Journal of Network and Computer Applications ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Secure virtualization for cloud computing

Flavio Lombardi[a], Roberto Di Pietro[b,c,*]

[a] Consiglio Nazionale delle Ricerche, DCSPI-Sistemi Informativi, Piazzale Aldo Moro 7, 00187 Roma, Italy
[b] Università di Roma Tre, Dipartimento di Matematica, L.go S. Leonardo Murialdo, 1 00149 Roma, Italy
[c] UNESCO Chair in Data Privacy, Universitat Rovira i Virgili, Tarragona, Spain

ARTICLE INFO

Article history:
Received 29 November 2009
Received in revised form
23 April 2010
Accepted 7 June 2010

Keywords:
Security
Cloud computing
Virtualization technologies

ABSTRACT

Cloud computing adoption and diffusion are threatened by unresolved security issues that affect both the cloud provider and the cloud user. In this paper, we show how virtualization can increase the security of cloud computing, by protecting both the integrity of guest virtual machines and the cloud infrastructure components. In particular, we propose a novel architecture, Advanced Cloud Protection System (ACPS), aimed at guaranteeing increased security to cloud resources. ACPS can be deployed on several cloud solutions and can effectively monitor the integrity of guest and infrastructure components while remaining fully transparent to virtual machines and to cloud users. ACPS can locally react to security breaches as well as notify a further security management layer of such events. A prototype of our ACPS proposal is fully implemented on two current open source solutions: Eucalyptus and OpenECP. The prototype is tested against effectiveness and performance. In particular: (a) effectiveness is shown testing our prototype against attacks known in the literature; (b) performance evaluation of the ACPS prototype is carried out under different types of workload. Results show that our proposal is resilient against attacks and that the introduced overhead is small when compared to the provided features.

1. Introduction

Internet is on the edge of another revolution, where resources are globally networked and can be easily shared. *Cloud computing* is the main component of this paradigm, that renders the Internet a large repository where resources are available to everyone as services. In particular, cloud nodes are increasingly popular even though unresolved security and privacy issues are slowing down their adoption and success. Indeed, integrity, confidentiality, and availability concerns are still open problems that call for effective and efficient solutions. Cloud nodes are inherently more vulnerable to cyber attacks than traditional solutions, given their size and underlying service-related complexity—that brings an unprecedented exposure to third parties of services and interfaces. In fact, the cloud ''is'' the Internet, with all the pros and cons of this pervasive system. As a consequence, increased protection of cloud internetworked nodes is a challenging task. It becomes then crucial to recognize the possible threats and to establish security processes to protect services and hosting platforms from attacks.

Cloud Computing already leverages virtualization for load balancing via dynamic provisioning and migration of virtual machines (VM or *guest* in the following) among physical nodes. VMs on the Internet are exposed to many kinds of interactions that virtualization technology can help filtering while assuring a higher degree of security. In particular, virtualization can also be used as a security component; for instance, to provide monitoring of VMs, allowing easier management of the security of complex cluster, server farms, and cloud computing infrastructures to cite a few. However, virtualization technologies also create new potential concerns with respect to security, as we will see in Section 4.

Contributions: The goal of this paper is twofold: (a) to investigate the security issues of cloud computing; (b) to provide a solution to the above issues.

We analyzed cloud security issues and model, examined threats and identified the main requirements of a protection system. In particular, we developed an architecture framework, Advanced Cloud Protection System (ACPS), to increase the security of cloud nodes. ACPS is based on the results of KvmSec (Lombardi and Di Pietro, 2009) and KvmSma (Lombardi and Di Pietro, 2010) prototype security extensions of the Linux Kernel Virtual Machine (KVM Qumranet, year), It is also inspired by the TCPS architecture (Lombardi and Di Pietro, 2010). ACPS is a complete protection system for clouds that transparently monitors cloud components

* Corresponding author at: Università di Roma Tre, Dipartimento di Matematica, L.go S. Leonardo Murialdo, 1 00149 Roma, Italy. Tel.: +39 06 57338246.
E-mail addresses: flavio.lombardi@cnr.it (F. Lombardi), dipietro@mat.uniroma3.it, roberto.dipietro@urv.cat (R. Di Pietro).

1084-8045/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2010.06.008

and interacts with local and remote parties to protect and to recover from attacks.

In the following we show how ACPS can leverage full virtualization to provide increased protection to actually deployed cloud systems such as Eucalyptus (Nurmi et al., 2009) and (Openecp, 2010) (also referred to as Enomalism Enomaly, 2009 in the following). In fact, OpenECP is a fully open source code fork of the previously open source Enomalism offer; as such, it shares the same architecture and codebase. A prototype implementation is presented. Its effectiveness and performance are tested. Results indicate that our proposal is resilient against attacks and that the introduced overhead is small—especially when compared to the features provided.

One main outcome of our research is a framework that allows virtualization-supported cloud protection across physical hosts over the Internet.

Roadmap. The remainder of this document is organized as follows: next section surveys related work. Section 3 provides background information, while Section 4 classifies cloud security issues. Section 5 describes ACPS requirements and architecture. In Section 6 implementation details are provided, while effectiveness and performance are discussed in Section 7. Finally, Section 8 draws some conclusions.

2. Related work

While privacy issues in clouds have been described in depth by Pearson (2009), cloud security is less discussed in the literature (Gu and Cheung, 2009). Some interesting security issues are discussed in Siebenlist (2009), while an almost complete survey of security in the context of cloud storage services is provided by Cachin et al. (2009). An exhaustive cloud security risk assessment has been recently presented by Enisa (2009). Also worth reading is the survey on cloud computing presented in Armbrust et al. (2009). These papers have been the starting points of our work and we refer to them in terms of problems and terms definition.

A fundamental reference for our research is the work on co-location (Ristenpart, 2009) by Ristenpart. This work shows that it is possible to instantiate an increasing number of guest VMs until one is placed co-resident with the target VM. Once successfully achieved co-residence, attacks can theoretically extract information from a target VM on the same machine. An attacker might also actively trigger new victim instances exploiting cloud auto-scaling systems. Ristenpart shows that it practical to hire additional VMs whose launch can produce a high chance of co-residence with the target VM. He also shows that determining co-residence is quite simple.

Most current integrity monitoring and intrusion detection solutions can be successfully applied to cloud computing. Filesystem Integrity Tools and Intrusion Detection Systems such as *Tripwire* (Kim and Spafford, 1994) and (*AIDE*) (AIDEteam, 2005) can also be deployed in virtual machines, but are exposed to attacks possibly coming from a malicious guest machine user. Furthermore, when an attacker detects that the target machine is in a virtual environment, it may attempt to break out of the virtual environment through vulnerabilities (very rare at the time of writing Secunia, 2009) in the Virtual Machine Monitor (VMM). Most present approaches leverage VMM isolation properties to secure VMs by leveraging various levels of virtual introspection. Virtual introspection (Jiang et al., 2007) is a process that allows to observe the state of a VM from the VMM. *SecVisor* (Seshadri et al., 2007) *Lares* (Payne et al., 2008) and *KVM-L4* (Peter et al., 2009), to name a few, leverage virtualization to observe and monitor guest kernel code integrity from a privileged VM or from the VMM. *Nickle* (Riley et al., 2008) aims at detecting kernel rootkits by monitoring the integrity of kernel code. However, *Nickle* does not protect against kernel data attacks (Rhee et al., 2009), whereas our solution does. Most proposals have limitations that prevent them from being used in distributed computing scenarios (e.g.. *SecVisor* only supports one guest per each host) or just do not consider the special requirements or peculiarities of distributed systems; for instance, *KVM-L4* shares the same underlying technology as Lombardi and Di Pietro (2009) but the additional context switching overhead in the 64-bit scenario, representing the vast majority of cloud hosts, remains to be verified. Also worth citing are *IBMon* (Ranadive et al., 2009), a monitoring utility using introspection for asynchronous monitoring of virtualized network devices, and *LoGrid* (Salza et al., 2006), an example of autonomic reaction system.

