CloudIDEA: A Malware Defense Architecture for Cloud Data Centers

Andreas Fischer[3], Thomas Kittel[1], Bojan Kolosnjaji[1], Tamas K Lengyel[1], Waseem Mandarawi[3], Hermann de Meer[3], Tilo Müller[2], Mykola Protsenko[2], Hans P. Reiser[3], Benjamin Taubmann[3], and Eva Weishäupl[4]

[1] Technische Universität München
`{kittel, kolosnjaji, tklengyel}@sec.in.tum.de`
[2] University of Erlangen-Nürnberg
`tilo.mueller@cs.fau.de, mykola.protsenko@fau.de`
[3] University of Passau
`{`*`firstname.lastname`*`}@uni-passau.de`
[4] University of Regensburg
`eva.weishaeupl@wiwi.uni-regensburg.de`

Abstract. Due to the proliferation of cloud computing, cloud-based systems are becoming an increasingly attractive target for malware. In an Infrastructure-as-a-Service (IaaS) cloud, malware located in a customer's virtual machine (VM) affects not only this customer, but may also attack the cloud infrastructure and other co-hosted customers directly. This paper presents CloudIDEA, an architecture that provides a security service for malware defense in cloud environments. It combines lightweight intrusion monitoring with on-demand isolation, evidence collection, and in-depth analysis of VMs on dedicated analysis hosts. A dynamic decision engine makes on-demand decisions on how to handle suspicious events considering cost-efficiency and quality-of-service constraints.

1 Introduction

Cloud computing has become a dominant computing paradigm over the past years. The flexibility and scalability offered by cloud providers lead to more and more services to be outsourced to the cloud. In this paper, we address security challenges that arise in an Infrastructure-as-a-Service (IaaS) cloud environment.

Detecting, analyzing and preserving evidence about attacks against customers' virtual machines (VMs) in the cloud is more complex than handling similar attacks on a local infrastructure controlled by the customer: the customer on the one hand has detailed knowledge about his own VMs, but lacks direct access to the cloud infrastructure. On the other hand, the cloud provider has full control over the cloud infrastructure and is potentially in a good position for analyzing abnormal activities, but lacks the contextual knowledge about the hosted VMs.

This difficulty limits the applicability of intrusion detection systems (IDSs) by the cloud operator. Using a heuristics-based IDS incurs the danger of false positives, i.e., classifying normal behavior of a VM as an attack. If an IDS suspicious activities alarm is used to automatically terminate the affected VM, false alarms will inhibit a legitimate service. If the only reaction to the alarm is a notification of the cloud customer, a delayed manual reaction to the alarm puts the cloud provider and other customers' VMs at risk.

The accuracy of attack detection, the comprehensiveness of malware analysis and the conclusiveness of evidence depend on the selection of information sources and monitoring mechanisms. Typically, more detailed information gathering incurs higher runtime cost. A non-trivial tradeoff to be made is between performance impact on a production system, analysis costs and quality of the collected data. Collecting data within a VM is susceptible to manipulation by the attacker. Acquiring conclusive evidence requires data collection at a place where an adversary is not able to alter it.

There are many existing approaches that address partial aspects of the outlined problem. To our best knowledge, however, there is no solution that combines stealthy intrusion detection, comprehensive evidence collection and in-depth automated malware analysis in a joint architecture for cloud-based environments. In this paper, we address the following core research questions:

- How can both cloud providers and customers take advantage of state-of-the-art malware defense mechanisms based on virtual machine introspection?
- What introspection-based techniques are sufficiently lightweight in order to be used continuously on a production system?
- What introspection-based techniques – possibly heavy-weight – can be activated on demand to yield additional information and a detailed understanding of malicious attacks and malware behavior?

In this paper, we present CloudIDEA, a system that monitors the activity of VMs using lightweight detection mechanisms with low overhead on the production environment. Upon observing abnormal symptoms, these mechanisms will trigger events that a decision engine (DE) will combine with quality of service (QoS) parameters and the current virtual resources allocation to take appropriate actions such as isolating the VM to a dedicated analysis environment.

2 Threat model

As illustrated by Figure 1, there are three basic attack targets in a cloud environment: the cloud management system, the hypervisor and node management, and the VMs executed in the cloud.

The *cloud management system*, as any other software, potentially contains exploitable security flaws [39]. Avoiding or detecting such attacks is currently not in the main focus of our work. The *hypervisor/node management* subsystem manages VMs and is controlled by the cloud management system. We assume that the management interface is not accessible by external attackers, as it is typically hidden within a dedicated management VLAN, and thus do not consider direct external attacks against this subsystem. If the attacker controls a VM, the hypercall interface of the VM is a potential attack vector [31].

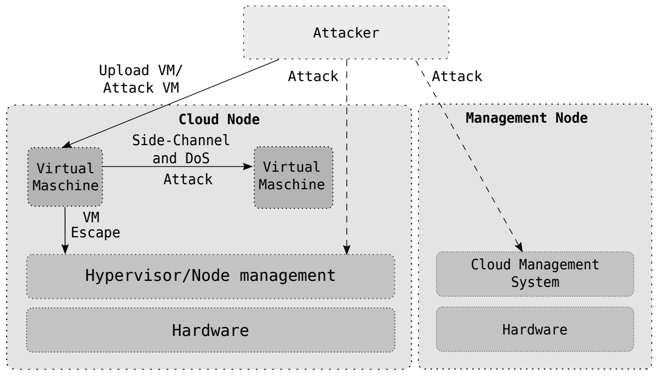

Fig. 1. IaaS cloud threat model; arrows with solid lines are attacks considered in this paper

The customer VMs can be both a target and a potential source of attacks. As a target, an attacker can directly exploit vulnerabilities of software running within the virtual machine. Such attacks cause immediate harm for the affected cloud customer, and a core focus of our work is supporting the cloud customer in the defense against such attacks. Furthermore, an adversary can simply upload any malicious VM in a public cloud. In both cases, a malicious VM can perform two attacks against other VMs: denial of service (DoS) attacks and cross-VM side channel attacks. DoS attacks can be performed by overloading physical resources of a cloud node or through the virtual network [38]. Cross-VM side channel attacks can be performed by examining the behavior of virtual or physical hardware modules, such as the L2 cache [53], and deducing information about the state of other VMs. Additionally, an infected VM might try to escape the hypervisor using different mechanisms such as privilege escalation or memory brute force attacks. In this case, the attacker gains control of other VMs, the hypervisor, or the cloud node completely [40].

