: Correlation of multilayer information.

Input : A network connection information N_i
Output: A integrated dataset with information correlation M_i

if N_i *is inbound network traffic* **then**
  extrat the destination IP address IP from N_i
else
  extrat the source IP address IP from N_i
end
find the VM VM_t whose IP address is IP;
obtain the list of local sockets SL in VM_t;
repeat
  fetch a socket s from SL;
  if *the port s bound to match with* N_i **then**
    $R_i \longleftarrow$ reference to s;
  end
until *reference* R_i *is not NULL*;
obtain the list of active process Ps in VM_t;
set the $flag$ with $FALSE$;
while $flag = FALSE$ **do**
  obtain a process p from Ps;
  traverse the structure of p and extract socket sc it holds;
  if *sc match with* R_i **then**
    parse p to extract process id pid and process name pn;
    add a tuple <pid,pn,N_i> to M_i;
    set the $flag$ with $TRUE$;
  end
end

4. Implementation of SeCloud

We implemented a prototype of SeCloud based on Xen virtualization platform. As described in Section 3, our overall implementation consists of three entities: (i) Security Proxy in the management VM, including the VMI engine inside; (ii) Controller in the cloud controller node for authentication and authorization; (iii) Monitoring Backend for tenants or their security service providers.

4.1. VM Monitoring for Security Service

To assist remote security service to secure guest VMs, SeCloud should provide fine-grained monitoring over the target VMs. We achieve this by developing two components inside the management VM. The monitoring components are illustrated in Figure 3. The security proxy acts as the server for responding to tenants monitoring requests. It parses the commands from tenants and relays them to the VMI engine with corresponding parameters. When getting monitoring results from the VMI engine, the security proxy calculates a hash value of obtained data with SHA-256 and encrypts them with AES-CBC algorithm first, and then send them back to the remote backend.

The core component for monitoring is the VMI engine, which leverages introspection techniques to inspect the guest system and acquire corresponding information requested by remote tenants. In addition to common functionalities required by security analysis, the framework provide a LibVMI-based library as well. It equipped our framework with extensibility to implement tenant specified monitoring functionality with our exposed library. In our implementation, the monitored resource in the guest VM mainly includes virtual memory, virtual disk, and network.

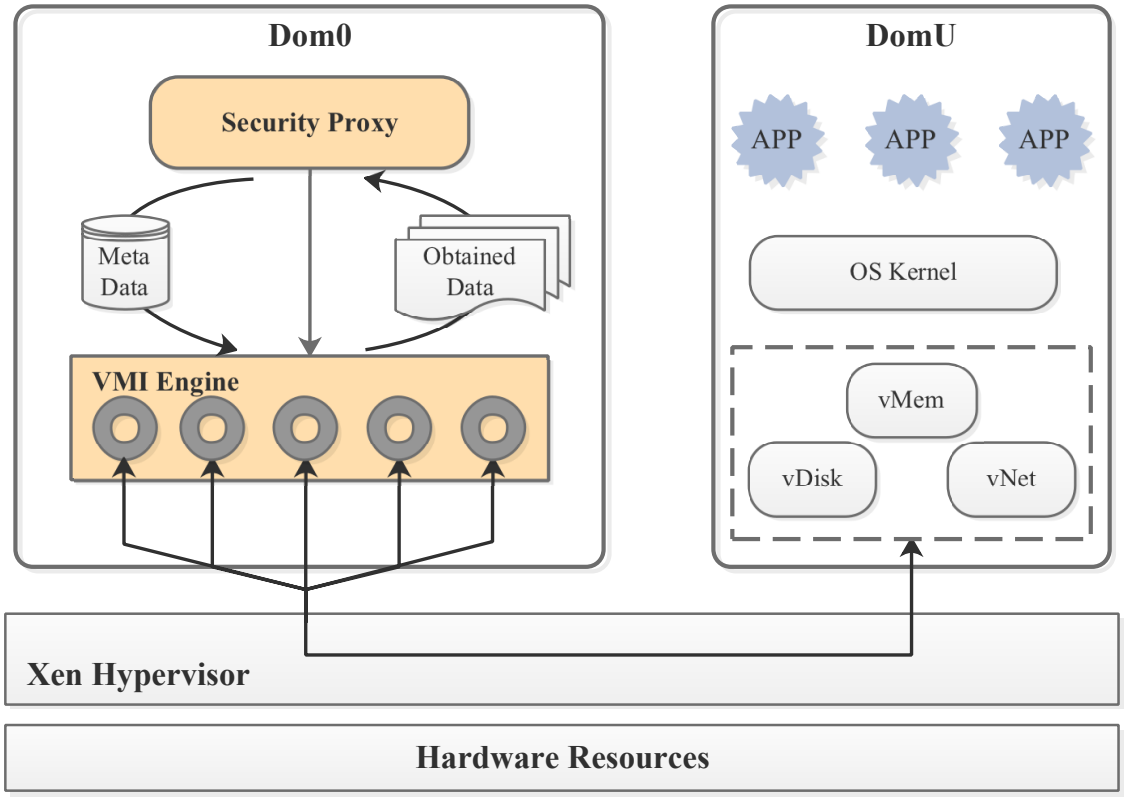

Figure 3. Implementation of SECLOUD Architecture.

4.1.1. Memory Monitoring

The memory contains the major activities of a guest system and is always the primary concern of security analysis. In our implementation, we access to guest virtual memory from the management VM through the assistance of LibVMI library [33]. In essence, LibVMI library leverages some low-level APIs of Xen (such as `xc_map_foreign_range`), which map arbitrary memory pages of a guest VM into the management VM. LibVMI provides abundant APIs for interacting with guest VM (pause/resume), inspecting guest memory, inspecting guest registers, and monitoring guest state. It makes it greatly easy to access low-level details of guest VMs in our implementation with LibVMI library and its Python bindings.

