2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing

Intrusion Detection in the Cloud

Sebastian Roschke, Feng Cheng, Christoph Meinel
Hasso Plattner Institute (HPI), University of Potsdam
P.O.Box 900460, 14440, Potsdam, Germany
{sebastian.roschke, feng.cheng, meinel}@hpi.uni-potsdam.de

***Abstract*—Intrusion Detection Systems (IDS) have been used widely to detect malicious behaviors in network communication and hosts. IDS management is an important capability for distributed IDS solutions, which makes it possible to integrate and handle different types of sensors or collect and synthesize alerts generated from multiple hosts located in the distributed environment. Facing new application scenarios in Cloud Computing, the IDS approaches yield several problems since the operator of the IDS should be the user, not the administrator of the Cloud infrastructure. Extensibility, efficient management, and compatibility to virtualization-based context need to be introduced into many existing IDS implementations. Additionally, the Cloud providers need to enable possibilities to deploy and configure IDS for the user. Within this paper, we summarize several requirements for deploying IDS in the Cloud and propose an extensible IDS architecture for being easily used in a distributed cloud infrastructure.**

Keywords-IDS, IDS Management, Cloud Computing, Virtualization, Virtual Machine

I. Introduction

Along with the proposal of the concept Cloud Computing, a new paradigm of software development and deployment has emerged. As described in [1], Cloud Computing can be interpreted as the sum of Software as a Service (SaaS) and Utility Computing. There are Cloud providers, which offer a specific virtualized infrastructure, and Cloud users, which use the provided services and infrastructure. Furthermore, there are three layers involved in Cloud Computing: the system layer, the platform layer, and the application layer. The hardware layer is the basis for Cloud computing and is not provided directly to the user. Therefore we do not consider it as a part of the Cloud itself. The system layer includes the virtual machine (VM) abstraction of a server and related virtual networks. The platform layer includes the infrastructure software, e.g., a virtualized operating system (OS) of a server or the runtime and the API of a specific programming language. Finally, the application layer includes other software running in the Cloud, such as a web application. An important property of Cloud computing is that the user has only partial control over the infrastructure being used, i.e., the user can only control the used services on the specific layer. Important examples for the different layers are: Google App Engine [4], Windows Azure [5], and Amazon EC2 [6]. As important technologies, Grid computing and virtualization are often applied in the area of Cloud Computing.

Intrusion Detection Systems (IDS) has been proposed for years as an efficient security measure and is nowadays widely deployed for securing critical IT-Infrastructures. Based on the protected objective, IDS can be classified into host-based intrusion detection systems (HIDS), network-based intrusion detection systems (NIDS), or distributed intrusion detection systems (DIDS), which contain both types of sensors. Due to different deployment mechanisms, IDS can be categorized as software-based IDS, hardware-based IDS, and VM-based IDS [15]. Thanks to some beneficial properties provided by the virtualization technology, such as strong isolation, fast recovery, and cross-platform, the newly emerged VM-based IDS implementations are usually more robust and portable. Nowadays, lots of commercial and open source intrusion detection systems implementations have emerged and been widely used in practice for identifying malicious behaviors against protected hosts or network environments. Some known examples of existing IDS solutions are F-Secure Linux Security [8], Samhain [9], and Snort [11]. An effective IDS should be capable of detecting different types of attacks as well as all the possible variants of a certain type of attack. IDS should detect not only known attacks but also unknown attacks. Furthermore, the IDS itself needs to be robust against any evasion techniques. To simultaneously provide multiple benefits from various IDS sensors, an integrated IDS solution is required [12]. The Intrusion Detection Message Exchange Format (IDMEF) [16] has been proposed as a standard to enable interoperability among different IDS approaches.