We assume that, in the future, the biggest threats to IaaS based clouds are network based attacks against VMs, internal cross-VM side channel attacks and VM escapes. Thus, the main focus of CloudIDEA is to detect those attacks by monitoring and tracing VMs.

3 Architecture

The CloudIDEA architecture enhances the security of IaaS cloud data centers by improving the capability to detect, analyze in depth, and preserve evidence about the attacks described in Section 2. The architecture aims to satisfy the following design goals:

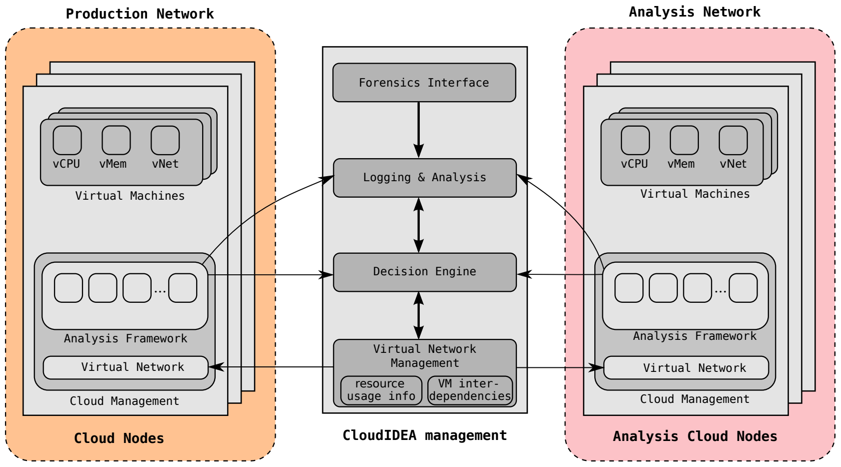

Fig. 2. The CloudIDEA architecture

– *Security-as-a-Service*: Enable a cloud customer to use advanced detection, analysis and evidence preservation techniques, such as transparent virtual machine introspection (VMI).
– *Security for the provider*: Support the cloud provider in dealing with malicious behavior originating from virtual machines, targeting the cloud infrastructure, other customers, or external targets.
– *Running in production systems*: Be well-suited to be used in a production system. This means that we need to support mechanisms that are sufficiently lightweight to cause only negligible overhead. Also, the system needs to be tailored to the specific needs of both cloud provider and cloud customer.
– *In-depth analysis and evidence preservation*: Provide detailed information and conclusive evidence about attacks.

3.1 Overview

The CloudIDEA architecture is shown in Fig. 2. This architecture contains a decentralized, modular, and scalable analysis framework, which supports both lightweight monitoring and heavyweight in-depth analysis, controlled by a central CloudIDEA management component. For isolating potentially infected VMs and for analysing them with resource-intensive mechanisms, the suspicious VMs can be migrated to a dedicated analysis environment.

Every VM is monitored by a decentralized *analysis framework*, which is part of every physical cloud node. It includes plug-ins for tracing and introspection of VMs, e.g. , network traffic monitoring and hypercall tracing. Those plug-ins can be activated and deactivated at runtime on demand, in order to minimize the overhead in a production environment and tailor the monitoring mechanisms to the observed current threat level.

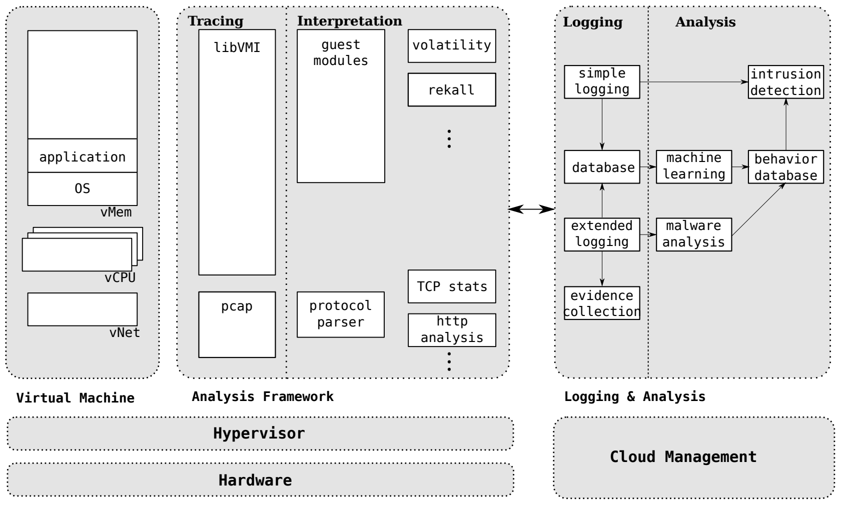

Fig. 3. Analysis Framework and Cloud Management

The traces are processed in a central *CloudIDEA management* component. It incorporates all components that are responsible for attack detection, mitigation and evidence collection in a central place in order to learn and compare common patterns for all VMs in a cloud data center. This information is used, for example, to detect distributed attacks against several targets. If suspicious behaviour is observed, further processing, such as deeper analysis or restarting, is computed by the DE. The components of the *CloudIDEA management* are: *the forensics interface, logging & analysis, DE and the virtual network management.*

The *logging & analysis* component is responsible for storing the traces of VMs to provide that information for intrusion detection and evidence collection (see Figure 3). The *behavior database* contains a model for each VM which is deduced with machine learning algorithms from the traces of the analysis framework. These models serve as a basis for *intrusion detection* to recognize deviant behavior. The *forensics interface* provides customers access to this data and enables customers to execute forensics analysis on their VMs. Whenever an anomaly is detected, the *DE* is informed and determines how to react, e.g., by isolating or restarting the corresponding VM. The decision is influenced by the customer (e.g., predefined configurations and service level agreements (SLAs)) and cloud internal information (e.g., available resources and VM interdependencies). If a VM shows a suspicious behavior, which needs to be analyzed more intensely, it is migrated to an analysis host. Therefore, the *virtual network management* reconfigures the virtual network infrastructure and ensures that resources are still available and SLA compliances can still be guaranteed.