When we get the raw byte array from guest virtual memory by VMI, we need to interpret them into structured runtime information of the guest system. The raw memory covers both kernel data and process areas. Kernel data structures such as `task_struct` for processes, retain the major runtime information of the guest system. The agent-based approach simply extracts system information by interacting with the OS context, such as `/proc` or `/sys`. In the case of our framework based on LibVMI, we fulfill this by leveraging location information gleaned from guest's debugging symbols. We can gain this information assisted by tenants from the metadata they submitted along with their requests as shown in Figure 3. The helpful information includes the running kernel version, the target VM architecture, the starting address of various structures and the field offsets of relevant fields in the structures. A typical approach to extract the desired kernel object from linked lists and binary trees raw memory can be summarized as:

1. Finding the exported kernel symbol address of statically addressed kernel object as the starting point.
2. Locating the relevant structure member by adding relative offsets with the starting point.
3. Traversing the linked lists of objects via pointer fields (`prev, next`) until reaching the desired kernel object instance.

As for process memory, we can traverse per-process page table to carve up its memory areas. These areas can be accessed by tenant defined routines which have specialized knowledge about user processes.

In addition to basic structured information interpretation, we also implement advanced memory inspecting functions by combining LibVMI with forensic software. We use Rekall (Available online, https://github.com/google/rekall) to support robust introspection. By using Rekall profiles, LibVMI can bypass the use of the in-memory KdDebuggerData (KDBG) structure customarily used by memory forensics tools and thus allows introspecting domains where this structure is either corrupted or encoded (like in the case of Windows 8 $\times$ 64). Furthermore, the LibVMI Python bindings enable us to utilize Volatility (Available online, https://github.com/volatilityfoundation/volatility) for memory inspection by integrating an address space plugin into Volatility. We can gain higher level semantic information which would aid significantly in performing useful memory analysis tasks. For Extensibility, it is also convenient for security service providers to develop their plugins with the interfaces to access memory information provided by Volatility.

4.1.2. Network Monitoring

In our implementation for monitoring network activities, we use the bridge mode network connection, which is the default network connection mode and widely used in Xen VMs. Dom0 provides a virtual Ethernet bridge to multiplex and demultiplex packages between the physical network card to all virtual network devices provided by Xen to the DomU under bridge mode.

For the generality of usage, we intercept all the network packets by hooking the virtual bridge in the Dom0 kernel space instead of modifying the VMM to capture network traffic. In our present implementation, we use a modified ebtables (Available online, http://ebtables.sourceforge.net/) to intercept packets on the bridge. The ebtables is a filtering tool for a Linux-based bridging firewall. It enables transparent filtering of network traffic passing through a Linux bridge. To separate obtained packets for different users, we use the ulog mechanism of ebtables to pass packets received by the bridge to the user-specific storage entity created for each virtual nic of user VMs.

4.1.3. File Monitoring

To avoid the semantic gap between low-level disk traffic and high-level file system-oriented view, we implement files monitoring different from previous work [13], which must have a built-in understanding of each file system format.

The hypervisor we used, i.e., Xen, leverages QEMU to emulate the VM disk. When a VM request for a file operation, the driver in Dom0 converts the block device I/O request to a file I/O request in Dom0. We implement our file monitoring based on the network block device(NBD), which could mount the disk image as a virtual device in the management VM. To this end, we need to create a nbd device under `/dev/`, and then bind it to the disk image of a VM using `qemu-nbd`. As a result, we can perform file system operation directly in the management VM by standard filesystem methods. However, we should pay attention to that the mounting of a disk image for a VM must remain in a read-only manner since the image has been mounted to the guest filesystem and cannot be modified simultaneously.

4.2. Controller of SeCloud

In our current implementation, we use OpenStack as the cloud management software. OpenStack (OpenStack.https://www.openstack.org/), an open-source cloud computing platform, is popular among many large organizations. It comprises various components addressing necessities of a cloud system. We focus on the computing component Nova and the identity service Keystone in our framework. We deploy all the components in the same machine for convenience, which would not impact the practicability of our framework in the real world. For the generality of usage, we try to avoid modifying the management system too much. We leverage existing functions of OpenStack to

satisfy our requirements. By registering our monitoring service in the Keystone, we leverage Keystone API for authentication and authorization of tenants or security service providers. The tenants could interact with management by the user interface for authentication. The credentials of tenants are distributed in advance.

4.3. Monitoring Backend

The backend is used for interaction between security proxy in the cloud and tenants. It comprises three primary functions: sending monitoring command to a remote server; getting results of monitoring for security analysis; setting up secure channels with cryptographic operations. At present we implemented the backend with ZeroMQ (https://zeromq.org/) python library, which is a high-performance asynchronous messaging library for distributed applications. Tenants can use the interfaces of the backend to start a remote introspection procedure according to their requirements. To realize secure communication with integrity validation, we use the AES-CBC algorithm to encrypt and decrypt the state information of guest VMs with a symmetric key provided by the tenant. In addition, we also use SHA-256 to ensure the integrity of the acquired data.

5. Experimental Evaluation

In this section we present our evaluation of the SECLOUD prototype implementation. Our evaluation was conducted in three aspects. We first tested the effectiveness of our framework through a set of security services with different requirements of monitoring. Then we measured the performance of our framework compared with the previous cloud monitoring system. At last, we evaluated the security properties of our framework.

Our experimental setup consists of a physical server and a tenant host which were connected with a 200M-Ethernet switch. The physical server used to simulate cloud server was furnished with two Intel Xeon E5-2640 v2 @2.0GHz processors, 20GB RAM and 1TB HDD. We ran Xen 4.8.0 hypervisor and a Ubuntu 14.04 dom0 with eight virtual CPUs and 20GB memory assigned. The guest VMs for experiments were fully virtualized Ubuntu 14.04 and Windows7-sp1, each of which was assigned with 1 vCPU, 1GB virtual memory, and 10GB storage. For a remote tenant host, we used a PC with Intel i7-4510U CPU, 8GB RAM, 500GB HDD, and Linux Mint 18.2 inside of it.