Although distributed IDS technology has been tested to be capable of working well in some large scale networks, the utilization and deployment in Cloud Computing is still a challenging task. The complex architecture of a Cloud infrastructure and the different kinds of users lead to different requirements and possibilities for being secured by IDS. An important issue for the user is the missing full control over the used infrastructure. Although some of the Cloud users require to separate the IDS and the monitored target, it is still necessary for the Cloud provider to provide such a possibility that certain end users can fully control at least the currently used resources. A user needs different types of sensors and various configuration possibilities of the rulesets and thresholds to efficiently monitor their virtualized components. The Cloud providers want to recognize attacks performed on virtualized components as well as direct attacks on the underlying infrastructure. To correlate and analyze alerts generated on multiple distributed sensors deployed in the Cloud, a central management unit is necessary. A high amount of security related events produced by the integrated sensors need to be synchronized, unified, and analyzed. Furthermore, the system needs to be compatible in deployment by supporting different

978-0-7695-3929-4/09 $26.00 © 2009 IEEE
DOI 10.1109/DASC.2009.94

environments, e.g., network-based or virtualized environments.

To meet these requirements, we propose and implement an extensible IDS management architecture in this paper, which consists of several sensors and a central management unit. Different types of sensors can be easily integrated into the extensible architecture. A new plugin concept of the *Event Gatherer* is realized as *Handler*, *Receiver*, and *Sender* to provide flexible integration of various sensors. The IDMEF standard is used to represent and exchange the alarm information. A standardized interface is designed to provide a unified view of result reports for users. By combining the system-level virtualization technology and some known VM Monitor (VMM) approaches, the new IDS management system can be used to handle most of VM-based IDSs. A prototype, which implements the proposed architecture, is presented in the paper as Proof-of-Concept. It includes sensor connectors for several well known IDS, such as Samhain, Snort, and F-Secure Linux Security. Additionally, different communication methods, including faster network-based communication and more reliable file-based communication, are realized based on IDMEF standard to support the storage and exchange of alert information within the management system. Information as VM status, VM workload, and IDS-VM assignments can be monitored and the involved IDS VMs can be stopped, started, and recovered by the management system. This provides basic capabilities to integrate the system in a virtualization context, such as in a Cloud architecture.

The rest of the paper is organized as follows. Section II describes a view on security in Cloud Computing and related work in the field of IDS management. In Section III, the deployment of IDS in Cloud computing is described. Section IV describes the proposed IDS management architecture. Section V lists some possible future works and section VI gives a short summary.

II. Security in Cloud Computing

A. Modeling the Cloud

As shown in Figure 1, the Cloud architecture can be divided in three layers: the system layer, the platform layer, and the application layer. The underlying hardware is not considered as a part of the Cloud, but as the fundamental basis. The system layer is the lowest layer in the Cloud architecture and includes virtualized hosts and networks. One example for this layer is the Amazon EC2 [6] service, which provides virtual hosts and networks to the customers. The possibility of buying hosts per hour provides IT companies a certain flexibility. The platform layer is the second layer in the architecture and includes virtualized operating systems as well as runtimes and APIs. A famous example is the Platform Windows Azure [5], which provides the user with several APIs for storage and management as well as an independent Common Language Runtime (CLR). The application layer is the top level of the architecture and provides virtual applications. Google App Engine[4] is a known infrastructure on this layer. A customer can write and upload web applications which are executed on the Google infrastructure and accessible via web browser. Currently, the programming languages Python and Java are supported. Furthermore, Google App Engine provides an API for storage functions and convenient programming.

As described in [1], there are Cloud providers and Cloud users in the Cloud infrastructure. Cloud providers offer a specific service in the Cloud and Cloud users consume this service. The Cloud provider is responsible for ensuring the quality of the offered service, i.e., the performance, the reliability, as well as basic security. The Cloud user consumes services from the providers and has to pay based on consumed resources. The Cloud user can also resell some of the consumed services to other users. Thus, it becomes a Cloud provider. Users and providers on the two upper layers are often called Software as a Service (SaaS) users/providers. Interaction between the different layers is possible, e.g., web application can use web services provided by a Windows Azure, which can be deployed on several virtual hosts in the Cloud. The architecture is expected to be highly inter-connected.