3.2 Intrusion detection and analysis

Monitoring and tracing the state of VMs is an important feature of the analysis framework in order to detect possible intrusions. In this section we will discuss the applied techniques.

VM introspection for intrusion detection and prevention Over the last decade, there has been a significant push to move the security stack from VMs into regions protected by the hypervisor [23,30]. With the proliferation of kernel-mode rootkits, this elevated protection has become ever more important [3,13]. While running external to the context of the VM being protected, such security software inherently had to tackle the *semantic gap problem*: reconstructing high-level state information from low-level data-sources [14]. This problem has been a main motivation for a significant portion of research over the last decade. However, with recent advances in forensics tools, the semantic gap problem can be considered a solved engineering problem [21].

In recent years, open-source forensics memory analysis frameworks such as Volatility[5] and Rekall[6] have achieved an unparalleled view into the execution of modern operating systems, such as Windows, Linux and MacOS X. Furthermore, as the underlying use-case for these frameworks has been extracting evidence of compromise, the inclusion of various methods to side-step obfuscation and rootkit techniques has significantly raised the bar for malware to achieve effective stealth [9]. Further combined with active monitoring systems, such as utilizing the CPUs two-stage paging mechanism (Intel EPT) to monitor memory accesses [47] or via stealthy breakpoint-injection [7], external security software are now capable of not just extracting state-information of running VMs but also interposing on the execution to deliver additional protection mechanisms [8,26].

Ongoing research thus has not been focusing on how to gain access to the required information but rather how to utilize it to deliver additional security services: intrusion detection and intrusion prevention [15]. Furthermore, it is essential for such systems to exert limited overhead on production systems. Research has shown that system call interception [19] and heap-tracing [34] are a viable approach to detect intrusions; however, the exerted overhead may be prohibitive on production systems. While upcoming next-generation virtualization extensions such as Intel #VE may be capable of mitigating some of that overhead, there is a clear need for lightweight detection mechanisms as well [11].

Lightweight malware detection Compared to classical antivirus (AV) engines, VMI based detection systems have the advantage of being able to monitor and alter the execution of the operating systems they protect from an elevated CPU mode [5]. While static signature based malware detection mechanisms can be ported to use VMI for data-access, additional protection mechanisms can also be implemented. For example, buffer-overflow and heap-spray attacks inherently rely on placing code into memory regions normally used only for data: the stack

[5] The volatility framework - https://www.volatilesystems.com/default/volatility

[6] Rekall: Memory forensics analysis framework - http://www.rekall-forensic.com

and the heap [41]. While modern OS's provide counter-measures to these types of attacks, such as data execution protection (DEP) and address space layout randomization (ASLR), they are known to be easily circumvented. However, if DEP is implemented by the hypervisor as well, it can act as a low-overhead indicator of compromise (IoC). Recent extensions to open-source hypervisor technologies, such as Xen[7], now actively support such protection schemes, where the shell-code can further be turned into a NOP-sled via the use of emulators [2].

Rootkit mechanisms also notoriously rely on placing *hooks* into the OS execution [50]. The behavior is well documented in several malware families, and the immediate target of such hooks are core tables, for example, holding the addresses for system call handlers and interrupts [27]. By replacing members of these tables, rootkits can actively interpose on the execution of the OS, thus being able to log and mangle the output of any such event. Stealthier versions place *inline-hooks* [4] directly into the target function entry points, which have the advantage of allowing a wider set of combination of placing the hooks, thus detection can be complicated on systems that allow dynamic run-time kernel patching and updates to be deployed [25]. Other hardware events may also act as low-overhead indicators of compromise, such as the rapid querying of the Time Stamp Counter (TSC) counter. Such behavior usually indicates an in-guest agent attempting to detect the presence of out-of-guest monitors [8]. Furthermore, the rapid querying of the TSC has also been used during side-channel attacks [53]. Given that the `RDTSC instruction` can be configured to be trapped into the hypervisor, an external frequency analysis would easily reveal such malicicous behaviors.

These malware behaviors naturally lend themselves to lightweight detection: under normal operation of the operating system well-behaved application would never exhibit such behaviors. Furthermore, as the overhead added by the protection only applies when a malicious behavior is triggered, it would arguably be negligible.

Heavyweight malware detection While the lightweight detection mechanisms apply to a potentially wide-range of malware, it is by no means exhaustive. A lot of malware operates without exhibiting the behaviors previously discussed. Therefore, our lightweight detection is not a replacement for existing security solutions, such as in-VM antivirus engines and network IDS solutions, rather an augmentation on the existing stack. While ideally lightweight detection would be possible for all malware instances, we need to collect additional artifacts for systems which exhibit *suspicious* but unidentifiable malicious behavior.

Growing number of malware instances and families create significant difficulties for traditional signature-based detection systems to correctly detect and classify malware samples. A recent trend of malware having properties of multiple families further complicates the detection and classification. An attempt to solve this is using statistical machine learning methods. These methods leverage gathered behavioral data about malware to generate statistically confident

[7] Xen - http://www.xenproject.org

knowledge. This has been attempted in the past with various data sources that characterize program behaviour: system calls [45], registry accesses [18], network packets [42]. These event sequences are analyzed using unsupervised (clustering) or supervised learning (classification) methods. The used methods can be further divided into one-class anomaly detection and multiclass learning.