In an effort to make nodes resilient against long-lasting attacks, *Self-Cleansing Intrusion Tolerance* (SCIT) (Huang et al., 2006) treats all servers as potentially compromised (since undetected attacks are extremely dangerous over time). SCIT restores servers from secure images on a regular basis. The drawback of such a system is that it does not support long-lasting sessions required by most cloud applications. Similarly, *VM-FIT* (Distler et al., 2008) creates redundant server copies which can periodically be refreshed to increase the resilience of the server. Finally, Sousa et al. (2007) approach combines proactive recovery with services that allow correct replicas to react and be recovered when there is a sufficient probability that they have been compromised. Along with the many advantages brought by virtualization, there are additional technological challenges that virtualization presents, which include an increase in the complexity of digital forensics (Pollitt et al., 2008) investigations as well as questions regarding the forensics boundaries of a system.

Finally, the same authors of this paper proposed Transparent Cloud Protection System (TCPS)—appearing as a poster at SAC'10 (Lombardi and Di Pietro, 2010). That poster introduces some of the scenarios and requirements that are also common to ACPS, however they are only partly sketched in TCPS. In particular, ACPS and TCPS share the positioning of the monitoring system and the requirement that it has to be as much transparent as possible to guests. ACPS extends and completes the architecture just sketched in TCPS. For instance, ACPS enjoys unique features, such as the SWADR approach, the increased decoupling of action and reaction, the increased immunity and integrity of the platform—as well as the integration with real-world architecture—and the support for accountability. All these new relevant features, as well as extensive experiments on both security and performance, make the present proposal a novel contribution (see also Table 1).

Table 1
Comparison of features provided by ACPS, TCPS, KvmSma (KSma) and KvmSec (KSec).

Feature	KSec	KSma	TCPS	ACPS
Semantic View	N	Y	Y	Y
Guest Component	Y	N	N	N
Transparency	N	Y	Part.	Full
Non-Blocking	Y	Y	Y	Y
SWADR	N	N	N	Y
Hot Recovery (by Replacement)	N	N	N	Y
Accountability	N	N	N	Y

3. Background

A cloud (Vaquero et al., 2009) is a pool of virtualized resources across the Internet that follows a pay-per-use model and can be dynamically reconfigured to satisfy user requests via on-the-fly

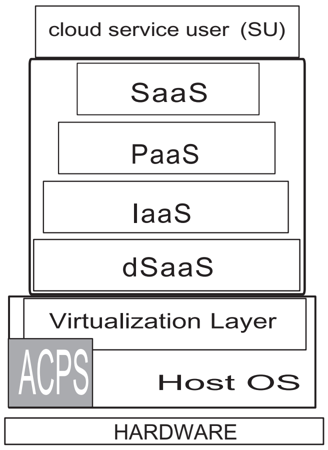

Fig. 1. Cloud layers and the advanced cloud protection system.

provisioning/deprovisioning of virtual machines. Cloud computing is a service model for IT provisioning, often based on virtualization and distributed computing technologies. Within the cloud paradigm, concepts such as virtualization, distributed computing and utility computing are applied (Lenk et al., 2009). Cloud computing approach to distributed computing shares many ideas with grid computing, but these two differ in target, in focus, and in the implementation technologies (Foster et al., 2009). On the one hand, with respect to a grid, the cloud user has less control over the location of data and computation. On the other hand, cloud computing management costs are usually much lower and management is less cumbersome. In the following we will also refer to the cloud infrastructure components as middleware.

Cloud services are available at different layers (see the *-as-a-Service or *aaS layers in Fig. 1): *dSaaS* The data Storage as a Service delivering basic storage capability over the network; *IaaS* The Infrastructure as a Service layer providing bare virtual hardware with no software stack; *PaaS* The Platform as a Service layer providing a virtualized servers, OS, and applications; *SaaS* The Software as a Service layer providing access to software over the Internet as a service.

In this work, efforts have been focused on the "lowest" computational layer (i.e. *IaaS*) since we can more effectively provide a security foundation on top of which more secure services can be offered. Most existing cloud computing systems are proprietary (even though APIs are open and well-known) and as such do not allow modifications, enhancements or integration with other systems for research purposes. This is the reason why we have chosen Eucalyptus and OpenECP, both open source cloud implementations, for integration with our architecture. In the following, even though we will focus on the security issues of those two platforms, most considerations will be general enough to be valid for other platforms as well.

4. Cloud security issues

One of the key issues of cloud computing (see Fig. 1) is loss of control. As a first example, the service user (SU) does not know where exactly its data is stored and processed in the cloud. Computation and data are mobile and can be migrated to systems the SU cannot directly control. Over the Internet, data is free to cross international borders and this can expose to further security threats. A second example of loss of control is that the cloud provider (CP) gets paid for running a service he does not know the details of. This is the dark side of the "Infrastructure as a Service" model, but also of other "as a Service" approaches. To date, misuse problems tend to be regulated by a service contract, where such an agreement should be enforced and controlled by monitoring tools (Haeberlen, 2009).

Some of the security issues of a cloud are (Foster et al., 2009):

SEI1	Privileged user access: access to sensitive outsourced data has to be limited to a subset of privileged users (to mitigate the risk of abuse of high privilege roles);
SEI2	Data segregation: one instance of customer data has to be fully segregated from other customer data;
SEI3	Privacy: exposure of sensitive information stored on the platforms implies legal liability and loss of reputation;
SEI4	Bug Exploitation: an attacker can exploit a software bug to steal valuable data or to take over resources and allow for further attacks;
SEI5	Recovery: the cloud provider has to provide an efficient replication and recovery mechanism to restore services, should a disaster occur;
SEI6	Accountability: even though cloud services are difficult to trace for accountability purposes, in some cases this is a mandatory application requirement.

With respect to the latter point, accountability can increase security and reduce risks for both the service user and the service provider. A trade-off between privacy and accountability exists, since the latter produces a record of actions that can be examined by a third party when something goes wrong. Such an investigation might show faulty components or internal cloud resource configuration details. This way, a cloud customer might be able to learn information about the internal structure of the cloud that could be used to perform an attack. A possible solution could be the use of obfuscation and privacy-preserving techniques to limit the information the VM exposes to the cloud (Bethencourt et al., 2009). Anyway, current technology cannot prevent a VMM from accessing guest raw memory. This leaves open confidentiality issues with respect to the service provider (or with respect to an attacker if he compromises the hosting platform).