5.1. Effectiveness Evaluation

To confirm that SECLOUD can provide tenants with flexible monitoring of guest VMs effectively, we evaluate our implementation with three different test scenarios driven by different security requirements. First, we evaluate the capacity of monitoring different guest systems to satisfy individual security services. To evaluate the generality of usage of our framework further, we also test it with with a real-world security service. Next, we evaluate the enhanced functionalities of our framework for improving effectiveness.

5.1.1. Supporting Customized Security Services

The goal of our work is to build up a monitoring system which underpins tenants to secure their remote VMs with customized security services. To verify the effectiveness of it, we ran two VMs with different operating systems to emulate different guest system of different tenants in this test scenario. Tenant `A` deployed a host-based IDS system in his PC to protect his remote VM with Windows-XP running inside. Tenant `B` deployed a remote attestation service assisted by a trusted third party (TTP) to check the kernel integrity of his ubuntu VM. We implemented a monitoring backend for each of the tenants. The backend for tenant `A` monitor virtual memory with SECLOUD to find suspicious activities in the VM. The backend for tenant `B` requests binary of his system kernel and measures its integrity by calculating the hash value of it periodically.

To validate that the security services above worked correctly, we ran different malware against the security service in the target VMs. We ran `hexdef100`, a Windows rootkit based on Windows NT, in the

Windows VM. When `hexdef100.exe` started to run, it rewrote the memory of running processes and hid its system components, such as the files, processes, system services, system drivers, the registry, and opened port. As a result, we could not find it in the Windows `taskmgr`. However, when the IDS service scan the guest memory by invoking VMI engine of SECLOUD, it detected that `hexdef100.exe` was running as a child process of `services.exe`, whose parent process was `winlogon.exe`. The IDS of tenant `A` also detected a hidden port of `winlogon.exe`. For the Ubuntu VM of tenant `B`, we installed a kernel rootkit `adore-ng` inside of it. The rootkit hid from modules list and could not be found in the VM. When the remote attestation service checks the kernel integrity, it could find the hidden module by performing a cross-view comparison between current measurement and previous trusted results.

Our framework is designed for tenants to monitor their VMs for security services. They are free to use self-build service or authorize to a managed security service. To this end, our implemented framework should support legacy services. To verify generality of our implementation, we implemented `SeScan`, a security service which was built over the popular open source anti-virus project ClamAV (https://www.clamav.net/). It can be used to scan the virus in the guest VM by the remote tenant. The conventional ClamAV tool executed inside the guest VM to find out malicious files. `SeScan` fulfilled this task by executing the `clamscan` tool in the monitoring VM based on file introspection of SECLOUD.

To confirm that the security service based on our implementation could work effectively, we tested it with the EICAR test files (https://www.eicar.org/). We downloaded three test files (i.e., *eicar.com*, *eicar.com.zip*, *eicar.com.txt*) into the `Downloads` directory and ran ClamAV to scan it inside the VM. Subsequently, we scanned the same directory with `SeScan` in the management VM. As a result, both in-VM scanning and `SeScan` could find the infected files and report to the tenant. Our evaluation showed that our framework is compatible with existing security services without modifying the guest VMs.

5.1.2. Enhanced Functionality Evaluation

Existing VMI-based security solutions for different VMs are isolated. They do not share acquired state of a VM with each other which makes them ineffective to handle sophisticated attacks. To improve the situation, we build up some advanced functionalities by taking advantages of virtualization techniques in our implementation. In this section, we introduce a security service, termed `ProDetect`, that detects anomalous processes based on our enhanced functionalities. By evaluating it with an example scenario, we illustrate how our enhanced functionalities improve the effectiveness of security services.

`ProDetect` takes advantage of cross-VM view of SECLOUD to find out the anomalous processes across a set of guest VMs. In its implementation, it collects information of all the running processes of each guest, including in-memory executable sections. Then it compares the hash value of executable pages of processes with same basic information (e.g., process name) from different guest VMs. Tow processes are considered identical if their hashes match.

In our evaluation, we set up six VMs and ran an example process binary named *h*ello world in each of the first five VMs. We altered the code of hello world and ran it in the sixth VM. When the tenant of these VMs requested to inspect his VMs with `ProDetect` service, the altered process was found and reported as an anomalous process. Table 1 contrasts `ProDetect` with conventional VMI-based security service to perform the equivalent task. The latter requires three monitoring entities to obtain process information of different VMs. In addition, they can only get one item with a request each time. It is obvious that `ProDetect` should be superior because it reduces network traffic and avoids big round trip between the tenant and the cloud.

Table 1. Comparison of `ProDetect` with conventional VMI-based security service.

	Monitoring Entities	Communication RT **	Network Traffic
Conventional VMI *	6	12	processes related info, executable sections
`ProDetect`	1	1	anomalous process info

* Statistics of conventional VMI-based security service to perform the equivalent task of ProDetect. ** RT: round trip.

5.2. Performance Evaluation

To provide tenant-oriented security monitoring with privacy preservation, our implementation involved additional network communication and cryptographic operations, which would incur performance overhead inevitably. We have assessed the performance overhead of our implemented SECLOUD and compared with existing VMI-based systems. To show the practicability of our implementation for supporting individual security service, we compared the performance of `SeScan` with ClamAV running inside the VM. Finally, we measured the overhead imposed on the VMs being monitored by our monitoring framework.

5.2.1. Performance of Security Monitoring

To measure the performance of security monitoring in our implementation and make a comparison with previous work, we focused on the common introspection tool, `process list`, which has been done in the previous work. We used the `gettimeofday` function to measure the time required to traverse the linked list of active processes. Our measurements comprise four functionally identical introspection tools to investigate the difference of performance between different systems. The selected tools include a native LibVMI-based tool, a CloudVMI-based tool running in a local VM, a CloudVMI-based tool running in a remote system, and the tool of SECLOUD.