The underlying hardware is the basis for Cloud Computing. It is operated by the Cloud providers offering services on the system layer. It consists of many physical machines connected to the network. The Cloud providers need a certain kind of management on each layer to simplify the configuration of the Cloud infrastructure. The management components create the related virtual components on their specific layer.

B. Threats in the Cloud

As the Cloud combines many well known technologies and leads to complicated IT systems and networks, the occurrence of security problems is likely. Each layer of the architecture may suffer from certain vulnerabilities, introduced by different programming or configuration errors of the user or the provider.

Consider a web shop application deployed on Google App Engine. It consists of several product pages, a login area, and a simple function to comment and rate products. After loging into the web page and providing valid credentials, the user gets a session cookie to identify him in the session. The comment function of the web shop is vulnerable to Cross-Site-Scripting (XSS) [7] attacks. In case of web applications, XSS attacks are a common class of attacks. In a vulnerable application, an attacker can insert code (e.g. JavaScript) into the web page to steal user data, such as a session cookie used for authorization. This can be done by using the comment function for the products. As shown in Listing 1, the attacker can insert JavaScript into the comment of several products. By viewing the specific product page, each user can be forced to give away their session cookie which provides the attacker full access to the users' account.

Listing 1. XSS JavaScript

```
<SCRIPT type="text/javascript">
var adr = 'https://evil.example.com/ \
  grab.php?cookie=' \
  + escape(document.cookie);
</SCRIPT>
```

An example of the vulnerability on the platform layer can be constructed by means of the CLR. Consider an application

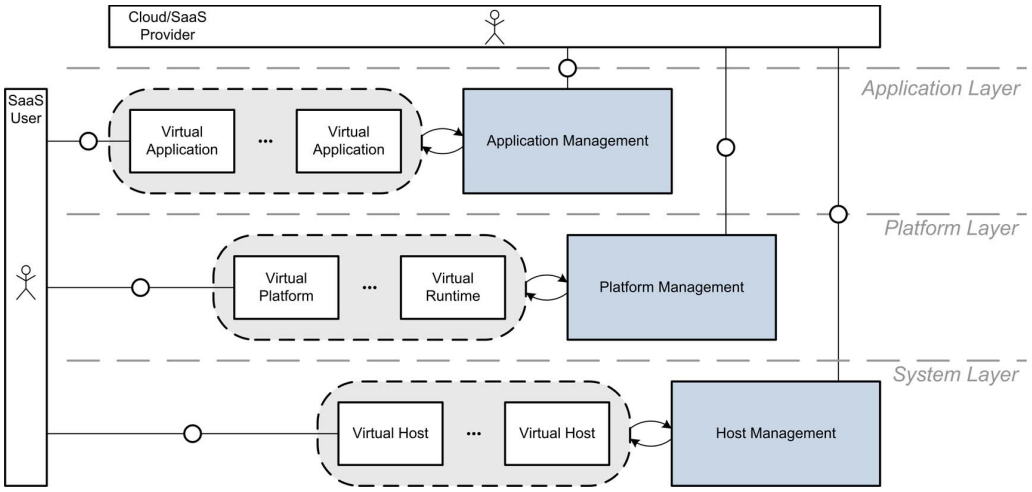

Fig. 1. Architecture of Cloud Computing

written with .Net which uses code in managed mode, i.e., it does memory management (allocating and deallocating memory) and can access pointers. Furthermore, this application can rely on a C-based network library which includes vulnerable code, as shown in Listing 2. This code is vulnerable to a buffer overflow which can lead to arbitrary code execution. An attacker can get access to the Cloud infrastructure which can affect the Cloud provider as well as the user.