For the heavyweight detection mechanism it is crucial to maintain the sample set from different malware families and their behavioral patterns to be able to properly classify the suspicious applications. Furthermore, in the case of sequential data, automatic methods for extraction of semantically relevant features must be used to cope with the possibly noisy and high-dimensional data. An example of this is given by recent application of topic modeling approaches to the classification of system call sequences [49]. This approach can be extended by including memory allocation patterns and other traceable operations. The standard methods for direct classification of sequential data are Hidden Markov Models (HMM) and Recurrent Neural Networks (RNN). Support vector machines (SVM) with string kernels give another approach, where a standard classification scheme is augmented to work with sequences of variable length [32]. Network traffic produced by the analyzed samples can be classified by taking into account the frequency and length of different type of packets or generating n-gram features out of packet payloads.

Results of the heavyweight detection mechanisms can be structured to form a significant knowledge base. The knowledge acquired after executing the heavyweight detection mechanisms can be fed back to the lightweight detection as behavioral signatures, to further decrease the necessary resource usage. Also, malware samples that do not belong to the previously defined families can be added as prototypes to the clustering or classification schemes. This method has shown success in the previously published work [35].

In case of suspicious activity, our architecture enables extensive data collection using tools for VMI and network traffic analysis. This data is an input for the machine learning-based malware detection mechanism. The machine learning engine contains two main parts: extraction of relevant, preferably semantically interpretable features from raw data and classification based on those features. Different feature extraction methods are applied based on the type of data. Based on these features, a suspicious application is either classified into known families, as benign, or as a malware of unknown family. In the last case, the behavioral signature of this application is used to form a new class. Ensemble learning enables the usage of different classifiers and averaging out of their results. The classification results are fed back to the DE. Feature values and the classification results of the analyzed sample are saved to the behavior database for further reference.

3.3 Decision engine

The task of the DE is to orchestrate the decentralized analysis frameworks. A main mechanism of the architecture is to isolate suspicious VMs by migrating them to a dedicated analysis environment. This environment provides an infrastructure with smaller attack surface and with the required features and resources for the analysis, which may not be available in the production environment. The main challenge of the migration is keeping the service requirements and dependencies between VMs satisfied. The approaches for resource allocation in virtual environments try to optimally allocate VMs according to certain requirements such as to minimize energy consumption [22]. An optimized reconfiguration process is required to determine the minimum set of VM migrations that can reconfigure the virtual network in a way that allows for a detailed analysis and at the same time adhere to the QoS requirements.

Economical aspects The distribution of specialized investigation hosts plays a main role in the decision process. Finding an efficient distribution of these hosts is also an economical challenge of the proposed architecture. This requires a detailed analysis of the target environments and expected attack scenarios. The trade-off between protection costs and attack costs will be a major output of this analysis. Another economic aspect is the minimization of the VM's downtime during the migration to prevent a monetary [48] and a reputational loss [16] for the cloud provider. The service provider commits to a certain level of service, which is described by a SLA. Usually, the SLA expresses the VM availability. For example, 99.999% guaranteed availability means 5 minutes downtime per year. Non-compliance to such SLAs can lead to (monetary) penalties for the providers and can harm their reputation [44]. Reputational damage leads to economical long-term consequences, because the consumer's trust in the service might be lost and fewer customers might use the provider's services in future. Therefore, the downtime and migration time of the VM need to be minimized. Although live migration usually causes only a short downtime to a VM, small interruptions ranging from 60 milliseconds to 3 seconds are inevitable [1]. In order to guarantee a certain QoS it is, therefore, essential to predict the worst case downtime as precisely as possible [36]. Based on the model presented by Salfner et al. [36] and the characteristics of our architecture, the migration time and downtime will be predicted and measured. Voorsluys et al. [44] performed experiments to evaluate the cost of migration of VMs considering service disruptions and violations of SLAs. They concluded that most SLAs can still be met when migrations are performed, so that the reputational harm is within limits.

Inputs The DE depicted in Figure 4 is the central component of the proposed architecture. It has communication interfaces with the cloud management system, detection mechanisms deployed in cloud nodes, a set of databases, and the cloud administrator and customer. The main role of this engine is to react to alarms received from the detection mechanisms when certain behavior patterns are detected. The reaction can be a certain reconfiguration action in the cloud environment, or raising an alarm to the cloud administrator or customer. To perform this task, the DE needs four main inputs.

The first input is the configuration and monitoring data that is acquired from the cloud management system and contains four data sets. The physical

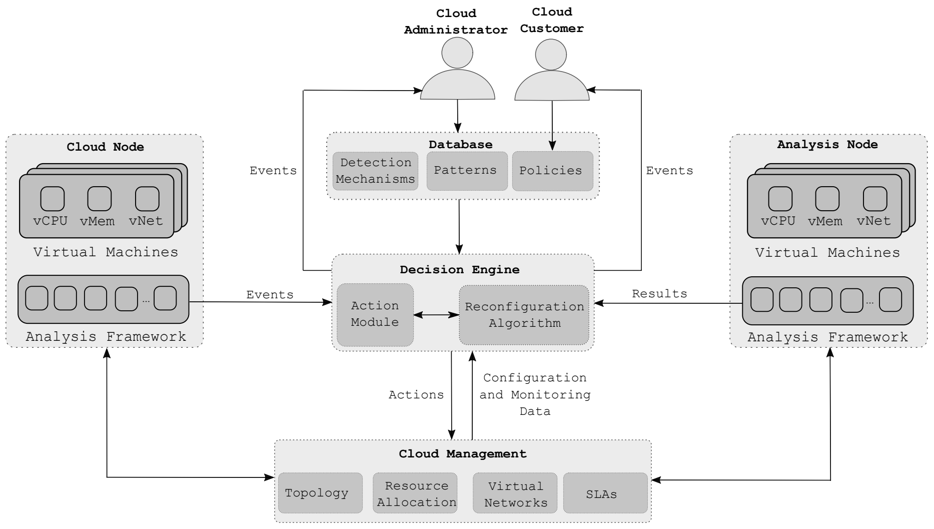

Fig. 4. Decision Engine Interfaces

network defines the cloud hosts and analysis nodes and detection mechanisms deployed in them, the resource capacities and usage, and the network topology. The virtual networks deployed in the environment define the topology, resource demands, and constraints of the virtual networks. An example constraint is the maximum communication latency between two VMs. The resource allocation defines how resources are allocated to the virtual networks in the cloud environment. The SLAs define the QoS requirements of the virtual networks such as service availability. This availability could modeled as a downtime budget for each VM.