First, we estimated the time of different introspection tools took for monitoring the VMs with different system configuration. VM1 and VM2 ran Windows7 inside while VM3 and VM4 ran Ubuntu14. The number of processes increased from VM1 to VM4 due to the diversity of OS types and workload. As shown in Figure 4, the amount of time it took for extracting active processes increases with the number of processes inside the VM. The CloudVMI-based tool running in the remote system took more time than other tools for introspecting each tested VM except for the VM1. SECLOUD took more time than CloudVMI for introspection since there were only 27 active processes inside the VM1. Communication and cryptographic operations accounted for the significant time consumption in this scenario. As the number of processes in the monitored VM increased, SECLOUD outperformed the remote CloudVMI-based introspection tool. When the processes up to 170 in the VM4, SECLOUD could be 1.3× faster than CloudVMI. The result indicated that our implemented framework is more capable of monitoring a complex system than CloudVMI. As the security monitoring framework working through the Internet, both CloudVMI and SECLOUD are highly sensitive to network delay. To test the impact of network delay on these frameworks, we emulated a WAN by inserting network delay with the `tc` command at the remote host. The lines at the top of Figure 4 showed the variation of monitoring latency in CloudVMI and SECLOUD with the increasing of network delay. The slope of the linear fitting showed that SECLOUD was less affected by network delay than CloudVMI (only 1/3). The results imply that our implemented framework would perform better than CloudVMI in practice.

In addition to network communication, our framework implemented additional cryptographic operations to protect the privacy of the tenants. The whole process of monitoring guest systems in SECLOUD comprises three phases: (i) runtime information capture via VMI; (ii) cryptographic operations on obtained data; (iii) transmission of data on the network. We measured the time consumed in each phase with a series of security monitoring tools in our implementation. As depicted in Figure 5, the time for cryptographic operations was negligible in each monitoring tools. Transmission of data took up the majority of monitoring time in most tools, except the *netpacket* and *syscall*. The time for VMI-based data capture in these two tools depends on the system activities.

The results all above confirm the intuition that SECLOUD should be slower than native VMI tools. While compared with CloudVMI, SECLOUD has a noticeable performance improvement for security monitoring, especially in the situation of handling complex system under poor network conditions.

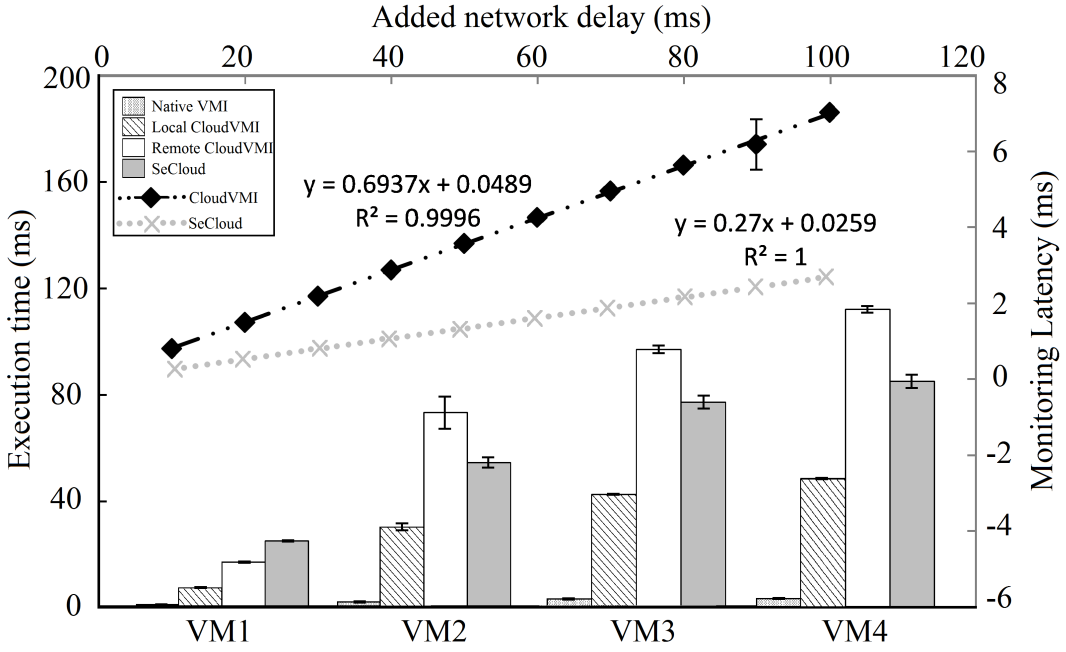

Figure 4. The introspection time of different tool and the impact of network delay.