Listing 2. Vulnerable Code

```
...
recv(sock, netbuf);
sprintf(buf, netbuf, strlen(netbuf));
...
```

An example of the vulnerability on the system layer can be constructed by means of traditional security considerations. A Cloud user can run several virtual hosts based on Linux and Windows to provide several web pages and an FTP file share. The web pages are provided by the Linux host running the Apache web server. The FTP server is provided by the Microsoft IIS FTP server running on the Windows host. As the Apache web server shows some of the content uploaded on the IIS[13] FTP server, the Windows host transfers the content using the SMB[14] server deployed on the Linux host. In case the running server applications are vulnerable to specific attacks, the hosts can be compromised by an attacker. Consider an IIS FTP server vulnerable to the *Microsoft IIS FTP Server NLST Response Overflow* described in CVE-2009-3023 [10] flaw and a Samba server vulnerable to the *LSA Heap Overflow* described in CVE-2007-2446 [10]. An attacker can easily gain access to the systems by using public exploits.

As described in [2], it is even possible to exploit the virtual machine monitor using vulnerabilities in the implementation. By this technique, the attacker can break out of the virtual machine to get access to the hardware layer. In case the attacker gains access to the hardware layer, he can easily compromise any VM provided by this infrastructure. This heavily affects all Cloud users running on this infrastructure. As described in [3], interpreters can also be vulnerable to specific language based attacks. An attacker can exploit the interpreter and gain access to the infrastructure running it. Generally, it seems to be possible for attackers to cross the different layers in the Cloud architecture using specific attacks on infrastructure implementations, e.g., language interpreters and virtual machine monitors. Furthermore, attackers might consider using the Cloud to run attacks on additional victims. A compromised Cloud infrastructure could be used to run distributed Denial of Service (DDoS) attacks on large networks. Thus, the traditional approach of Intrusion Detection can be useful in such environments. One of the most important questions in a distributed and complex environment (such as the Cloud infrastructure) is how to manage many components in the context of virtualization.

III. Deploying IDS in the Cloud

A. Requirements

For both Cloud users and Cloud provides, deploying IDS sensors in the Cloud infrastructure is highly motivated. The Cloud users need IDS to detect attacks on their services, e.g., if the Cloud user runs a web shop for its customers, it needs to know if there are attempts to run XSS attacks. Additionally, the Cloud user needs to know if the used services or hosts are used to attack other victims, e.g., if the Cloud user runs several VM hosts, it should know when these hosts are abused by an attacker to penetrate other hosts. Furthermore, it could be useful to separate the IDS for each user from the actual component being monitored. If the IDS is used to monitor a VM host on the host itself, it can not be guaranteed that the IDS (e.g. a HIDS) works properly when the host is compromised, as the attacker could have modified it not to send any reports. A similar issue can be observed in case of a web-based IDS detecting XSS attempts, which is integrated into the web shop it should monitor. Thus, the IDS and its monitoring target need to be separated. Additionally, the Cloud user needs a separate set of rules and thresholds to

configure their private IDS. Each user may have different requirements and rulesets for the concrete IDS running to secure its infrastructure, as the user running only Linux VMs with ssh service can simply drop the rules used to detect web based attacks and Windows-based attacks in the related NIDS.

The Cloud providers need to detect attacks on their Cloud infrastructure, i.e., attacks on the infrastructure itself as well as attacks on the services of the users. These attacks can be performed by an external attacker or by the user itself who may or may not be compromised. Furthermore, the providers need to know if their infrastructure is being used by attackers to penetrate other victims, e.g., a DDoS attack conducted from a Cloud provider may affect its reputation and can be easily disturbed by the provider itself. To secure the IDS itself and to optimize the efficiency, the IDS needs to be separated from the target being monitored. Additionally, the Cloud provider can use VMM functions to monitor virtual machines. This increases the efficiency of the detection a lot.

