Towards Targeted Intrusion Detection Deployments in Cloud Computing

Noor Ahmed, AND Bharat Bhargava
Purdue University

Preventing security violation incidents or collecting dependable system audit trails for post incidents requires successfully detecting anomaly-based abnormal or intrusion activities. However, properly positioning the necessary tools for maximum detection can be inelegant and limited for systems that can be formed and destroyed on demand, such as the cloud. In this paper, we present a simplified taxonomy to aid targeted intrusion detection system deployments in cloud platforms. To illustrate the effectiveness of the proposed approach, we show two stealthy intrusion schemes and a preventive and adoptable detection strategy using Virtual Machine Introspection in a realistic use case scenario.

Keywords: Cloud Computing, Cloud Security, Anomaly and Intrusion Detection Systems, VM-Security, Virtual Machine Introspection.

1. INTRODUCTION

Cloud computing is a cost-efficient computing paradigm that can scale up and down on demand with limited system management or maintenance requirements. This system management simplification is due to the structured building blocks and the hierarchical deployment models of the cloud. The three widely accepted cloud platform categories are; a private cloud, built in within organizational boundaries; a public cloud, typically rented from commercial companies (e.g. Amazon, RackSpace, HP, Google and Microsoft); a hybrid cloud that provides a composition of private and public solutions. Within each of these platform categories lie three service deployment models similar to the utility services, pay-as-you-go service fee model: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS).

The main difference between the three service deployment models is the level of administrative responsibility for managing and maintaining the cloud systems between the cloud user and the cloud provider. For example, for SaaS, the software is a fee-based offered by the cloud provider, i.e. salesforce.com. For PaaS in public clouds, for example, some applications (e.g., web-servers) can be deployed with little or no cloud conformance or modification requirements where the management/maintenance responsibilities are left to the cloud provider. Additional management responsibilities are transferred to the user for the IaaS model.

Given the economic benefits of the cloud, a recent survey on Cloud Adaptation and Trends for 2013 [Averest et al. 2013] showed that security is the major drawback for taking full advantage of the cloud. One of the main reasons is that traditional defensive security techniques have failed to prevent undesired system activities or denying attackers from gaining control of the system. For example, the impact of the cyber attack on Target, Sony, and others [Granville . Feb, 2015] clearly show such security failures.

Since protecting systems in dynamic platforms (i.e. cloud) with the traditional security tools and techniques is increasingly challenging, one common defensive scheme is to continually monitor anomalous system behavior for intrusion in order to prevent security violation incidents or reduce its impact once it happens using an Intrusion Detection System (IDS). However, existing IDS techniques are ad-hoc and ill-suited to properly map to cloud platforms, for example, advanced network-level IDS work has never been deployed in realistic settings [Catania et al. 2012].

The different deployment models further complicate the security problems [Subashini et al. 2011]. Clearly, one of the main reason is that the deployment models have contributed abstraction layers that further complicate IDS deployment. Such abstractions is widely accepted, due to its financial benefits of reducing IT operational costs, simplifying system integration, and ease of system deployment managements in global scale. Consequently, dictate the type and the IDS positioning for a given platform. Thus, we believe a simplified IDS taxonomy to easily map the different deployment models is critical.

In this paper, we present a general simplified taxonomy for IDS solutions and cloud service deployments to enable targeted IDS positioning in such environments. To the best of our knowledge, this is an area that has yet been explored, thus, the key contribution of our work. In section 2, we give a brief background overview of cloud computing ecosystems and its threat models, followed by the state-of-the-art IDS and its limitations. We present the taxonomy and the targeted IDS scheme in section 3. In section 4, we present a use case scenario of an attack and preventive IDS scheme to illustrate the effectiveness of the proposed solution. We discuss related work and conclusion in sections 5 and 6, respectively.

2. BACKGROUND

In this section we give a brief overview of cloud ecosystem and IDS solutions to set the context of targeted deployments of such environments.