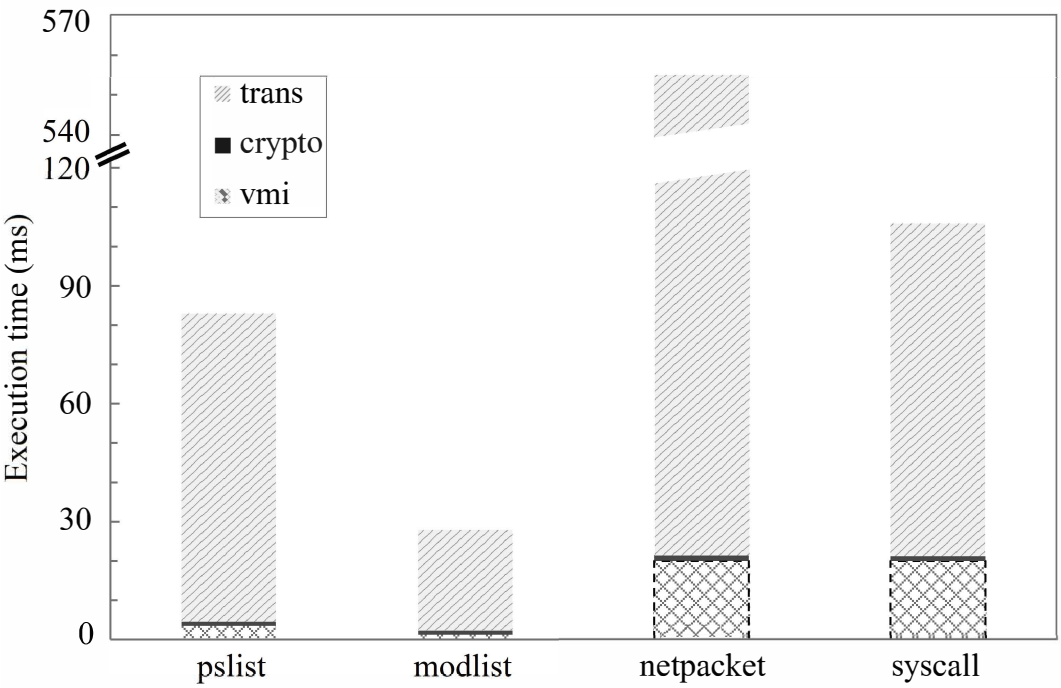

Figure 5. Time consumed in different phases of security monitoring tools in SECLOUD. The part surrounded by the dotted line indicates that the time of this part is uncertain and depends on the system activities.

5.2.2. Performance of Security Service

To quantify the efficiency improvements of security service achievable with SECLOUD, we evaluated the virus scanning service based on different implementation. We used the `SeScan` as the out-of-VM scanner in our framework. We installed ClamAV in the test VM as the in-VM scanner. We configured the VM with one vCPU and 1GB virtual memory. Table 2 shows the summary of the two security services for scanning the `/home` directory in the test VM. We can find that the `SeScan` examined more data which could be hidden to the in-guest ClamAV while spent less time (89.2% of ClamAV). Both of the security services found all the infected files we set for testing.

Table 2. Summary of different security service for virus scanning.

	Scanning Time (s)	Data Scanned (MB)	Infected Files
ClamAV in-VM	97.283	122.73	3
`SeScan`	86.777	122.78	3

During this evaluation, we tracked the usage of CPU and memory resources when the security service scanned the test VM. Figure 6 shows that the trend of CPU use of the two scanners is almost the same. The variation in measurements is due to the inevitable impact of other services in the system. However, the CPU use of `SeScan` was higher than ClamAV at start time, which could be the reason `SeScan` was faster than ClamAV. As seen, the memory use of both scanner kept stabilization relatively during the evaluation. Though the metrics shown in the figure are distinct, the physical memory used by the two scanners were equal. The difference derived from the different configurations of the test VM and the management VM. It can be inferred that the security services in our framework would not incur extra resource overhead.

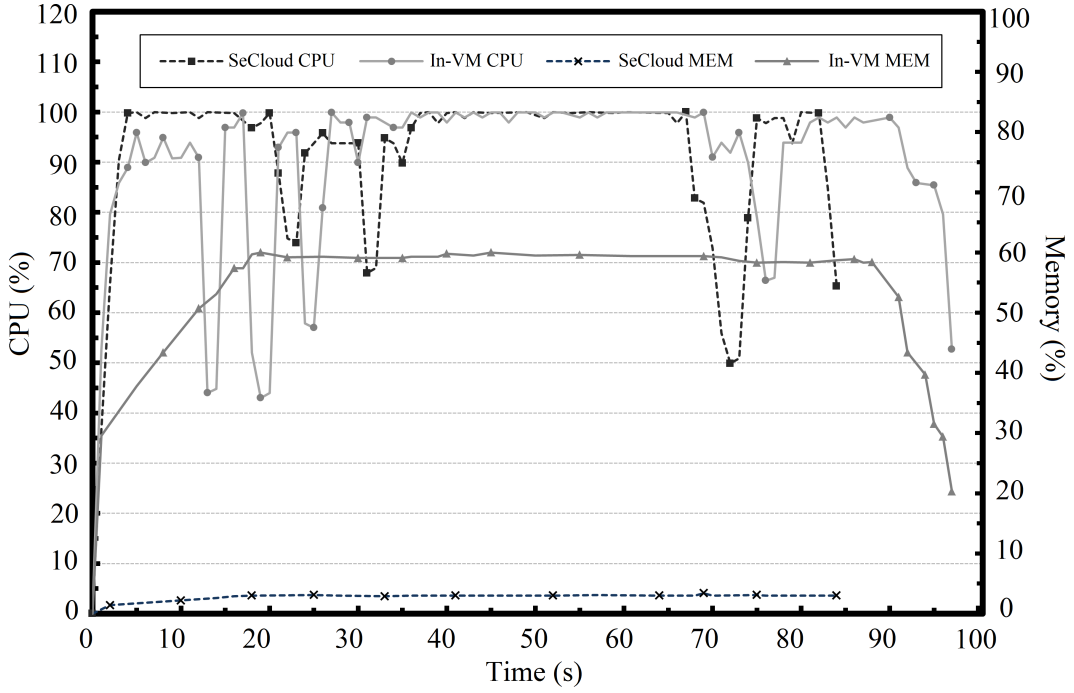

Figure 6. The CPU and memory usage of different security services during scanning the guest VM.

5.2.3. Performance Overhead on Target VMs

We measured the performance impact of SECLOUD on a target system's workload with two experiments. First, we used the sysbench (https://github.com/akopytov/sysbench) micro-benchmark to evaluate the primitive-level performance reduction. The sysbench is a scriptable multi-thread benchmark tool based on LuaJIT. It can be used to create arbitrarily complex workloads for testing. We used three of the bundled benchmarks in sysbench, i.e., `cpu`, `memory`, `fileio`, to measure the performance overhead imposed on different operations in the target VM. The workload configuration for the tested VM in our experiment is shown in Table 3.

Table 3. Workload configuration for the tested VM.

Benchmarks	Number of Threads *	Execution Time	Number of Files	Memory Block	Total Size	Test Model
`cpu`	1	10 s	-	-	-	-
`memory`	1	-	-	1 K	1 G	write
`fileio`	1	-	128	-	2 G	random rw

* We used one thread in our experiment since the tested VM was configured only one vCPU. - indicates an invalid configuration item for the benchmark.