B. Architecture

Figure 2 shows a possible IDS deployment in the Cloud infrastructure. Each virtual component should be secured by a separated IDS sensor, which is responsible for one virtual machine and can be configured by the Cloud user. This sensor should monitor the virtual components based on the requirements of the user. For each layer, there should be Network-based sensors and Host-based sensors deployed accordingly. Additionally, the IDS sensor should report alerts to a central IDS management system, which is responsible to gather and preprocess the alerts of all sensors. Cloud users can view and configure their sensors by using the IDS Management system. Users can configure the type of IDS which is attached to their virtual component. The specific ruleset and thresholds can also be configured for each deployed sensor to improve the efficiency of the sensor. The user can see the detected attacks in the management system and can decide to perform additional countermeasures, e.g., DROP the network packet or IGNORE the POST request including *SCRIPT* tags.

The Cloud provider is responsible to enable different IDS VMs and provide the possibility to attach them to a specific VM component. There should be at least one IDS VM per layer: Network-based and Host-based IDS for the system and the platform layer, as well as Network-based IDS for the application layer. Furthermore, the provider can recognize attacks on the virtual components of their users by using the IDS management system. Large scale attacks on several users or performed by users can be detected easily by correlating the incoming alerts. The provider can also use automatic countermeasures to disturb attacks immediately, e.g., to decrease the provided ressources related to a user who is running a DDoS attack, or to shut down compromised hosts before more harm can be done.

Furthermore, the Cloud providers should use IDS sensors to monitor their Cloud infrastructure, i.e., the hardware level of the infrastructure to detect attacks on the infrastructure itself, but on the virtualized components. Alerts referring to attacks on the infrastructure can also be correlated together to determine large scale attacks.

IV. IDS Management in the Cloud

As already mentioned, we focus on the concept of virtualization in this paper, since we consider it as an important technology to implement Cloud Computing. IDS management needs to provide functionality to handle virtualization to be successfully deployed in the Cloud. The virtualization can provide a separation of the IDS or the monitored system, to minimize the impact of a possible attack. VMs support fast recovery in case of an emergency. Additionally, a VM-based system allows to use cross-platform sensors. The capability to stop and resume VMs can be used as an easy and effective countermeasure for an ongoing attack. Derived by the motivation to integrate these properties supported by virtualization, we propose the IDS VM management system as shown in Figure 3. It includes several *IDS Sensor VMs* and a *IDS Management Unit*. The *IDS Management Unit* consists of four active components: *Event Gatherer*, *Event Database*, *Analysis Component*, and *IDS Remote Controller*. The *Event Database* is a passive storage that holds information about all received events. It can be accessed through the *Analysis Component*. *User* controls the IDS management through direct interaction and configuration of the core components. The *IDS Sensors* on the VMs are responsible for detecting and reporting malicious behaviors. Each sensor is connected to the *Event Gatherer* component to transmit triggered events. A sensor, which could be a running NIDS with all its signatures and configurations, can be configured through the *IDS Remote Controller*.

The IDS sensor identifies malicious behavior and generates alerts through a reporting component to the output, which will be processed by the *Event Gatherer*. The sensor is an independent process, which can be any NIDS or HIDS, e.g., Snort or Samhain. The *Event Gatherer* is responsible for collecting all events from *IDS Sensors*. As shown in Figure 3, the *Event Gatherer* is used to standardize the outputs from different sensors as well as realize the logical communication, such as file-based or network-based, between the sensor and the management unit. The gatherer consists of several Plugins: *Senders*, *Receivers*, and *Handlers*. *Receivers* are used to read alerts and convert them to IDMEF. *Senders* are used to write alerts to a destination, e.g., a network, a database, or a folder. *Handlers* can be used to modify alerts in processing, e.g., to log each alert from a specific sensor. Each event is made persistent in the *Event Database* storage. The gatherer can be configured by the user and is connected to each sensor it receives events from. A gatherer can be the running instance of an IDS management component that accepts connections and writes events to the database.