The second input comes from three databases. The *behavior patterns* database is maintained by the cloud administrator and by the analysis nodes that update the information about possible attacks based on the results of the analysis. This database includes pre-defined behavior patterns that refer to a probability of a certain attack. This database consists of records of the form: *{Pattern, Parameter ranges, Suspected malware, Possible attacks, Suspicion level}*. An example of such a record is: *{Network traffic spike, Period $>$ 5 seconds, Unknown, DoS, 50%}*. The *policy* database is maintained by both the cloud administrator and customer. It specifies a list of actions that should be executed by the DE when an event from a detection mechanism or an analysis result from an analysis node is received for a specified VM. This database consists of records of the form: *{Pattern, VM ID, Actions}*. An example of such a record is: *{Network traffic spike, VM 100, block}*. The *detection mechanisms* database is maintained by the cloud administrator. It is used by the DE mainly to find the estimated resource requirements of each mechanism. The structure of this database can only be determined after an extensive evaluation of the resource utilization by the required detection mechanisms.

The third input of the DE is the events received from detection mechanisms. The detection mechanisms running in cloud nodes should have access to the behavior patterns database. When a certain behavior is detected, an event is reported to the DE. The event includes the following information: *{VM ID, node ID, pattern, Parameters}*. An example of such an event is: *{VM 100, Node 1, Memory usage spike, Period:5 seconds}*.

The last input is the analysis results received from analysis nodes. When a suspicious VM is migrated to an analysis node, the node sends the analysis results to the decision engine that executes further actions defined in the policy.

Actions The DE includes an action module that receives the events from the detection mechanism, reads the behavior and policy database, and determines the required list of actions to respond to this event. Possible simple cloud actions are blocking the network connections of a VM when the VM is suspected to be performing a network attack, restarting a VM when facing a transient attack, and shutting down the VM immediately when an attack is highly probable and the damage that might be caused to the environment cannot be easily recovered. A significant action is activating additional detection mechanisms in the cloud node to perform a more detailed analysis on the VM if the required resources for running these mechanisms are available in the node. The most important action in the DE is migrating the VM to a dedicated analysis node where an extensive analysis can be performed without interrupting the functionality of this VM. A similar action is cloning the VM and moving the copy to an analysis node when service interruption is not feasible and the suspicion level is low. One important action after receiving an analysis result of a migrated or cloned VM is recovering the VM in case the analysis identifies the VM as harmless.

Reconfiguration algorithm To perform the cloning and migration actions, the DE uses a reconfiguration algorithm that reads the configuration and monitoring data for the affected virtual network and finds the available analysis nodes. In the cloning action, the algorithm has only to find an analysis node that has enough resource capacities to host the VM and perform a detailed analysis. In the migration action, the algorithm is more complex, and three main factors should be considered in the decision making. The first factor is the migration downtime. The live migration of VMs usually causes a small downtime. However, a maximum downtime should be estimated before the migration is performed. The migration downtime mainly depends on the memory usage of the VM, the network bandwidth, and migration strategy. This downtime should be then compared with the downtime budget of the VM. The second factor is the resource capacities of the analysis nodes that have to match the requirements of the VM. The last factor is VMs dependencies in the virtual network. A typical example of these dependencies is the communication delay among virtual machines [20]. The dependencies might require a full or partial reconfiguration of the virtual network when a VM is migrated to an analysis node. The reconfiguration might need to migrate other VMs to keep the constraints of the virtual network satisfied. The migration downtime and resource requirements of the these VMs should also be considered.

The decision engine waits for the events or analysis results from the production environment and the analysis environment respectively. On one hand, when an event is received, the appropriate action will be performed according to the event parameters, detected behavior, and security policies. The migration action needs to adhere to the service requirements of the VM. It might also require a reconfiguration of the virtual network according to its dependencies. On the other hand, when an analysis result is received, the appropriate action is performed depending on the VM state by either recovering the original configuration of the virtual network if no attack is identified, or performing a certain action according to the policies if an attack is identified.

4 Evaluation and Future Work

Performance In order to illustrate the classification of monitoring tools as lightweight or heavyweight, we have implemented a framework for monitoring system calls of VMs, which supports four different monitoring configurations: (1) without any tracing system calls (**none**), (2) tracing **execve** system calls, (3) tracing **open** system calls and (4) tracing **all** system calls of the virtual machine. The system call tracing is implemented by inserting software breakpoints into the guest system and by handling the corresponding interrupts using libVMI[8]. The overhead depends not only on the monitoring mechanism, but also on the software running within the virtual machines. For this reason, we measured the overhead for four different use cases executed within the VM: (1) the extraction of a Linux Kernel archive, (2) the compilation of the Linux kernel, (3) writing of 1.2 GB file with zero bytes and (4) the computation of 5000 digits of the number pi. Table 5 presents the results of the measurements.

Based on these results we can do a very basic classification: By tracing only the `execve` or `open` system call, the additional overhead is still acceptable and can be further decreased by a faster implementation. Tracing the `execve` system call is for example useful for forensics purposes, e.g., to log the executed commands of an attacker or to perform intrusion detection. Hence, tracing a small set of system calls can be classified as lightweight.

The overhead which is required to monitor all system calls is obviously higher but it also allows to get a better insight of the processes in a VM, e.g., in order to do malware analysis. Thus, tracing all system calls can be classified as heavyweight analysis. Another example for a heavyweight analysis is the tracing of library calls, e.g., `malloc` in the libc library. The decision about which tracing strategy to apply depends on the current threat level of a VM. However, the computation of the threat level and the corresponding action needs to be investigated in more detail in future work.