As illustrated in Figure 7, we measured the relative performance of each benchmark with different monitoring intervals, which refers to the baseline test. The baseline shown in the first group of columns was measured in the tested VM without being introspected by SECLOUD. From Figure 7, we can find that when SECLOUD started to capture runtime information, it induced a performance degradation on the tested benchmarks. The performance of the cpu benchmark is more susceptible to monitoring tools in SECLOUD. In the worst situation, it is down to 89.5% when at a frequency of 0.2 s. The performance of memory benchmark is little affected by monitoring tools. It can achieve 98% of baseline at a rate of 1 s.

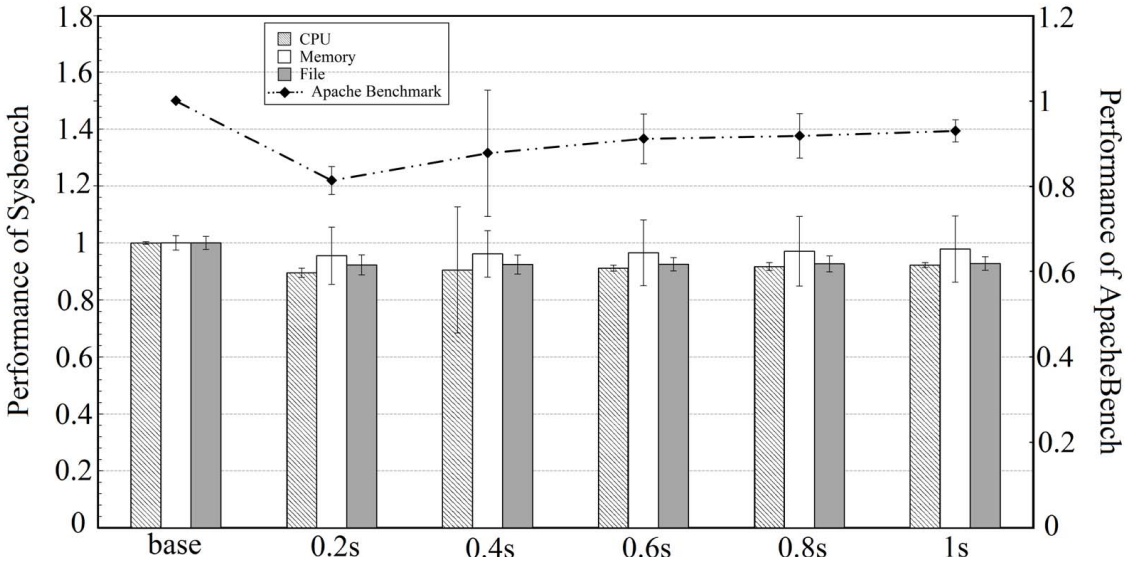

Figure 7. Performance impact imposed by SECLOUD on different benchmarks with different monitoring frequencies.

Overall, we can conclude that the lower the monitoring frequency is, the less the performance reduction is. When we set the monitoring interval as 1 s, the degradation for all of the benchmarks is less than 8%. Especially for the memory benchmark, the performance reduction imposed by SECLOUD is less than 5% in all cases.

To further analyze the holistic performance effect on the monitored VM, we used a real-world workload to quantify the performance reduction at the macro level. We selected the ApacheBench (https://httpd.apache.org/docs/2.4/programs/ab.html) (ab) as our macro-benchmark. It is a load

testing and benchmarking tool for the HTTP server. We measured the performance of ApacheBench at different monitoring frequencies and made 10,000 requests with concurrency 100 at each evaluation iteration. As shown with the line in Figure 7, the performance of ApacheBench down to 81% of baseline when we monitored the tested VM at a frequency of 0.2 s. It increases with the reduction of monitoring frequency and approach baseline slowly. We believe that the performance effect will be negligible if the monitoring frequency is low enough.

In general, the performance overhead imposed on both the micro- and macro-benchmarks is little and will be negligible by adjusting the monitoring frequency. However, it is worth noting that the result of the benchmark had a larger margin of error than the one of the baseline when we monitored the guest VM outside with SECLOUD. It implies that our framework could cause a little instability of the result of benchmarks. However, for long-running processes, it is a little effect.

6. Related Works

In order to make use of virtual machine introspection for securing guest in the cloud effectively, researchers have investigated various usage model.

Baek et al. [34] also aims at making privileged VMI as-a-service to cloud users and enables cloud-centric introspection by allowing VMI actions to be performed across different physical machines via remote procedure calls (RPCs). CloudVMI is flexible and compatible with existing VMI-based tools because they only need to recompile the tools against the vlibVMI client library. However, the RPC server should maintain multiple VMI instance for different VMI tools, which is inefficient. Furthermore, additional latency introduced by the use of RPC for each VMI call makes matters worse.

On the issue of access control of using VMI in the cloud environment, CloudPhylactor [30] build a dedicated monitoring VM for each tenant and harness mandatory access control with Flux Advanced Security Kernel (Flask) architecture of the Xen hypervisor. This method does not introduce a significant overhead because it uses the same APIs as native tools. However, it is impractical for resource consumption in the cloud environment. A tenant must provide and maintain a dedicated VM for each monitored guest VM.

In order to assist tenants in troubleshooting the cloud problems, *CloudSight* [35] dynamically monitors state changes of resources in different points, and associates information from various data sources. It allows tenants to have greater visibility and maintain the state change history in a graph database. Tow tenant-oriented applications demonstrated the efficacy of *CloudSight*. However it is an invasive monitoring tool which needs to inject monitoring code into the target system. It is also a coarse-grained tool without investigating the guest system in detail.

FROST [36] is a set of user-driven forensic tools for the OpenStack cloud platform. FROST provides trustworthy forensic acquisition of virtual disks, API logs, and guest firewall logs from the management plane of OpenStack. However, FROST is not pervasive for it is integrated with OpenStack. Its functionalities are limited relatively, which are restricted to logs and virtual disks. Although they declared more usage of FROST in [36], e.g., real-time monitoring, it is impracticable regarding its coarse generality and postmortem analysis model.