The *Analysis Component* is responsible for representing the gathered events as well as analyzing the events, e.g., correlating events to complex attack scenarios. Therefore, it has access to the *Event Database* storage. It can be configured and controlled by the user. Such an analysis component might be the front end to the database within your IDS management component. The *IDS Remote Controller* is responsible for remote configuration and control of all connected *IDS Sensors*. It has access to each sensor configuration. The remote controller itself can be controlled and configured by the user. The

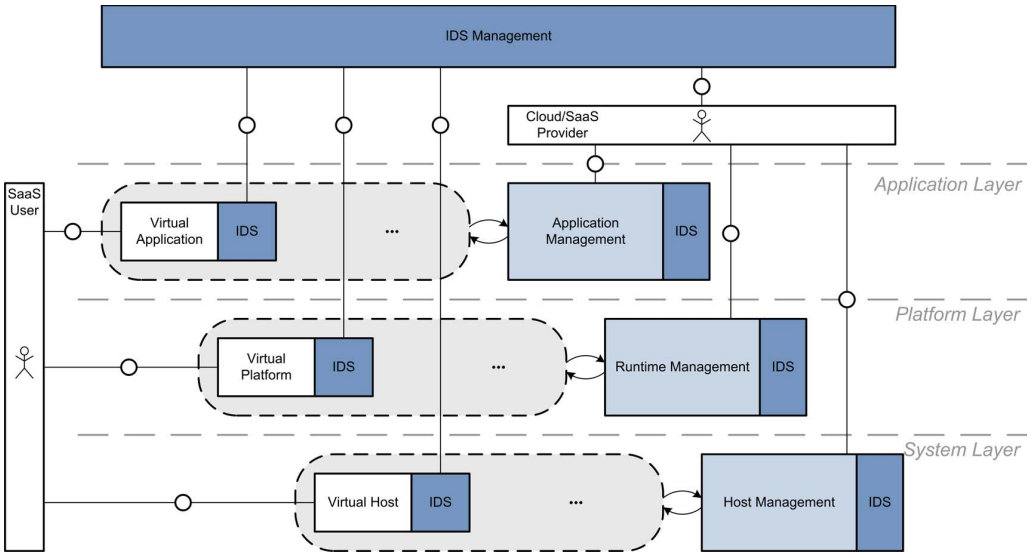

Fig. 2. IDS in the Cloud

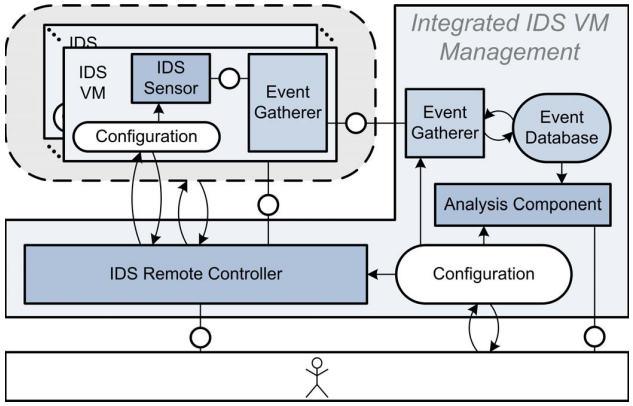

Fig. 3. Architecture of VM-integrated IDS Management

remote controller can be a remote controlling and monitoring software.

For managing IDS VM, the IDS *Remote Controller* plays an important role. The remote controller is able to communicate with IDS VMs and their hosted IDS sensors. Three important aspects of IDS VM management, i.e., VM control, VM monitoring, and VM configuration, are integrated into the remote controller. The control aspect of VM management includes VM operations: startup, shutdown, stop, resume, restore, and update. The remote controller can read and modify the configuration of the IDS sensor. The virtual machine management is implemented to start, stop and restart User-Mode-Linux based VMs. Additionally the recovery of the VM is possible from the management system. All features are implemented by using the User-Mode-Linux (UML) utilities, in particular, the UML management console [22].