4.1. Cloud security model

Fig. 2 illustrates the scenario we are concerned with in this paper. A service provider (SP) runs one or more service instances (SI) on the cloud, which can be accessed by a group of final service

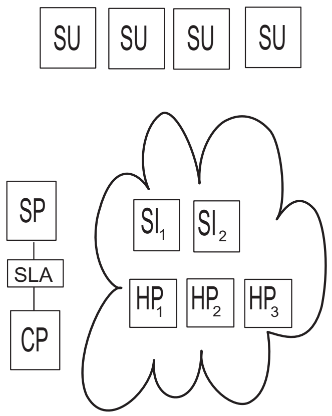

Fig. 2. Cloud service model components: Cloud Provider (CP), Hosting Platform (HP), Service Level Agreement (SLA), Service Provider (SP), Service Instance (SI), Service User (SU).

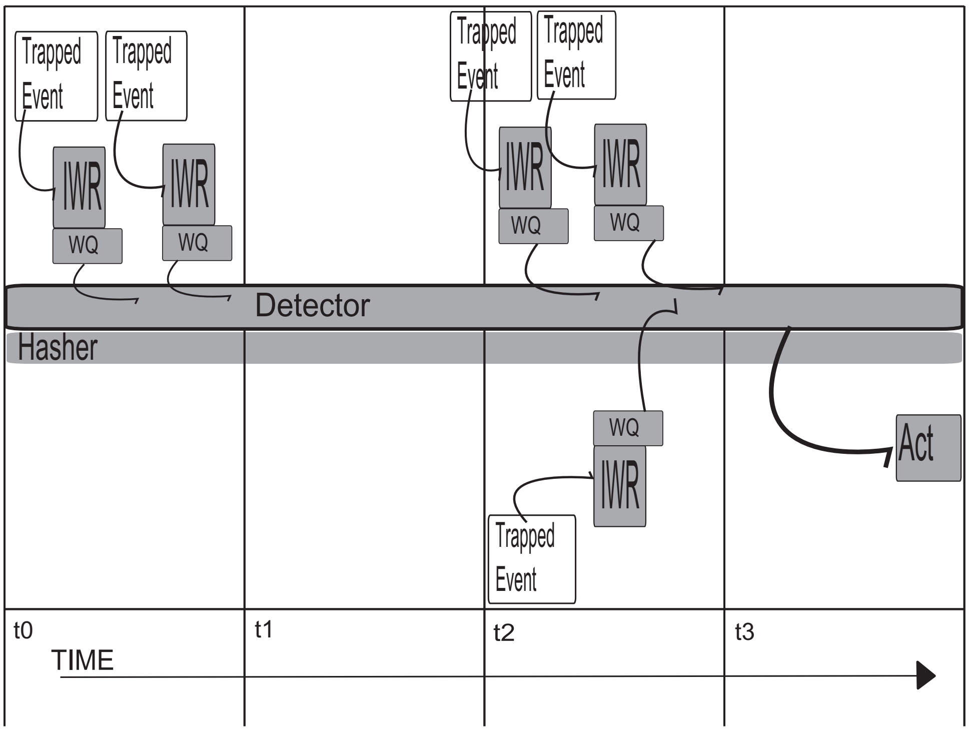

Fig. 3. SWADR monitoring workflow: Interceptor and Warning Recorder (IWR), Warning Pool (WP), Actuator (Act) interactions over time.

users (SU). For this purpose, the SP hires resources from the cloud provider (CP). It is worth noticing that the SU and the SP do not have any physical control over cloud machines, whose status cannot be observed. The SU and the CP enter into a Service Level Agreement that describes how the cloud is going to run service SI.

Possible attacks against cloud systems can be classified as follows (see also Smith et al., 2006):

CAT1	Resource attacks against CPs;
CAT2	Resource attacks against SPs;
CAT3	Data attacks against CPs;
CAT4	Data attacks against SPs;
CAT5	Data attacks against SUs.

Resource attacks (CAT1-CAT2) regard the misuse of resources, such as stealing virtual resources to mount a large scale botnet attack. Data attacks (CAT3-CAT4) steal or modify service or node configuration data (that can be used later to perform an attack). Data attacks against service users (CAT5) can lead to leakage of sensitive data. CAT1 and CAT3 attack classes involve an attack to cloud infrastructure components. Virtualization technologies underlying cloud computing infrastructure can pose security challenges themselves (Secunia, 2009). In addition, cloud computing middleware potentially allows some novel attacks that have not been identified yet. We will later see how ACPS deals with such threats.

5. Advanced cloud protection system

The proposed Advanced Cloud Protection System (ACPS) is intended to actively protect the integrity of the guest VMs and of the distributed computing middleware by allowing the host to monitor guest virtual machines and infrastructure components. Our proposal extends the KvmSec (Lombardi and Di Pietro, 2009) approach to protect monitored components against intruders and attacks such as worms and viruses.

ACPS is a purely host side architecture leveraging virtual introspection (Hay and Nance, 2008). This allows: to deploy any guest virtual appliance "as it is"; to enforce some form of accountability on guest activity without being noticed by an attacker located on the guest. This latter feature is provided being the protection system hard to detect, as it is immune to timing analysis attacks—it is completely asynchronous. In the following we describe the ACPS threat model and requirements for different distributed computing platforms. We then give implementation details as well as an evaluation of the ACPS effectiveness and performance, having provided an implementation of the designed cloud computing protection architecture.

5.1. *Threat model*

In our model we can rely on host integrity, since we assume the host to be part of the Trusted Computing Base (TCB) (Hohmuth et al., 2004). When the VM image is provided by a trusted entity, guest integrity is assumed at setup time but it is subject to threats as soon as the VM is deployed and exposed to the network. Indeed, guests can be the target of possible kinds of cyber attacks and intrusions such as viruses, code injection, and buffer overflow to cite a few. In case the guest image is provided by the user, VM trustfulness cannot be guaranteed and guest actions have to be monitored to trace possibly malicious activities. In our model, attackers can be cloud users (SP) or cloud applications users (SU), whereas victims can be the providers running services in the cloud (CAT2-CAT4), the cloud infrastructure itself (CAT1-CAT3) or other users (CAT5). A traditional threat is when an attacker attempts to perform remote exploitation of software vulnerabilities in the guest system (CAT2). Some attacks are made possible by exploiting cloud services (CAT1-CAT2), since a malicious party can legally hire other instances within the cloud and, as previously highlighted, it can manage to learn confidential information (CAT5). Other attacks are also possible such as Denial of Service

(CAT1-CAT2), estimating traffic rates, and keystroke timing (CAT2-CAT5) (see Ristenpart, 2009).

5.2. Requirements

We identified the core set of requirements to be met by a security monitoring system for clouds are the following (see also Lombardi and Di Pietro, 2009; Lombardi and Di Pietro, 2010):

REQ1 Effectiveness: the system should be able to detect most kinds of attacks and integrity violations.
REQ2 Precision: the system should be able to (ideally) avoid false-positives; that is, mistakenly detecting malware attacks where authorized activities are taking place.
REQ3 Transparency: the system should minimize visibility from VMs; that is: SP, SU, and potential intruders should not be able to detect the presence of the monitoring system.
REQ4 Non-subvertability: the host system, cloud infrastructure and the sibling VMs should be protected from attacks proceeding from a compromised guest and it should not be possible to disable or alter the monitoring system itself.
REQ5 Deployability: the system should be deployable on the vast majority of available cloud middleware and HW/SW configurations.
REQ6 Dynamic Reaction: the system should detect an intrusion attempt over a cloud component and, if required by the security policy, it should take appropriate actions against the attempt and against the compromised guest and/or notify remote middleware security-management components.
REQ7 Accountability: the system should not interfere with cloud and cloud application actions, but collect data and snapshots to enforce accountability policies.