We list several desirable features of cloud monitor framework and compare previous work with SECLOUD against these features. As shown in Table 4, our proposed framework has advantages over listed related works in most aspects. We have a slight deficiency in performance for remote monitoring. Nevertheless, it is an essential trade-off between performance and remote availability which is valuable.

Table 4. Comparison of previous work on cloud monitor framework based on VMI with our work.

Features	CloudVMI [34]	CloudPhylactor [30]	CloudSight [35]	FROST [36]	SECLOUD
Flexibility	●	●	◐	◐	●
Privacy Preservation	○	●	○	○	●
Performance	○	●	◐	◐	◐
Effectiveness	◐	◐	●	◐	●
Generality	●	◐	○	○	●
Extensibility	●	●	◐	◐	●
Usability	●	○	◐	●	◐

●: better feature ◐: medium feature ○: weak feature.

7. Conclusions

In this paper, we exploit the defects and limitations of existing solutions for securing guest VMs in the cloud. To address these problems, a tenant-oriented monitoring framework is presented. By extending VMI techniques, it enables tenants to obtain a genuine and comprehensive view of their remote system stealthily, including memory, filesystem, and network activities. They can take advantage of this ability to deploy individualized security services by themselves or collaborate with experienced managed security service providers. Compared to other VMI-based security solutions, our framework decouples security analysis from cloud monitoring and involves tenants in securing their VMs, which improves the flexibility and trustworthiness of security services. Our framework also outperforms conventional in-guest method when coping with sophisticated attacks. We improve system monitoring with better isolation and enhanced function, which attributed to the adoption of VMI techniques. In addition to security improvement, we enforce the confidentiality of obtained data during monitoring in our framework. Thus we could alleviate the privacy concern of tenants about the risk of sensitive data leak during privileged introspection.

With our effort to implement and evaluate the prototype framework, we believe that our work throws light upon a feasible solution for cloud monitoring to support flexible security service. Our current prototype implementation relies on the assumption that the invariability of the layout of kernel structure would be held during introspection. However, existing research [15,37] has shown the possibility of a violation of it. To counter such attacks, our future work will focus on harnessing robust invariants for bridging semantic gap.

Author Contributions: H.Z. and H.B. contributed equally to the design of the ideas, the implementation of the prototype and the writing of the paper. Y.L. contributed to the analysis of results and proofread the paper. Y.W., Z.W., J.M., Y.L. and H.Q. proofread the paper.

Funding: This research was funded by the National Natural Science Foundation of China under Grant No. 61402508, No. 61303191 and No. 61472439, and the National High Technology Research and Development Program of China (863) under Grant No. 2015AA016010.

Acknowledgments: We would like to thank the anonymous reviewers for their insightful comments and suggestions on improving this paper. We are also thankful to our colleagues for their support, suggestions and reviews while conducting this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

OS	Operating System
VM	Virtual Machine
VMI	Virtual Machine Introspection
TPM	Trusted Platform Module
XSM	Xen Security Modules
VMM	Virtual Machine Monitor
MSSP	Managed Security Service Provider
DDoS	Distributed Denial of Service
IDS	Intrusion Detection System
TTP	Trusted Third Party

References

1. Singh, A.; Chatterjee, K. Cloud security issues and challenges: A survey. *J. Netw. Comput. Appl.* **2017**, *79*, 88–115. [CrossRef]
2. Baliga, A.; Kamat, P.; Iftode, L. Lurking in the shadows: Identifying systemic threats to kernel data. In Proceedings of the IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA, 20–23 May 2007; pp. 246–251.
3. Garfinkel, T.; Rosenblum, M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In Proceedings of the Conference on Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 18–21 February 2003; pp. 191–206.
4. Bauman, E.; Ayoade, G.; Lin, Z. A Survey on Hypervisor-Based Monitoring: Approaches, Applications, and Evolutions. *ACM Comput. Surv.* **2015**, *48*, 1–33. [CrossRef]
5. Hizver, J.; Chiueh, T.C. Real-time Deep Virtual Machine Introspection and Its Applications. In Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '14), Salt Lake City, UT, USA, 1–4 March; ACM: New York, NY, USA, 2014; pp. 3–14. [CrossRef]
6. Jiang, X.; Wang, X.; Xu, D. Stealthy Malware Detection Through Vmm-based "Out-of-the-box" Semantic View Reconstruction. In Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS), Alexandria, VA, USA, 28–31 October 2007; ACM: New York, NY, USA, 2007; pp. 128–138.
7. Lengyel, T.K.; Maresca, S.; Payne, B.D.; Webster, G.D.; Vogl, S.; Kiayias, A. Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System. In Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC), New Orleans, LA, USA, 8–12 December 2014; ACM: New York, NY, USA, 2014; pp. 386–395.
8. Laurén, S.; Leppänen, V. Virtual Machine Introspection based Cloud Monitoring Platform. In Proceedings of the 19th International Conference on Computer Systems and Technologies, Varanasi, India, 4–17 January 2018; ACM: New York, NY, USA, 2018; pp. 104–109.
9. Butt, S.; Lagar-Cavilla, H.A.; Srivastava, A.; Ganapathy, V. Self-service cloud computing. In Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS), Raleigh, NC, USA, 16–18 October 2012; ACM: New York, NY, USA, 2012; pp. 253–264.
10. Zhang, F.; Chen, J.; Chen, H.; Zang, B. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, Cascais, Portugal, 23–26 October 2011; ACM: New York, NY, USA, 2011; pp. 203–216.
11. Li, C.; Raghunathan, A.; Jha, N.K. Secure virtual machine execution under an untrusted management OS. In Proceedings of the IEEE 3rd International Conference on Cloud Computing (CLOUD), Miami, FL, USA, 5–10 July 2010; pp. 172–179.
12. Fu, Y.; Lin, Z. Bridging the semantic gap in virtual machine introspection via online kernel data redirection. *ACM Trans. Inf. Syst. Secur.* **2013**, *16*, 1–29. [CrossRef]
13. Payne, B.D.; Martim, D.d.A.; Lee, W. Secure and flexible monitoring of virtual machines. In Proceedings of the Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Miami, FL, USA, 11–14 December 2007; pp. 385–397.