2.1 Cloud Eco-System

The key promise of cloud utilization is the cost effectiveness of the computing resources that can scale up/down on demand, referred as pay-as-you-go elastic computing. While on-demand elasticity of computing resources enabled by the cloud is undeniably useful, its increasingly challenging to guarantee provable reliability due to the increased cyber attacks and transient hardware and software design faults inherent on the commercial-off-the-shelf hardware built on such infrastructures. Furthermore, the architectural blue print of the service deployment models further complicate the security issues. A survey of the security issues in service models is reported in [Subashini et al. 2011].

On the other hand, such infrastructures also provide unprecedented security capabilities, enabled by the structured underlying computing architecture and virtualization to safeguard systems that were not known to non-virtualized domains. Virtualization enables isolating many operating systems and applications to run on multiple instances on the same physical hardware, referred to as multi-tenancy.

The Virtual Machine Monitors (VMMs), or hypervisor, is the machinery that enables resource virtualization. In additon, the VMM enables intercepting the communication between the virtual requests and the available physical resources. This is known as Virtual Machine Introspection (VMI) which affords an opportunity to leverage as an intrusion prevention tool. However, VMI-based solutions are only suitable at IaaS abstraction layer deployments, not PaaS and SaaS. For other deployment models, the cloud providers typically employ similar or proprietary IDS schemes which we will not discuss in this work. We present a lightweight, VMI-based IDS solution to illustrate how our proposed simplified IDS taxonomy maps to a targeted IaaS deployment.

2.2 Threat Models

Since our work develops taxonomy for mapping a VMI-based IDS solution to cloud platforms, we consider the standard assumption of cloud threat model, a node/VM compromise. To illustrate the preventive strategy using the proposed IDS solution and its mapping scheme, we consider an adversary that gains full control/privilege of a virtual machine undetected by the traditional defensive mechanisms, a valid assumption in cyber space. The adversary can be a user with physical access to the guest VM node (not the hypervisors), escalated privilege, or even an escaped tenant from the neighboring VM.

We assume that hypervisors/VMMs are secure. We also assume the cloud provider's software management stack is secure, especially when managing cloud resources such as provisioning and de-provisioning instances as well as virtual introspection. Clearly, the threat model can be seen as the cloud providers problem space; however, this can also be found in the problem space of private cloud as we show in our prototype in the evaluation section (Section 4).

2.3 Intrusion Detection Systems

There has been a wide array of IDS work in the past few years; a comprehensive review is given in [liao et al. 2013]. Special treatment of the network-level IDS current techniques and open issues is given in [Catania et al. 2012] and deployment strategies for selected IDS techniques are presented in this survey [modi et al. 2010]. Other notable techniques employed by the cloud providers include techniques to independently collect monitoring information at different levels of granularity to address the relationship behavior between the VMs rather than anomalies [Wang et al. 2010].

For network-level IDS schemes, also in IaaS service model, the timely collection of network traffic and deriving the conclusion for determining the correct anomalous behavior is critical. Yet, these decisions are not 100% accurate; some cases drop to 80% [Goldszmidt et al. 2011] which is unacceptable to some applications. Furthermore, the accuracy of detecting a threat is hindered as the systems grow (i.e. scalability issues). Typically, most applications are built with components authored by different developers who choose what to log [Xu et al. 2009] which further contributes to the volume of collected data that needs to be automatically analyzed in a timely manner with high accuracy; a problem further worsened if the data is unstructured. Labelling and associating such data to derive an accurate anomalous behavior typically poses a greater challenge.

One common theme among all state-of-the-art IDS solutions is that they are designed for one layer of abstraction of the cloud; thus, they require tweaks to accommodate other threats and layers across the infrastructure. Therefore, its prudent to position in the most effective spot in order to adapt to changes in any given threat. In this work, we propose a simplified methodology for mapping IDS techniques to a cloud platform in order to target the most effective spot to adopt new preventive strategies upon detecting a successful intrusion of a given threat while reducing data collection in variety of system areas.