There is a trade-off between transparency and dynamic reaction; we solved this problem by letting the set of possible ACPS reactions be a subset of regular guest maintenance capabilities, e.g. halting the guest, restarting it from a fresh image, and migrating the VM instance. The above actions are, from the point of view of the SU or SP, virtually indistinguishable from regular load-balance based VM operations.

5.3. Proposed approach

We monitor key components that would be targeted or affected by attacks in order to protect the VMs and the cloud infrastructure. By either actively or passively monitoring key kernel and middleware components we are able to detect any possible modification to kernel data and code, thus guaranteeing that kernel and middleware integrity have not been compromised. Furthermore, in order to monitor cloud entry points, we check behavior and integrity of cloud components via logging and periodic checksum verification of executable files and libraries. A further objective we want to achieve, especially when the guest image is not trusted by the cloud provider, is ensuring that an attacker-run application cannot detect that an external intrusion detection system is in place. Note that, as for introspection techniques, it is still not clear to what extent they can be detected by the target virtual machine. In fact, the presence of a monitoring system can potentially be detected through measurement of the time it takes for certain function calls to execute. Leveraging this observation, our monitoring system acts in a way that we define *SWADR*—synchronous warning—asynchronous detection and response. In particular, ACPS can provide protection:

PRT1 from attacks coming from outside the cloud;
PRT2 from attacks coming from sibling VMs;
PRT3 from attacks coming from VMs.

The high level description of ACPS combined with Eucalyptus, and ACPS combined with OpenECP architectures, are shown in Figs. 4 and 5, respectively, where potentially dangerous data flows are depicted in continuous lines and monitoring data flows are depicted in dashed lines. All ACPS modules are located on the Host. ACPS makes use of *Qemu* (Bellard, 2005) to access the guest. Suspicious guest activities (e.g. system_call invocation) can be noticed by the *Interceptor* and recorded by the *Warning Recorder* into the *Warning Pool*, where the potential threat will be evaluated by the *Evaluator* component. The *Interceptor* has been conceived not to block or deny any system call, in order to prevent the monitoring system from being detected: in *SWADR* mode, the timing attack is neutralized. Indeed, the evaluation components (*Evaluator* and *Hasher*) are always active—see Fig. 3—running and continuously performing security checks. In fact, the *Evaluator* and the *Hasher* are active and running even when the *Warning Pool* is empty. In this case, the purpose of the *Warning Pool* is mainly to cache warnings in order not to choke the evaluation component. The *Warning Pool* also allows setting priorities with respect to the order of evaluation. This guarantees increased invisibility, even though a large number of warnings might potentially delay decision and reaction by the *Actuator*. With respect to such an issue, an increasing rate of incoming warnings could be treated as a security threat on its own. It is true that, the *SWADR* asynchronous, non-blocking approach can potentially allow the attacker to perform some—limited in time—tampering with the target system. It is also true that in order to perform modifications to the guest system, the attacker must have already taken control of such system. Furthermore, the undetectability of the monitoring system allows malware behavior to be observed in a honeypot-fashion.

ACPS enjoys the following features: it is transparent to guest machines (even malicious or untrusted ones); it supports full virtualization (Perez et al., 2008), which renders the system less detectable on guest side; and, it can be deployed on most x86 and x86_64-based distributed cloud computing platforms.

ACPS is significantly different from the security monitoring systems presented in *KvmSec* (Lombardi and Di Pietro, 2009) and *KvmSma* (Lombardi and Di Pietro, 2010) or sketched in *TCPS* (Lombardi and Di Pietro, 2010); some of the main differences between *KvmSec*, KvmSma, TCPS, and ACPS are shown in Table 1. Most important, ACPS is completely transparent to guest machines, features SWADR mode (see REQ3), Warning Pools and enables hot recovery by replacement of a compromised service as well as resuming execution from the lates secure snapshot (see REQ6). ACPS is difficult to compromise even from an (already) compromised or untrusted virtual machine (see CAT1 and CAT3), while it can transparently inspect and analyze data inside guests. ACPS supports accountability (see REQ7), as discussed later in this section, and allows tracing and recording of guest status and data via snapshots, thus supporting forensics analysis. Furthermore, it has been fully integrated within existing cloud middleware. ACPS, like TCPS, is entirely located on the host machine (see REQ3). In ACPS each Virtual Machine uses its own private memory area, so it is totally independent from other VMs (see REQ4 and CAT4).

In ACPS, the host-side database *Checksum DB* contains computed checksums for selected critical host infrastructure and guest kernel code, data, and files. The runtime *Warning*

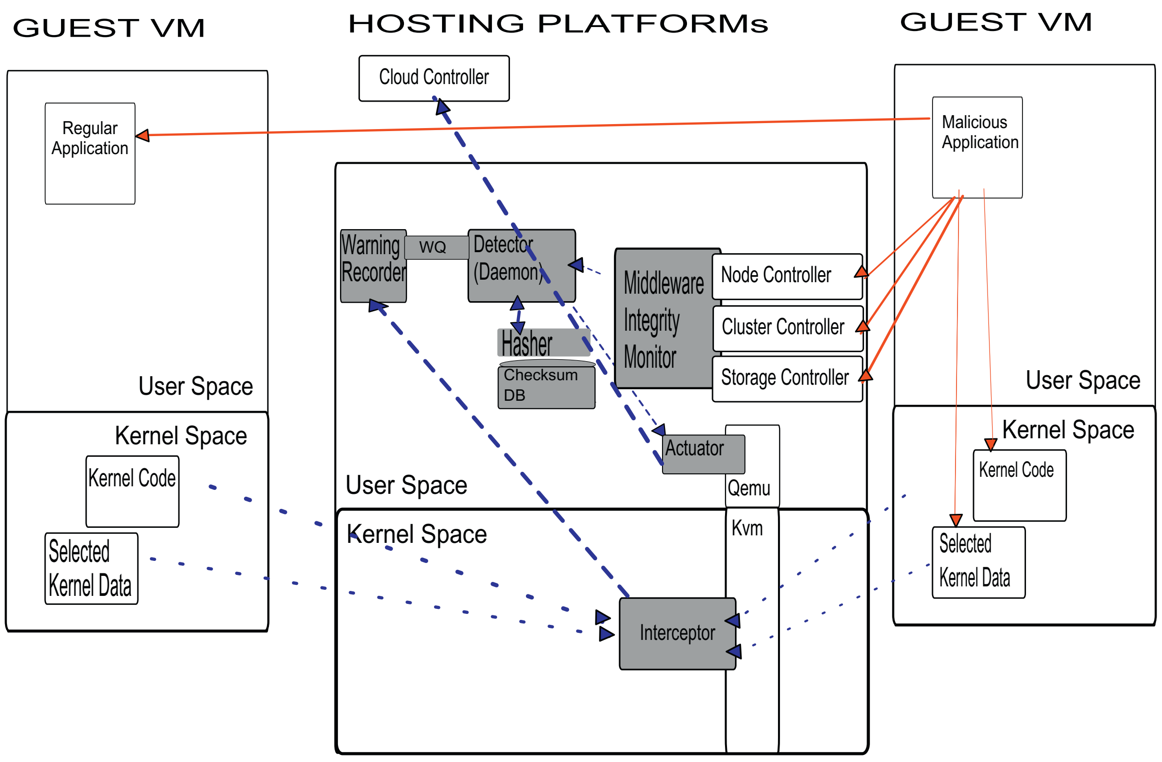

Fig. 4. ACPS (components in gray) combined with Eucalyptus—Architecture.