[8] libVMI - http://libvmi.com/

Usecase		**#syscalls**	$\mathbf{t_{real}}$**(s)**	overhead	$\mathbf{t_{user}}$**(s)**	overhead	$\mathbf{t_{sys}}$**(s)**	overhead
extract	none	0	11.350		6.508		1.664	
	execve	5	13.038	14.8%	6.560	0.80%	1.692	16.8%
	open	54	11.810	4.05%	6.532	0.37%	2.384	43.3%
	all	65.7k	116.664	928%	6.940	6.64%	108.936	655%
compile	none	0	657.508		595.704		24.116	
	execve	33k	849.250	29.2%	635.556	6.69%	176.404	631%
	open	4022k	1663.985	153%	651.360	9.34%	651.360	270%
	all	13539k	2987.852	354%	660.120	10.8%	2248.744	922%
write	none	0	10.008		0.028		0.924	
	execve	1	10.729	7.20%	0.044	57.1%	0.972	5.19%
	open	12	10.081	0.73%	0.036	28.6%	0.928	4.32%
	all	602k	94.324	842%	0.220	686%	93.220	1000%
compute	none	0	20.413		20.408		0.004	
	execve	1	20.437	0.61%	20.428	0.1%	0.004	0%
	open	5	20.460	0.23%	20.448	0.2%	0.004	0%
	all	1080	20.767	1.73%	20.450	0.2%	0.100	250%

Fig. 5. Overhead generated by tracing system calls of a virtual machine. t_{real} is the total execution time, t_{user} is the time where the programming was running in user space and t_{sys} is the time spent in system calls (times measured with the Linux tool `time`). The overhead is computed in relation to the case when no tracing is active.

Security Currently, the analysis framework runs in the XEN Dom0. Thus, if an intruder is able to exploit it he gets full control over all other VMs. To improve the security level and in order to increase the trustworthiness of the forensics data we plan to start the analysis framework in a separate and very minimal monitoring VM for each production VM. Thereby each monitoring VM shall be restricted to access only one production VM, e.g., by applying XSM flask policies [6].

Limitations of VMI In order to execute VMI, a priori knowledge about the guest operating system is required in order to bridge the semantic gap and interpret the guest memory. We assume that a cloud customer who wants to benefit from CloudIDEA makes this information available to analysis framework. In future work we will discuss whether malware can fool the interpretation of the analysis framework.

5 Related work

Several researchers have addressed the problem of *evidence collection in the cloud*. Zafarullah et al. [51] propose a centralized approach to analyze log files in an IaaS environment, in which the cloud provider is responsible for forensic investigations. Martini and Choo describe an integrated conceptual digital forensic framework for cloud computing [29]. Dykstra and Sherman [12] present FROST, a digital forensics tool for the OpenStack cloud platform that enables the acquisition of virtual disk snapshots, API logs, and guest firewall logs. While these approaches focus on enabling interactive forensic data collection, our proposed architecture enhances evidence collection by providing a analysis framework on each host of a production system and by automatically deciding which detector module to activate in suspicious situations. Poisel et al. [33] they discuss the possibility of acquiring evidence at the hypervisor level using virtual machine introspection, and identify the need for extending such approaches from evidence collection on a single host to a larger setup with multiple cloud nodes.

A second area related to CloudIDEA is the *detection of malware in the cloud*. Harrison et al. [17] define a framework for detecting malware in the cloud by identifying the symptoms of malicious behaviour. Small "Forensic Virtual Machines" (FVMs) monitor customer VMs for specific malware symptoms using VMI and collaborate with each other by exchanging messages via secure channels. Examples for detectable symptoms are missing processes such as sysclean and tcpview, modification of in-memory code such winlogon, absence of antivirus software from the process table, and snippets of program code that has been obfuscated or uses certain cryptographic algorithms. FVMs report to a command & control module that collects and correlates the information so that suitable remedial actions can take place in real-time. For mitigating the cost of monitoring, the authors propose a mobility algorithm in which FVMs monitor only a small subset of all VMs that changes in time in an unpredictable way. While the proposed framework shares some similarities with our approach, it lacks means for evidence preservation and it is not intended for continuous lightweight monitoring of all VMs. Also, we propose a DE that automatically decides taking appropriate actions in a more flexible way.

Schmidt et al. [37] present an architecture for malware detection and kernel rootkit prevention in cloud environments. In this approach, all running binaries are intercepted by a small in-kernel agent and submitted to one or more backend units where the actual classification process happens. The in-kernel agent can be deployed by the cloud provider by replacing the OS kernel within the customer's VM image. In contrast, CloudIDEA aims at detecting and analyzing malware in the cloud without depending on modifications of the guest system.

Many researchers addressed the *behavior study* of both traditional malware and virtual environment specific malware in cloud environments. For example, Dolgikh et al. [10] propose an efficient behavioral modeling scheme to detect suspicious processes in client VMs by monitoring system calls. Manerides et al. [28] describe how to detect the traditional Kelihos malware in virtual machines by monitoring the memory usage and number of processes. According to the authors, Kelihos malware causes a memory explosion for few seconds, which is not a normal behavior for traditional applications. Zhang et al. [52] use machine learning to deploy a lightweight mechanism in a VM to detect the behavior of L2 cache side channel attacks performed by other VMs.

Another aspect that influences the quality of a cloud malware detection and analysis environment is the *stealthiness of analysis*, i.e., the question whether malware can detect if it is being analysed. Cobra, a framework by Vasudevan and Yerraballi [43], is known to be one of the first malware dynamic analysis frameworks with an explicit emphasis on stealth as a design goal, also providing support for self-modifying, self-checking code, and any form of code obfuscation. A somewhat stronger solution was proposed by Dinaburg et al. [8]. Their framework, Ether, is based on hardware virtualization extensions, e.g., Intel VT. The analysis functionality is facilitated by the Xen hypervisor and therefore resides completely outside the target OS. Ether is capable of monitoring single instruction executions, memory writes, and system calls. Since it does not have any presence in the guest OS, it is immune against the most of detection methods, except some timing attacks. Spider [7] is another tool based on hardware virtualization and provides stealth binary instrumentation and debugging capabilities. Drakvuf by Lengyel et al. [26], also based on hardware virtualization, is able to track behavior of both user- and kernel-level malware. Willems et al. [46] proposed an approach to stealth dynamic malware analysis based on the *branch tracing* feature of modern CPUs. The use of this feature can provide full execution traces, and could only be detected by ring-0 programs or by means of timing attacks. However, with this approach neither the process memory state nor CPU registers can be revealed. Kirat et al. [24] proposed a method of detecting Malware with analysis-evasion capabilities by executing it within different environments, for which they have used Anubis, based on emulation, Ether, based on Xen hypervisor, Cuckoo sandbox based on virtualization, and on "bare metal" devices. By considering the hierarchical similarity of the obtained behavior profiles, they were able to successfully detect evasive malware.