3. TARGETED IDS DEPLOYMENTS

Targeted IDS deployments in cloud environments require mapping IDS solutions to specific layers of abstraction of the cloud. However, such mapping is not one-to-one. In this section, we classify existing IDS solutions in relation to cloud types and service deployment models to simplify such mappings.

3.1 Mapping IDS Taxonomy to Cloud Service Models

Typically, IDS solutions in the literature are classified as network or host-based. For cloud computing environments, there are guests on top of the hosts and the network which are not accessible to service models; SaaS and PaaS. Therefore, such classifications lags on mapping IDS solutions to a proper cloud service deployment model. An overview of an extended IDS taxonomy is given in [liao et al. 2013]. In relation to cloud deployment models, we introduce a simplified IDS taxonomy;*System, Data and Implementation Scheme* depicted as three rollers in Figure 1 below on the left and, on the right, cloud deployment models are depicted as layered rings.

In Figure 1, private and public cloud flavours are depicted in the left and right quadrants of the rings, respectively. Combing the use of the two produces the hybrid model, represented in the bottom half. At the core of the cloud platform, the inner ring, represents the hardware and networking where a hypervisor is used to virtualize the underlying resources, depicted as the second ring. On top of the hypervisor, are the three service delivery models; IaaS, PaaS, and SaaS respectively. The short double arrows represent IDS positions, e.g. by the data center (cloud providers) or the internet service provider (ISP) for corporate private clouds.

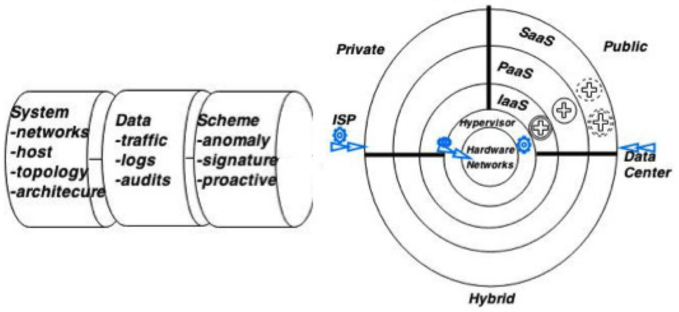


Figure 1: System Model - IDS Taxonomy Classes: System, Data and Scheme (left) and Cloud Service Deployment Models (right). The attributes in each taxonomy class (*i.e. network*) can only be mapped to (*IaaS*) service model.

IaaS, PaaS, and SaaS service models offer different protection schemes for the applications deployed (depicted as a plus sign). From the cloud users prospective, applications deployed closer to the hypervisor ring (e.g. on IaaS) have greater protection layers (depicted as double rings and dotted circles as moving further away from IaaS). The arrows and gears represent positioning additional protection schemes like firewalls at the ISP found in private clouds or the data centers in public clouds. Its intuitive to see that these layering schemes are easier to map a simplified IDS capabilities on such environments.

We consider the traditional network and host taxonomy under the *Systems* class (depicted in the 1st roller). The main rationale behind our grouping is due to the cloud deployment models where the line between the host and network is merged into the guest and the host. For example, SaaS and PaaS deployment models have no access to the network traffic and the host machine due to the multi-tenancy nature of the cloud. Therefore, the Systems class covers everything including the network, host, guest, the topology, and the architecture.

Data collected and the scheme used to detect intrusion are critical to the success of the IDS because different abstraction layers of the cloud offer different data sources and permissible IDS scheme deployments. Therefore, we classify *Data and Scheme* independently, where the data can be the network traffic data, logs, or system audit trails the *Scheme* is simply the scheme implemented the IDS system. For example, anomalous-based, behaviour-based, signature-based, and proactive monitoring (continuous monitoring), etc.

Anomalous-based intrusion capabilities available for the applications are in direct relation to their deployed service ring. For example, its intuitive to see network traffic data cannot be collected from an application deployed in SaaS unless customized tenant traffic is directed to the users VM by the cloud provider. Additional data sources in this ring include: audit trails logs, applications or database logs, the cloud providers logs (for billing purposes), and application signatures (HMAC hashes).