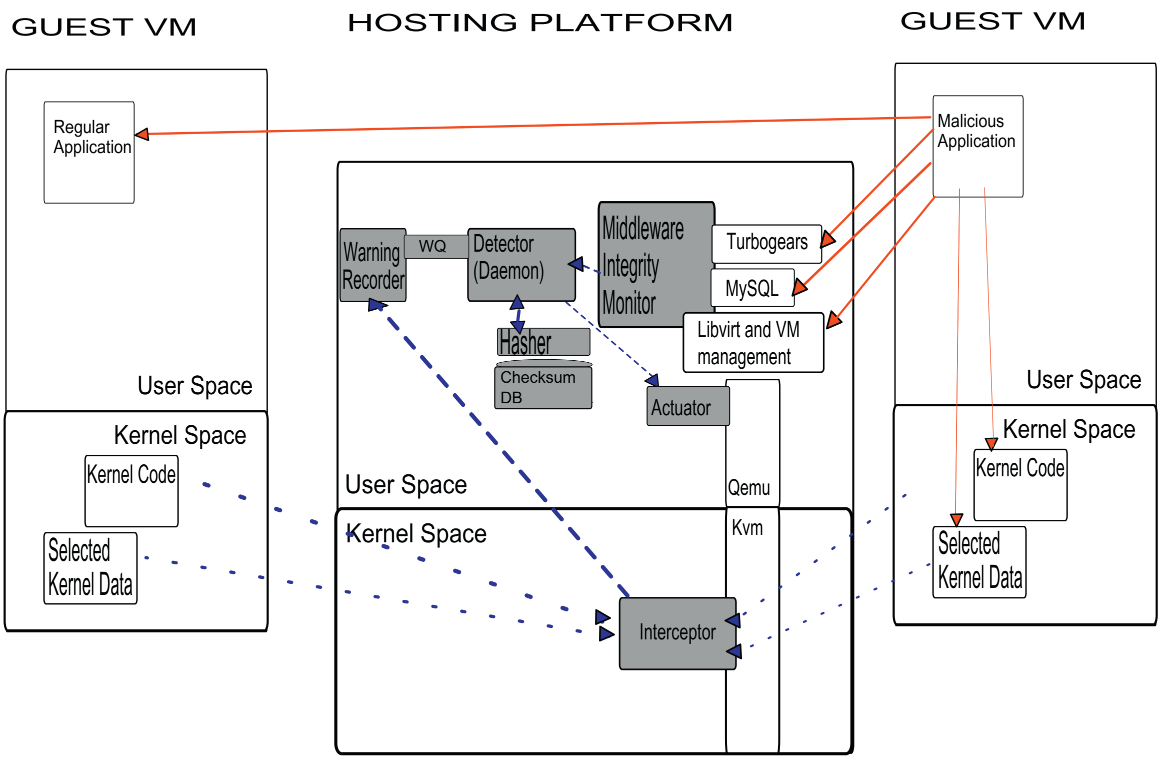

Fig. 5. ACPS (components in gray) combined with OpenECP—Architecture.

Recorder daemon can asynchronously recompute hash values for such monitored objects and can file warnings towards the *Evaluator*. The *Evaluator* daemon examines such warnings and evaluates (see REQ1-REQ2) whether the security of the system has been endangered. In such a case the *Actuator* daemon is invoked to act according to a specified security policy (REQ6). Consequently, ACPS can locally react to security breaches or notify the security management layer for such components of the occurred events. ACPS can also replace a compromised server on-the-fly by restoring that VM from a clean backup image (see Distler et al., 2008). To avoid false positives as much as possible (REQ2), an administrator or the Cloud Controller component can notify ACPS of the new components' checksums. ACPS is integrated in the virtualization software and leverages hardware virtualization support to monitor the integrity of the guest and middleware components by performing a checksum of such objects. It is worth noting that no system_call is ever blocked or delayed by ACPS to check for permission violation and the kind of reaction our monitoring system can perform (freezing/halting/restarting the guest VM) is virtually indistinguishable from normal system maintenance tasks. Furthermore, the *Interceptor* and *Warning Recorder* can trace events, actions and the actors who performed them. These data, combined with *Checksum DB* data, can be used for accountability purposes (REQ7) (Haeberlen, 2009). This provides the proposed architecture with the necessary support to implement external secure event logging and accountability tools.

6. Implementation

We implemented ACPS over Eucalyptus and OpenECP (REQ5). Eucalyptus high-level system components are implemented as webservices. Eucalyptus (Nurmi et al., 2009) is composed of: a Node Controller (NC) that controls the execution, inspection, and termination of VM instances on the host where it runs; a Cluster Controller (CC) that gathers information about VM and schedules VM execution on specific node controllers; further, it manages virtual instance networks; a Storage Controller (SC)—Walrus—that is, a storage service providing a mechanism for storing and accessing VM images and user data; a Cloud Controller (CLC), the webservices entry point for users and administrators that makes high level scheduling decisions.

A more detailed description of how ACPS integrates with the Eucalyptus component is reported in Fig. 6. On Eucalyptus, ACPS can be deployed with the Cloud Controller, the Cluster Controller and, most importantly, the Node Controller. The NC runs on every node hosting VM instances. We especially monitor NC activity and integrity, since this is the key component for this cloud implementation (Rellermeyer et al., 2009). In fact, as shown in Fig. 6, in case an attack or a potentially dangerous alteration is detected, the *actuator* can change the NC, Libvirt, and Iptables configuration in order to prevent further damages. The possible reactions include migrating the guests that did not raise any warning (clean guests) to other hosts, while disabling the suspicious host node itself.

OpenECP, like its proprietary Enomalism (Enomaly, 2009) sibling, provisions and manages resources by leveraging Turbogears, Python, and the Libvirt library (RedHat, 2007). These are the additional infrastructure resources we need to monitor the integrity of. The components that have to be monitored are

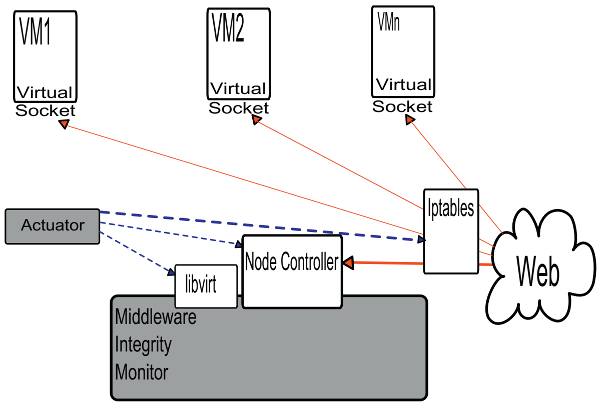

Fig. 6. ACPS (components in gray) combined with Eucalyptus—detail.

Python, Libvirt and Mysql processes, executable files and libraries, as well as configuration files. Turbogears front-end components need to be especially monitored, since they are particularly exposed to the network. Such monitoring provides integrity protection for both front-end and back-end systems (against CAT1). Enomalism integration details are shown in Fig. 7. In particular, in case an attack or a potentially dangerous alteration is detected, the *actuator* can change the Mysql, Turbogears, Libvirt and Iptables configuration in order to prevent further damages. The possible reactions include filtering out selected web requests, migrating clean guests to other hosts and disabling the suspicious host node itself.