6 Conclusion

CloudIDEA enables cloud customers to benefit from advanced techniques for attack detection, analysis, and evidence preservation in case of malicious attacks against VMs running on IaaS infrastructures. CloudIDEA combines continuous monitoring on production systems with in-depth analysis, including the possibility to migrate VMs to a dedicated analysis environment.

The low-level analysis framework allows using a configurable set of tracing and introspection plug-ins, which include lightweight techniques that are suitable for continuous monitoring as well as heavyweight techniques that have a significant run-time overhead and can be activated on demand. Measurements on our prototype system illustrate the run-time cost of several monitoring techniques.

The high-level decision engine automates the dynamic selection of monitoring plug-ins and reconfigurations of the virtual machine and virtual network deployments. Within constraints regarding quality of service (as defined in SLAs) and security requirements (policies defined by cloud customer and cloud provider) it takes decision about actions such as terminating a VM or isolating it in a dedicated analysis environment.

In summary, the CloudIDEA architecture is a step forward in defending against malware in cloud infrastructures.

References

1. Akoush, S., Sohan, R., Rice, A., Moore, A., Hopper, A.: Predicting the performance of virtual machine migration. In: IEEE Int. Symp. on Modeling, Analysis Simulation of Comp. and Telecomm. Systems (MASCOTS). pp. 37–46 (2010)
2. Bitdefender: Xen: Emulate with no writes (2014), `http://lists.xen.org/archives/html/xen-devel/2014-08/msg00264.html`
3. Butler, J.: DKOM (direct kernel object manipulation). Black Hat Windows Security (2004)
4. Butler, J., Silberman, P.: Raide: Rootkit analysis identification elimination. Black Hat USA 47 (2006)
5. Chen, P.M., Noble, B.D.: When virtual is better than real. In: Proc. of the 8th Workshop on Hot Topics in Operating Systems. pp. 133–138. IEEE (2001)
6. Coker, G.: Xen security modules (xsm). `http://mail.xen.org/files/summit_3/coker-xsm-summit-090706.pdf` (March 24 2015)
7. Deng, Z., Zhang, X., Xu, D.: SPIDER: stealthy binary program instrumentation and debugging via hardware virtualization. In: Proc. of the 29th Annual Computer Security Applications Conference. pp. 289–298. ACSAC '13, ACM (2013)
8. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: Malware analysis via hardware virtualization extensions. In: Proceedings of the 15th ACM Conference on Computer and Communications Security. pp. 51–62. CCS '08, ACM (2008)
9. Dolan-Gavitt, B., Srivastava, A., Traynor, P., Giffin, J.: Robust signatures for kernel data structures. In: Proceedings of the 16th ACM conference on Computer and communications security. pp. 566–577. ACM (2009)
10. Dolgikh, A., Birnbaum, Z., Chen, Y., Skormin, V.: Behavioral modeling for suspicious process detection in cloud computing environments. In: IEEE 14th Int. Conf. on Mobile Data Management (MDM). vol. 2, pp. 177–181 (June 2013)
11. Dontu, M., Sahita, R.: Zero-footprint guest memory introspection from xen. `http://www.xenproject.org/component/allvideoshare/video/xpds14-introspection.html` (January 15 2015)
12. Dykstra, J., Sherman, A.T.: Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. Digit. Investig. 10, 87–95 (2013)
13. Florio, E.: When malware meets rootkits. Virus Bulletin (2005)
14. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: In Proc. Network and Distributed Systems Security Symposium. pp. 191–206 (2003)
15. Gionta, J., Azab, A., Enck, W., Ning, P., Zhang, X.: Seer: practical memory virus scanning as a service. In: Proceedings of the 30th Annual Computer Security Applications Conference. pp. 186–195. ACM (2014)
16. Gonzalez, N., Miers, C., Redigolo, F., Carvalho, T., Simplicio, M., Naslund, M., Pourzandi, M.: A quantitative analysis of current security concerns and solutions for cloud computing. In: Proc. of the 2011 IEEE 3rd Int. Conf. on Cloud Computing Technology and Science. pp. 231–238. CLOUDCOM '11, IEEE CS (2011)
17. Harrison, K., Bordbar, B., Ali, S., Dalton, C., Norman, A.: A framework for detecting malware in cloud by identifying symptoms. In: IEEE 16th Int. Enterprise Distributed Object Computing Conference (EDOC). pp. 164–172 (Sept 2012)
18. Heller, K., Svore, K., Keromytis, A.D., Stolfo, S.: One class support vector machines for detecting anomalous windows registry accesses. In: Workshop on Data Mining for Computer Security (DMSEC). pp. 2–9 (2003)