This VM-Compatible IDS management includes not only information on the sensors but also information on the virtualization environment, such as the status of the virtual machines: whether the VM is running or not, how the workload of the VM is, and any other related information. Additionally, it includes information on the IDS VM assignment to monitored VMs and a way to configure these assignments. Several parameters of the running IDS VMs can be configured directly through the IDS management, such as the basic file system, usable disk space, or memory. The recovery of VMs is used as attack prevention mechanism for compromised VMs. Another possible attack prevention could be a temporary stop of the VM.

To implement the proposed IDS management system, several challenges need to be addressed. First, the output of different sensors is not standardized due to different technologies and formats. Second, the communication between the sensor and the management component has to be flexible as different virtualization approaches allows different communication methods, e.g., communication over a simulated network, message files, or shared memory. Third, complex architectures with multiple sensors on different VMs require flexible deployment scheme. To unify the output of the sensor, we use IDMEF as the basis for all the communication inside the IDS management system. A message has the type *IDMEFMessage*, which is a part of the IDMEF library. The IDMEF library is generated by JAXB [20] based on the IDMEF XML schema provided by the RFC [16]. To connect different IDS sensors, a set of receiver plugins is implemented, e.g., *XMLTcpReceiver*, *SnortCSVReceiver*, *SamhainSyslogReceiver*, and *FSecureSyslogReceiver*. The storage of alerts is realized by different sender plugins, e.g., the *XMLTcpSender*, the *XMLFileSender*, and the *AlertSQLSender*. The *XMLTcpSender* and *XMLTcpReceiver* are capable of sending and receiving IDMEF messages through a specified socket. This pair is used for communication between the IDS management and its sensors. The JAXB technology is used to generate related Java classes and to perform marshalling and unmarshalling.

The VM management supports UML by using UML utilities, in particular, the UML management console. The analysis component and graphical front-end for displaying alerts and managing VMs and sensors is realized as web application using Java Servlets.

V. Future Work

The implemented Proof-of-Concept is just a first step in the direction of a complex IDS management system deployed in the Cloud. An interesting future topic is the correlation of alerts from the virtual components in the Cloud infrastructure. It is likely that new correlation methods will be needed, which take the special configuration and involved parties of the Cloud architecture into account. An area of research will emerge covering IDS techniques for virtualized components, e.g., a web application container can use external profiling techniques (provided by the container itself) to detect malicious behavior. To practically apply the deployment, performance and scalability issues need to be considered as the next step. For this purpose, a comparison of the performance between existing IDS management systems and our solution is an interesting task. Additionally, further types of IDS which are useful for the Cloud need to be determined and integrated into the architecture. The analysis component needs to be extended to enable different modules and approaches for event correlation, such as sorting, filtering, and tagging. Furthermore, visualization techniques for alerts can be analyzed and integrated to support manual analysis. More receivers and senders need to be realized to support most common IDS solutions and additional communication channels. Currently, only basic UML-based features for starting, stopping, restarting, and recovering IDS VMs are implemented. This needs to be extended by generic methods for VM management, e.g., to control and monitor Xen [19] based VMs. The possibility of performing countermeasures by means of VM management is another interesting topic for further research.

VI. Conclusion

Facing the complexity of a Cloud architecture, this paper focuses on proposing a deployment architecture of Intrusion Detection Systems in the Cloud. We discuss and list several existing threats for a Cloud infrastructure and are motivated to use Intrusion Detection Systems (IDS) and its management in the Cloud. We propose the deployment of IDS on each layer of the Cloud to gather and correlate the alerts from different sensors. As we consider virtualization as one of the key technologies for Cloud Computing, we propose an extensible architecture for integrating VM management and IDS management. The proposed architecture meets the requirement of extensibility and reflects the state-of-the-art architecture of general distributed IDS. Due to more supported logical communication channels, the sensors can be located anywhere in the network or even inside virtual machines. Such information as VM status, VM workload, and IDS-VM assignments can be monitored and the involved IDS VMs can be stopped, started, and recovered by the management system.