7. Effectiveness—ACPS under attack

In this section we show how our proposal copes with attacks the cloud can be subject to in real environments. In particular, we report on the practical experiments performed to assess the resilience of the proposed architecture and also provide discussion on how the key requirements set in previous sections are met by our proposal.

The detection capabilities (see Table 3) of our system are assessed against known attack techniques (see Table 2). However, since source code for many attacks is not publicly available, we performed our test by simulating the attack steps.

As shown earlier, we can partition attacks into 5 categories, ranging from CAT1 to CAT5. ACPS has been proven to detect and to react to attacks belonging to the above mentioned categories, in a way that is summarized in Table 3. In particular, we took from the current literature some relevant attacks that actual networked architectures can be subject to (Huang et al., 2007; Ristenpart, 2009; Costa et al., 2005) and we showed the degree of added protection provided by ACPS to guests and VMs when the system is exposed to such attacks.

In particular, we simulated an attack of type CAT1 by exploiting host service vulnerabilities (see Debian ssh Debian,

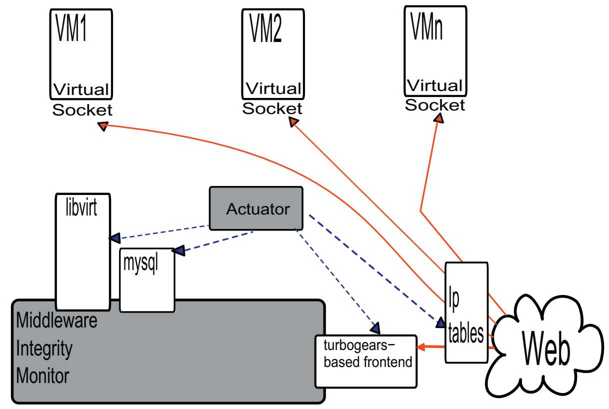

Fig. 7. ACPS (components in gray) combined with OpenECP—detail.

Table 2
Attacks instantiation.

Category	Implemented Attack
CAT1	Apache vuln. (Eucalyptus)/ssh Python vuln. (OpenECP)
CAT2	Sebek rootkit
CAT3	network probing
CAT4	colocation, detection
CAT5	colocation, keystroke timing

Table 3
ACPS Detection/Reaction capabilities.

Attack Technique	Detection reason	Implemented Reaction
Apache/Python/ssh	process footprint	service migration/restart
Sebek	altered sys_call table	clean VM restart
Process Hiding	tasklist navigation	clean VM restart
colocation, network probing	Iptables monitoring	Silently filter/drop network packets

2008 and *Apache* vulnerabilities CVE, 2008). In this case ACPS monitors the Apache process behavior and memory footprint and notices the abnormal memory usage and connection attempts. Once the attack is detected, ACPS restarts the compromised service from a verified executable and re-establishes its configuration files.

We implemented an attack of type CAT2 by inserting a Sebek rootkit (Honeynet Project. Sebek, 2003) in a guest VM. Sebek is a kernel module that hides its presence and intercepts filesystem and network activity. It does so by altering the syscall table in order to change the execution flow and to execute malicious code. Here ACPS detects both the alteration of the syscall table and the change in the checksum of kernel files on virtual storage.

We also implemented an attack of type CAT4 using a kernel data attack as described in (Rhee et al., 2009). In particular, given that network cards are emulated by the underlying *Qemu* software, guests are protected by ACPS from the Lying Network Card attack approach. In this context, we implemented the process hiding approach that allows the attacker to run tasks without having them appearing in the list of processes. This attack has been accomplished using a dynamic data attack leveraging /dev/kmem to manipulate the task list structure. ACPS detects the alteration when, by navigating the kernel scheduler task list, it discovers that additional hidden structures are present. As a reaction, ACPS restarts the guest from a clean VM disk image.

Finally, we implemented attacks of type CAT3 and CAT5 by using the techniques cited by Ristenpart in Ristenpart (2009). First of all, both external (outside the cloud) and internal (from sibling VMs) network probing via port scanning is intercepted by Iptables rules that raise an alarm to the Warning Recorder (WR). As regards keystroke timing (Ristenpart, 2009), given that the attacker resorts to co-residence load measurements to analyze the time between keystrokes and collect sensitive information, ACPS renders such attack less feasible. Indeed, having ACPS running on the CPU under attack makes times measurement much harder for the attacker.

7.1. Anatomy of attack and reaction

In the following, we describe the details of a sample attack we performed and the reaction we obtained from ACPS (host and guest systems' integrity is assumed granted at time t_0):

1. the attacker (ATT) exploits a ssh vulnerability (Debian, 2008) or a weak password to get access to an account;
2. ACPS *Iptables* logs to the Warning Recorder (WR), the number and targets of ssh connection attempts.
3. ATT then performs a symbolic link privilege escalation attack (Johnston, 2009) to gain root privileges;
4. ATT patches critical kernel syscall code;
5. ACPS *Interceptor* notices the operation on the kernel object and files and records such potentially dangerous operations in the WR;
6. ACPS *Evaluator* fetches warnings issued by the *Warning Recorder* and checks for the integrity of affected parts by comparing checksums;
7. when the alteration is detected an alert is issued to the remote security-management component; furthermore the VM is stopped and re-initiated (see REQ6).

The ACPS *Evaluator* can be configured to react (e.g. launching a service restart) when the desired number of attack clues have been collected. The increased security provided by this approach must be balanced by the possible increase of both service downtime (DoS) and computing resources usage—that arise in case of false positives.

7.2. Performance

In this section we present the results of the experiments aimed at evaluating ACPS performance when implemented over current cloud solutions. The hardware/software configuration adopted for such tests is depicted in Table 4. The guest operating systems were ×86 Centos 5.2 leveraging 1 virtual CPU and 1 GB RAM. Hardware virtualization was enabled on the hosts. Guests run 32-bit OSes whereas hosts run 64-bit OSes. Guest virtual disk made use of an image file on the hosts.

We tested the performance of our solution under three different types of workload:

1. CPU-intensive;
2. Mixed workload;
3. I/O intensive.

In detail, we provisioned an Eucalyptus and an OpenECP guest and measured the time it takes such Eucalyptus and OpenECP guests to perform three different kinds of operations: mp3 encoding of a wav file (CPU-intensive); vanilla 2.6.30 Linux kernel compilation (mixed workload); and, *dd* of a large file (1 GB) to a disk partition (I/O intensive).

Results are reported in Fig. 8 where bars represent execution times normalized with respect to the same test executed on a regular Kvm guest machine on the same hosts. Values are averaged over the tested CPUs and show that the overhead introduced by ACPS is quite small. There is a small performance loss due to the additional integrity checks ACPS performs on cloud middleware. The differences between Enomalism and Eucalyptus results can be explained by the difference in the number and complexity of the components of the two. This benchmarks helped us to quantify the actual real-world application overhead introduced by the additional asynchronous monitoring components. For this purpose, bars have to be compared pairwise, the left bar representing performance without ACPS, whereas the right one represents performance with ACPS active. Indeed, the impact of ACPS on the performance of current cloud solutions is quite limited, given that the maximum performance loss is under 6%. In particular, for the CPU-intensive test it can be as low as 3%.

Table 4
Host test environment.