14. Gu, Z.; Deng, Z.; Xu, D.; Jiang, X. Process implanting: A new active introspection framework for virtualization. In Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems (SRDS), Madrid, Spain, 4–7 October 2011; pp. 147–156.
15. Jain, B.; Baig, M.B.; Zhang, D.; Porter, D.E.; Sion, R. Sok: Introspections on trust and the semantic gap. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 17–18 May 2014; pp. 605–620.
16. Harrison, C.; Cook, D.; McGraw, R.; Hamilton, J. Constructing a cloud-based IDS by merging VMI with FMA. In Proceedings of the IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Liverpool, UK, 25–27 June 2012; pp. 163–169.
17. Azab, A.M.; Ning, P.; Sezer, E.C.; Zhang, X. HIMA: A hypervisor-based integrity measurement agent. In Proceedings of the Computer Security Applications Conference (ACSAC'09), Austin, TX, USA, 6–10 December 2009; pp. 461–470.
18. Srivastava, A.; Giffin, J. Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections. In Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID), Cambridge, MA, USA, 15–17 September 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 39–58.
19. Ahmed, I.; Richard, G.G.; Zoranic, A.; Roussev, V. Integrity Checking of Function Pointers in Kernel Pools via Virtual Machine Introspection. In Proceedings of the 16th International Conference on Information Security (ISC), Dallas, TX, USA, 13–15 November 2013; Springer: Berlin/Heidelberg, Germany, 2015; pp. 3–19.
20. Johnson, D.; Hibler, M.; Eide, E. Composable multi-level debugging with Stackdb. In *ACM SIGPLAN Notices*; ACM: New York, NY, USA, 2014; Volume 49, pp. 213–226.
21. Suneja, S.; Isci, C.; Bala, V.; De Lara, E.; Mummert, T. Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Austin, TX, USA, 16–20 June 2014; ACM: New York, NY, USA, 2014; pp. 249–261.
22. Zhang, S.; Meng, X.; Wang, L.; Xu, L.; Han, X. Secure Virtualization Environment Based on Advanced Memory Introspection. *Secur. Commun. Netw.* **2018**, *2018*, 1–16. [CrossRef]
23. Cui, L.; Song, Z.; Li, Y.; Hao, Z. XScope: Memory Introspection Based Malicious Application Detection. In Proceedings of the 5th International Conference on Information Science and Control Engineering (ICISCE), Zhengzhou, China, 20–22 July 2018; pp. 1296–1300.
24. Proskurin, S.; Lengyel, T.; Momeu, M.; Eckert, C.; Zarras, A. Hiding in the Shadows: Empowering ARM for Stealthy Virtual Machine Introspection. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; ACM: New York, NY, USA, 2018; pp. 407–417.
25. Stratsec IT Security Winter School. botCloud—An Emerging Platform for Cyber-Attacks. Available online: https://0xicf.wordpress.com/2012/11/02/botcloud-an-emerging-platform-for-cyber-attacks// (accessed on 1 September 2018).
26. Bhattasali, T.; Chaki, N. Poster: Exploring Security As a Service for IoT Enabled Remote Application Framework. In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion (MobiSys Companion), Singapore, 25–30 June 2016; ACM: New York, NY, USA, 2016; p. 15.
27. Hurel, G.; Badonnel, R.; Lahmadi, A.; Festor, O. Outsourcing Mobile security in the cloud. In Proceedings of the IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS), Munich, Germany, 20–23 June 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 69–73.
28. Daniel, J.; Dimitrakos, T.; El-Moussa, F.; Ducatel, G.; Pawar, P.; Sajjad, A. Seamless enablement of intelligent protection for enterprise cloud applications through service store. In Proceedings of the IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom), Singapore, 15–18 December 2014; pp. 1021–1026.
29. Wang, J.; Stavrou, A.; Ghosh, A. Hypercheck: A hardware-assisted integrity monitor. In *Recent Advances in Intrusion Detection*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 158–177.
30. Taubmann, B.; Rakotondravony, N.; Reiser, H.P. Cloudphylactor: Harnessing mandatory access control for virtual machine introspection in cloud data centers. In Proceedings of the Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016; pp. 957–964.
31. Taubmann, B.; Frädrich, C.; Dusold, D.; Reiser, H.P. TLSkex: Harnessing virtual machine introspection for decrypting TLS communication. *Digit. Investig.* **2016**, *16*, S114–S123. [CrossRef]

32. Kashyap, A.; Kumar, G.S.; Jangir, S.; Pilli, E.S.; Mishra, P. IHIDS: Introspection-based hybrid intrusion detection system in cloud environment. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017; pp. 687–693.
33. Payne, B.D. *Simplifying Virtual Machine Introspection Using Libvmi*; Technical Report SAND2012-7818; Sandia National Laboratories: Albuquerque, NW, USA; Livermore, CA, USA, 2012.
34. Baek, H.; Srivastava, A.; Van der Merwe, J. CloudVMI: Virtual Machine Introspection As a Cloud Service. In Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E), Boston, MA, USA, 11–14 March 2014; pp. 153–158.
35. Baek, H.; Srivastava, A.; Van der Merwe, J. CloudSight: A tenant-oriented transparency framework for cross-layer cloud troubleshooting. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Madrid, Spain, 14–17 May 2017; pp. 268–273.
36. Dykstra, J.; Sherman, A.T. Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. *Digit. Investig.* **2013**, *10*, S87–S95. [CrossRef]
37. Bahram, S.; Jiang, X.; Wang, Z.; Grace, M.; Li, J.; Srinivasan, D.; Rhee, J.; Xu, D. Dksm: Subverting virtual machine introspection for fun and profit. In Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems, New Delhi, India, 31 October–3 November 2010; pp. 82–91.