19. Hofmeyr, S.A., Somayaji, A., Forrest., S.: Intrusion detection using sequences of system calls. Journal of Computer Security 6, 151–180 (1998)
20. Ivaturi, K., Wolf, T.: Mapping of delay-sensitive virtual networks. In: Int. Conf. on Computing, Networking and Communications (ICNC). pp. 341–347 (2014)
21. Jain, B., Baig, M.B., Zhang, D., Porter, D.E., Sion, R.: Sok: Introspections on trust and the semantic gap. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. pp. 605–620. SP '14, IEEE CS (2014)
22. Jansen, R., Brenner, P.: Energy efficient virtual machine allocation in the cloud. In: Int. Green Computing Conference and Workshops (IGCC). pp. 1–8 (July 2011)
23. Jiang, X., Wang, X., Xu, D.: Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In: Proc. of the 14th ACM Conference on Computer and Communications Security. pp. 128–138. CCS '07, ACM (2007)
24. Kirat, D., Vigna, G., Kruegel, C.: Barecloud: Bare-metal analysis-based evasive malware detection. In: Proc.s of the 23rd USENIX Conference on Security Symposium. pp. 287–301. SEC'14, USENIX Association, Berkeley, CA, USA (2014)
25. Kittel, T., Vogl, S., Lengyel, T.K., Pfoh, J., Eckert, C.: Code validation for modern os kernels. In: Workshop on Malware Memory Forensics (MMF) (Dec 2014)
26. 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: Proc. of the 30th Annual Computer Security Applications Conference (2014)
27. Lobo, D., Watters, P., Wu, X., Sun, L., et al.: Windows rootkits: Attacks and countermeasures. In: 2010 Second Cybercrime and Trustworthy Computing Workshop. pp. 69–78. IEEE (2010)
28. Marnerides, A., Watson, M., Shirazi, N., Mauthe, A., Hutchison, D.: Malware analysis in cloud computing: Network and system characteristics. In: Globecom Workshops (GC Wkshps), 2013 IEEE. pp. 482–487 (Dec 2013)
29. Martini, B., Choo, K.R.: An integrated conceptual digital forensic framework for cloud computing. Digital Investigation 9(2), 71–80 (2012)
30. Payne, B.D., Carbone, M., Sharif, M., Lee, W.: Lares: An architecture for secure active monitoring using virtualization. In: Security and Privacy, 2008. SP 2008. IEEE Symposium on. pp. 233–247. IEEE (2008)
31. Perez-Botero, D., Szefer, J., Lee, R.B.: Characterizing hypervisor vulnerabilities in cloud computing servers. In: Proc. of the 2013 Int. Workshop on Security in Cloud Computing. pp. 3–10. Cloud Computing '13, ACM (2013)
32. Pfoh, J., Schneider, C., Eckert, C.: Leveraging string kernels for malware detection. In: Proc. of the 7th Int. Conf. on Network and System Security. Springer (2013)
33. Poisel, R., Malzer, E., Tjoa, S.: Evidence and cloud computing: The virtual machine introspection approach. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) 4(1), 135–152 (3 2013)
34. Rhee, J., Riley, R., Xu, D., Jiang, X.: Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory. In: Recent Advances in Intrusion Detection. Springer (2010)
35. Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic analysis of malware behavior using machine learning. Journal of Computer Security 19(4), 639–668 (2011)
36. Salfner, F., Tröger, P., Richly, M.: Dependable Estimation of Downtime for Virtual Machine Live Migration. Int. J. on Advances in Systems and Measurements 5 (2012)
37. Schmidt, M., Baumgartner, L., Graubner, P., Bock, D., Freisleben, B.: Malware detection and kernel rootkit prevention in cloud computing environments. In: Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on. pp. 603–610 (Feb 2011)

38. Shea, R., Liu, J.: Performance of virtual machines under networked denial of service attacks: Experiments and analysis. Systems Journal, IEEE 7(2), 335–345 (2013)
39. Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: All your clouds are belong to us: Security analysis of cloud management interfaces. In: Proc. of the 3rd ACM Workshop on Cloud Computing Security. pp. 3–14. CCSW '11, ACM, New York, NY, USA (2011)
40. Studnia, I., Alata, E., Deswarte, Y., Kaaniche, M., Nicomette, V.: Survey of security problems in cloud computing virtual machines. Tech. rep., CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France (2012)
41. Szekeres, L., Payer, M., Wei, T., Song, D.: Sok: Eternal war in memory. In: IEEE Symp. on Security and Privacy. pp. 48–62. IEEE (2013)
42. Tegeler, F., Fu, X., Vigna, G., Kruegel, C.: Botfinder: Finding bots in network traffic without deep packet inspection. In: Proc. of the 8th int. conf. on Emerging networking experiments and technologies. pp. 349–360. ACM (2012)
43. Vasudevan, A., Yerraballi, R.: Cobra: fine-grained malware analysis using stealth localized-executions. In: IEEE Symp. on Security and Privacy. pp. 15–279 (2006)
44. Voorsluys, W., Broberg, J., Venugopal, S., Buyya, R.: Cost of virtual machine live migration in clouds: A performance evaluation. In: 1st Int. Conf. on Cloud Computing (CloudCom 2009). pp. 254–265. Springer Berlin/Heidelberg (2009)
45. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proc. of the IEEE Symp. on Security and Privacy. pp. 133–145. IEEE (1999)
46. Willems, C., Hund, R., Fobian, A., Felsch, D., Holz, T., Vasudevan, A.: Down to the bare metal: Using processor features for binary analysis. In: Proc. of the 28th Ann. Computer Security Applications Conf. (ACSAC). pp. 189–198. ACM (2012)
47. Willems, C., Hund, R., Holz, T.: Cxpinspector: Hypervisor-based, hardware-assisted system monitoring. Tech. rep., Ruhr-Universitat Bochum (2013)
48. Wood, T., Cecchet, E., Ramakrishnan, K.K., Shenoy, P., van der Merwe, J., Venkataramani, A.: Disaster recovery as a cloud service: Economic benefits & deployment challenges. In: Proc. of the 2nd USENIX Conf. on Hot Topics in Cloud Computing. p. 8. HotCloud'10, USENIX Association (2010)
49. Xiao, H., Stibor, T.: A supervised topic transition model for detecting malicious system call sequences. In: Proceedings of the 2011 workshop on Knowledge discovery, modeling and simulation. pp. 23–30. ACM (2011)
50. Yin, H., Poosankam, P., Hanna, S., Song, D.: Hookscout: Proactive binary-centric hook detection. In: Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 1–20. Springer (2010)
51. Zafarullah, Anwar, F., Anwar, Z.: Digital forensics for Eucalyptus. In: Proc. of the 2011 Frontiers of Information Technology. pp. 110–116. FIT '11, IEEE CS (2011)
52. Zhang, Y., Juels, A., Oprea, A., Reiter, M.: Homealone: Co-residency detection in the cloud via side-channel analysis. In: IEEE Sympl. on Security and Privacy. pp. 313–328 (May 2011)
53. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-vm side channels and their use to extract private keys. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security. pp. 305–316. CCS '12, ACM (2012)