Feature	host A	host B
CPU Model	Athlon 64 4400+	Turion 64 RM-72
Cores	2	2
Ram	4096	4096
Host OS	Ubuntu 8.10 (O.ECP)	Ubuntu 8.10 (O.ECP)
	Ubuntu 9.10 (Eucal.)	Ubuntu 9.10 (Eucal.)
Kernel	Linux 2.6.30	Linux 2.6.30
VMM	Kvm 88	Kvm 88

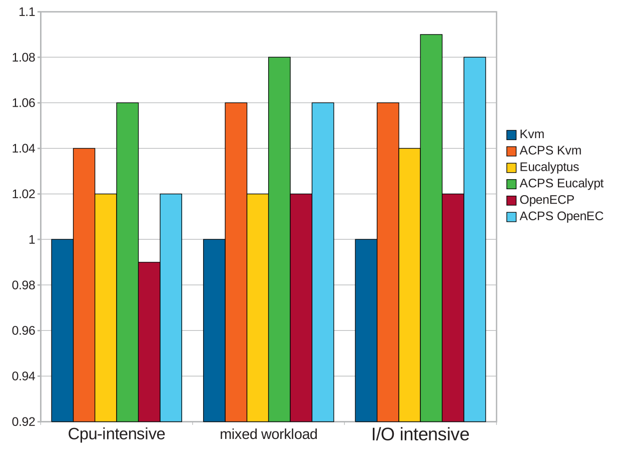

Fig. 8. ACPS execution times (normalized w.r.t. Kvm)—first test round.

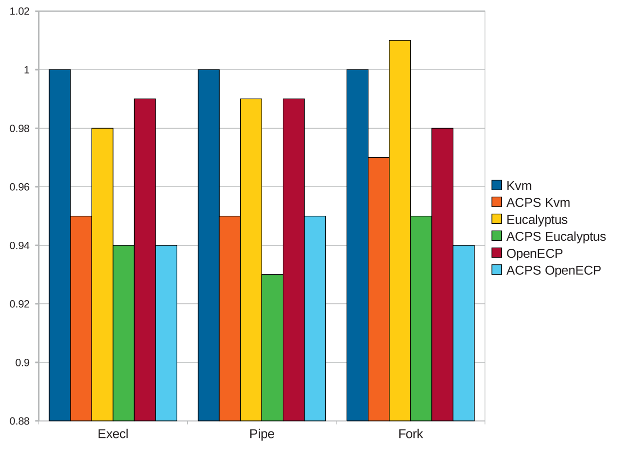

Fig. 9. ACPS performance comparison (normalized w.r.t. Kvm)—second test round.

This result is not surprising since in the *SWADR* approach the evaluation is run as a low priority process and it is spread over time, thus leaving the CPU resources free for the most part. A slightly more complex result is obtained when looking at the mixed workload and the I/O intensive workload results. This is probably due to the increased number of active ACPS interactions with filesystem activities. However, the impact of ACPS on the performance of these types of workload never exceeds 6% and, on average, provided results are quite interesting.

We then performed a further series of tests to collect more detailed performance measurements, with a special interest in I/O subsystems. In particular, the following selected tests from the well-known Unixbench (Smith et al., year) test suite were executed:

1. Execl: this test measures the number of *execl*() function calls that can be performed in one second. Execl is aimed at replacing the current process image with the new one; hence, this operation stresses memory I/O performance.
2. Pipe: this test measures the number of pipe-writes (512 bytes) a process successfully performs in one second. It is an indication of how fast the process is in performing I/O activities.
3. Fork: this test measures the number of times a *fork*() call can be invoked per unit of time. This test is an important indicator of overall performance.

Results are reported in Fig. 9. As above, the comparison has to be carried out pairwise with respect to columns. Bars represent the number of executed operations, hence a higher bar means a better performance. This benchmarks helped us to quantify the specific-operation performance overhead due to ACPS. The good news is that performance loss due to the additional integrity checks ACPS performs is less than 6% in any test. More in detail, in this case the performance for the two cloud computing environments is very similar, as expected, to the plain KVM guest. The reason is that the cloud computing infrastructure only indirectly affects the execution of such low-level operations, whose execution depends on operating system configuration and security-related checks (even though these latter ones cannot be distinguished from normal workload). ACPS activity mostly affects I/O performance (see Pipe throughput results affected by up to 6% performance loss) whereas the fork experiment performance loss is less than 4%. Such a difference can be due to the interaction with the ACPS interception components.

Results show that there is a margin of improvement for the I/O monitoring operations. This margin of improvement can be explained by the fact that the implemented I/O monitoring is not fully mature. Indeed, it requires extra interaction with the I/O subsystem that could be reduced in future implementations of the ACPS framework. Overall, these first results are interesting—due to the generally low overhead introduced—, and encourage us to further investigation aimed at leveraging the improvement margin previously highlighted. Finally, it is worth noticing that even though overall performance is degraded by the monitoring system itself, such performance penalty cannot be distinguished by the attacker from regular CPU load, since the system_call timing difference between protected and unprotected configurations is constantly within the 3%–6% range, which is virtually indistinguishable from the performance loss due to regular task operations.

8. Conclusion

In this paper, we have provided several contributions to secure clouds via virtualization. First, we have proposed a novel advanced architecture (ACPS) for cloud protection that can monitor both guest and middleware integrity and protect them from most kinds of attack while remaining fully transparent to the service user and to the service provider; ACPS has been tailored and deployed onto different cloud implementations and has been proven able to locally react to security breaches and capable of notifying the security management layer of such an occurrence. Second, the proposed architecture has been implemented entirely on current open source solutions and both protection effectiveness and performance results have been collected and analyzed. Results show that the proposed approach is effective and introduces just a small performance penalty.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments that helped improving the quality of the paper.

This work was partly supported by the Spanish Ministry of Education through Projects TSI2007-65406-C03-01 ‘‘E-AEGIS’’ and CONSOLIDER CSD2007-00004.

‘‘ARES’’, and by the Government of Catalonia under grant 2009 SGR 1135.

References

AIDEteam. Advanced intrusion detection environment. 〈http://sourceforge.net/projects/aide〉, November 2005.

Armbrust M, Fox A, Griffith R. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

Bellard F. Qemu, a fast and portable dynamic translator. In ATEC '05: Proceedings of the annual conference on USENIX annual technical conference, Berkeley, CA, USA, 2005. USENIX Association, p. 41.

Bethencourt J, Song D, Waters B. New techniques for private stream searching. ACM Trans. Inf. Syst. Secur. 2009;12(3):1–32.

Cachin C, Keidar I, Shraer A. Trusting the cloud. SIGACT News 2009;40(2):81–6.

Costa M, Crowcroft J, Castro M, Rowstron A, Zhou L, Zhang L, Barham P. Vigilante: end-to-end containment of internet worms. SIGOPS Oper. Syst. Rev. 2005;39(5):133–47.

CVE. Common vulnerabilities and exposures-2008-2364. 〈http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2364〉, 2008.

Debian. Dsa-1571-1 Openssl: predictable random number generator. 〈http://www.debian.org/security/2008/dsa-1571〉, 2008.

Distler R, Kapitza R, Reiser HP. Efficient state transfer for hypervisor-based proactive recovery. In WRAITS '08: Proceedings of the 2nd workshop on recent advances on intrusiton-tolerant systems. ACM, New York, NY, USA, 2008. pp. 1–6.

Enisa. Cloud computing risk assessment. 〈http://www.enisa.europa.eu/act/rm/files/deliverables〉, 2009.