References

[1] M. Armbrust, A. Fox, R. Griffith, et al.: *Above the Clouds: A Berkley View of Cloud Computing*, Website: *http://radlab.cd.berkley.edu/*, UC Berkley Reliable Adaptive Distributed Systems Laboratory (Feb. 2009).

[2] J. Rutkowska, R. Wojtczuk: *Xen Owning Trilogy*, BlackHat 2008, Las Vegas, July 7th, 2008.

[3] Justin Ferguson: *Advances in Attacking Interpreted Languages*, World Security Professional Summit in European Union 2008 (EUSecWest'08), Amsterdam, Netherlands (2008).

[4] *Google App Engine*, Website: *http://code.google.com/appengine/*, Google (accessed on Oct 2009).

[5] *Windows Azure Platform*, Website: *http://www.microsoft.com/azure/*, Microsoft Corporation (accessed on Oct 2009).

[6] *Amazon Elastic Compute Cloud (Amazon EC2)* Website: *http://aws.amazon.com/ec2/*, Amazon (accessed on Oct 2009).

[7] M. Martin and M. S. Lam: *Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking*, In: Proceedings of USENIX Security Symposium (USENIXSec'08), pp. 31-44 (2008).

[8] *F-Secure Linux Security*, Website: *http://www.f-secure.com/linux-weblog/*, F-Secure Corporation (accessed Oct 2009).

[9] *Samhain IDS*, Website: *http://www.la-samhna.de/samhain/* (accessed Oct 2009).

[10] *Common vulnerabilities and exposures (CVE)*, Website: *http://cve.mitre.org/*, Mitre Corporation (accessed Oct 2009).

[11] *Snort IDS*, Website: *http://www.snort.org/* (accessed Oct 2009).

[12] *Prelude IDS*, Website: *http://www.prelude-ids.com/*, PreludeIDS Technologies (accessed Oct 2009).

[13] *Microsoft Internet Information Server (IIS)*, Website: *http://www.iis.net/*, Microsoft Corporation (accessed Oct 2009).

[14] *SMB NetBIOS RFC 1001*, Website: *http://www.rfc-editor.org/rfc/rfc1001.txt* (accessed Oct 2009).

[15] Laureano, M., Maziero, C., Jamhour, E.: Protecting host-based intrusion detectors through virtual machines Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 51, Issue 5, pp. 1275-1283 (April 2007).

[16] Debar, H., Curry, D., Feinstein, B.: The Intrusion Detection Message Exchange Format, Internet Draft Technical Report, IETF Intrusion Detection Exchange Format Working Group (July 2004).

[17] Moore, D., Shannon, C., Brown, D.J., and et al.: Inferring internet denial-of-service activity. ACM Transactions on Computer Systems (TOCS), vol. 24, Issue 2, 2006, pp. 115-139.

[18] *The Anti-Virus or Anti-Malware Test File*, Website: *http://www.eicar.org/*, European Institute for Computer Antivirus Research (EICAR) (accessed Oct 2009).

[19] *Xen Project*, Website: *http://www.xen.org/* (accessed Oct 2009).

[20] *Java Architecture for XML Binding (JAXB)*, Website: *http://java.sun.com/developer/*, Sun Developer Network (SDN) (accessed Oct 2009).

[21] *Python*, Website: *http://www.python.org/* (accessed Oct 2009).

[22] *User Mode Linux (UML)*, Website: *http://user-mode-linux.sourceforge.net/* (accessed Oct 2009).

[23] Roschke, S., Cheng, F., Meinel, Ch.: An Extensible and Virtualization-Compatible IDS Management Architecture In: Proceedings of 5th International Conference on Information Assurance and Security (IAS'09), IEEE Press, vol. 2, Xi'an, China, pp. 130-134 (August 2009).

[24] Cheng, F., Roschke, S., Meinel, Ch.: Implementing IDS Management on Lock-Keeper In: Proceedings of 5th Information Security Practice and Experience Conference (ISPEC'09), Springer LNCS 5451, Xi'an, China, pp. 360-371 (April 2009).