Enomaly. Enomalism. 〈http://www.enomaly.com〉, 2009.

Foster T, Zhao Y, Raicu I, Lu S. Cloud computing resource management through a grid middleware: A case study with diet and eucalyptus. Cloud Computing, IEEE International Conference on, 2009. pp. 151–4.

Gu L, Cheung S-C. Constructing and testing privacy-aware services in a cloud computing environment: challenges and opportunities. In Internetware '09: Proceedings of the first Asia-Pacific symposium on internetware. ACM New York, NY, USA, 2009. pp. 1–10.

Haeberlen A. A case for the accountable cloud. In LADIS '09: 3rd ACM SIGOPS International workshop on large scale distributed systems and middleware, 2009.

Hay B, Nance K. Forensics examination of volatile system data using virtual introspection. SIGOPS Oper. Syst. Rev. 2008;42(3):74–82.

Hohmuth M, Peter M, Härtig H, Shapiro JS. Reducing tcb size by using untrusted components: small kernels versus virtual-machine monitors. In EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop, ACM, New York, NY, USA, 2004. p. 22.

Honeynet Project. Sebek. 〈https://projects.honeynet.org/sebek/〉, 2003.

Huang Y, Arsenault D, Sood A. Closing cluster attack windows through server redundancy and rotations. In CCGRID, 2006. p. 21.

Huang Y, Geng X, Whinston AB. Defeating DDoS attacks by fixing the incentive chain. ACM Trans. Internet Technol. 2007;7(1):5.

Jiang X, Wang X, Xu D. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In CCS '07: Proceedings of the 14th ACM conference on computer and communications security. ACM, New York, NY, USA, 2007. pp. 128–38.

Sam Johnston. Cve-2008-4990 enomaly ecp/enomalism: Insecure temporary file creation vulnerabilities. 〈http://www.securityfocus.com/archive/1/archive/1/500573/100/0/threaded〉, 2009.

Kim GH, Spafford EH.. The design and implementation of tripwire: a file system integrity checker. In CCS '94: Proceedings of the 2nd ACM conference on computer and communications security. ACM, New York, NY, USA, 1994. pp. 18–29.

Lenk A, Klems M, Nimis J, Tai S, Sandholm T. What's inside the Cloud? An architectural map of the cloud landscape. In it CLOUD '09: Proceedings of the 2009 ICSE workshop on software engineering challenges of cloud computing, IEEE Computer Society, Washington, DC, USA, 2009. pp. 23–31.

Lombardi F, Di Pietro R. Kvmsec: a security extension for linux kernel virtual machines. In SAC '09: Proceedings of the 2009 ACM symposium on applied computing, ACM, New York, NY, USA, 2009. pp. 2029–34.

Lombardi F, Di Pietro R. A security management architecture for the protection of kernel virtual machines. In TSP '10: Proceedings of the Third IEEE international symposium on trust, security and privacy for emerging applications (to appear), IEEE Computer Society, Washington, DC, USA, 2010.

Lombardi F, Di Pietro R. Transparent security for cloud. In SAC '10: Proceedings of the 2010 ACM symposium on applied computing (poster paper, to appear), ACM, New York, NY, USA, 2010.

Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. The Eucalyptus open-source cloud-computing system. In CCGRID '09: Proceedings of the 2009 9th IEEE/ACM international symposium on cluster computing and the grid, IEEE Computer Society, Washington, DC, USA, 2009. pp. 124–31.

Openecp. Openecp. 〈http://www.openecp.org〉, 2010.

Payne BD, Carbone M, Sharif M, Lee W. Lares: An architecture for secure active monitoring using virtualization. In SP '08: Proceedings of the 2008 IEEE symposium on security and privacy (sp 2008), IEEE Computer Society, Washington, DC, USA, 2008. pp. 233–47.

Pearson S. Taking account of privacy when designing cloud computing services. In CLOUD '09: Proceedings of the 2009 ICSE workshop on software engineering challenges of cloud computing, IEEE Computer Society, Washington, DC, USA, 2009. pp. 44–52.

Perez R, van Doorn L, Sailer R. Virtualization and hardware-based security. IEEE Security and Privacy 2008;6(5):24–31.

Peter M, Schild H, Lackorzynski A, Warg A. Virtual machines jailed: virtualization in systems with small trusted computing bases. In VDTS '09: Proceedings of the 1st EuroSys Workshop on virtualization technology for dependable systems, ACM, New York, NY, USA, 2009. p. 18–23.

Pollitt M, Nance K, Hay B, Dodge RC, Craiger P, Burke P, Marberry C, Brubaker B. Virtualization and digital forensics: a research and education agenda. J. Digit. Forensic Pract. 2008;2(2):62–73.

Qumranet. Linux kernel virtual machine. 〈http://kvm.qumranet.com〉.

Ranadive A, Gavrilovska A, Schwan K. Ibmon: monitoring vmm-bypass capable infiniband devices using memory introspection. In HPCVirt '09: Proceedings of the 3rd ACM workshop on system-level virtualization for high performance computing, ACM, New York, NY, USA, 2009. p. 25–32.

RedHat. Libvirt. 〈http://libvirt.org〉, 2007.

Rellermeyer JS, Duller M, Alonso G. Engineering the cloud from software modules. In CLOUD '09: Proceedings of the 2009 ICSE workshop on software engineering challenges of cloud computing, IEEE Computer Society, Washington, DC, USA, 2009. p. 32–7.

Rhee J, Riley R, Xu D, Jiang X. Defeating dynamic data kernel rootkit attacks via vmm-based guest-transparent monitoring. Availability, Reliability and Security, 2009. ARES '09. International Conference on, 2009.

Riley R, Jiang X, Xu D. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. In RAID '08: Proceedings of the 11th international symposium on recent advances in intrusion detection, Springer-Verlag, Berlin, Heidelberg, 2008. p. 1–20.

Ristenpart T, Tromert E, Shacham H, Savage S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In CCS '09: Proceedings of the 14th ACM conference on computer and communications security, ACM, New York, NY, USA, 2009. p. 103–15.

Salza S, DiCarlo Y, Lombardi F, Puccinelli R. Leveraging the grid for the autonomic management of complex infrastructures. In: GCA grid computing and applications conference proceedings, 2006. p. 32–7.

Secunia. Secunia advisory. 〈http://secunia.com/advisories/36389〉, 2009.

Seshadri A, Luk M, Qu N, Perrig A. Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, ACM, New York, NY, USA, 2007. p. 335–50.

Siebenlist F. Challenges and opportunities for virtualized security in the clouds. In SACMAT '09: Proceedings of the 14th ACM symposium on access control models and technologies, ACM, New York, NY, USA, 2009. p. 1–2.

Smith B, Grehan R, Yager T. Byte-unixbench: A Unix benchmark suite. 〈http://code.google.com/p/byte-unixbench/〉.

Smith M, Friese T, Engel M, Freisleben B. Countering security threats in service-oriented on-demand grid computing using sandboxing and trusted computing techniques. J. Parallel Distrib. Comput. 2006;66(9):1189–204.

Sousa P, Bessani AN, Correia M, Neves NF, Verissimo P. Resilient intrusion tolerance through proactive and reactive recovery. Pacific Rim International Symposium on Dependable Computing, IEEE 2007;0:373–80.

Vaquero LM, Rodero-Merino L, Caceres J, Lindner M. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 2009;39(1): 50–5.