Additional logs at the platform level (e.g. detailed OS logs, application servers, etc.) are available in PaaS. The traditional anomalous behavior found in these applications and system logs include function calls made/logged, temporary files written somewhere, failed connections to a system or database, or too many unsuccessful login attempts. Using these as the key data sources for the IDS, one can detect specific threats. However, if the guest OS/VM is compromised and undetected, the integrity of these logs is difficult to verify, thereby, receiving logs/signatures the attacker prepared for the IDS.

As noted earlier, threats that bypass defensive solutions within the OS can be detected below the OS, a mechanism known as Virtual Machine Introspection (VMI). However, cloud user-controlled VMI-based IDS solutions are only available in IaaS. Note that VMI-based or other proprietary solutions are typically employed in PaaS and SaaS by the providers. Each service deployment model offers different data sources and enumerating unknown threats to craft for specific IDS is cumbersome and ineffective. Therefore, applications need to be deployed based on their critically of protection needs into the appropriate service models (rings) and map the traditional IDS solutions suitable for that ring or craft if needed. To simplify the mapping process, we introduce a decision support matrix (discussed next).

3.2 Deployment Decision Support Matrix

Determining the target spot for IDS deployment for applications in cloud platforms is highly dependent on two factors; the threat model and the available data source to derive the anomalous behavior from. In other words, the fundamental question is what we consider anomalous behavior under a given threat and what are the data sources that we need to collect and derive the undesired behavior from? Thereby, the data sources are the key determining factor of the success or failure (miss or high false alarm rates) of the IDS solution deployed. Since this work is focused on the taxonomy simplification for targeted deployment, we don't consider covering the proposed IDS alarm rate accuracy.

Table I: IDS Decision Matrix

Provider Model	System	Data	Scheme
SaaS	N/A	App logs, Audit logs, etc.	Signature, Behavior, etc.
PaaS	N/A	OS logs, App logs, Audit logs, etc	Host, Signature, Behavior, etc.
IaaS	VMI	Live Memory snapshots	Proactive and Reactive

Table 1 show the decision matrix; the cloud provider model in column one, the IDS system in column 2, and Data and scheme used for a given threat in columns 3 and 4. We use a motivating example use case scenario (discussed next) with the system model reference in Figure 1 to show how the entries are filled in for a given IDS solution.

3.3 Use Case Scenario

Enumerating possible threats and defensive IDS tools to illustrate the benefits of the proposed simplified mapping scheme is infeasible, using the decision support matrix in Table 1 entries; we show how the targeted solution adopts to the threats of a given intrusion use case.

Consider an adversary compromising a VM node using kernel root kit, or other methods. The adversary can be an attacker or authorized users with escalated privileges, a valid assumption in cyber space. For effective prevention strategy, we are interested in solutions that are outside the VM that we are trying to protect. Thus, we develop a lightweight VMI-based IDS solution for this threat. VMI-based solutions can only be deployed in IaaS service models. Given the threat and the solution approach, we fill the matrix (Table 1) entries in SaaS and PaaS System columns with N/A which means that system-level or architectural solutions are not available in these service deployment models. *Data* and *Schemes* available in that service model, as described earlier, are also filled in their appropriate columns.

Our lightweight, VMI-based solution pro-actively gathers live memory data through time intervals and reactively take an action if incidents are detected in the given memory data structures, thereby, the entry of System column is *VMI* and *Data* column is memory data structure snapshots, and Scheme column, as the name implies, is filled with our proactive and reactive. Proactive is typically defined as the continuous monitoring or system observation while reactive is the reaction or the response upon detecting abnormal or intrusion behavior. To illustrate our intrusion scenario, we adopted two stealthy attacks that are difficult to detect inside the VM:

(1) ***Attack 1:*** *We mimic a node compromise by logging into the VM, stopping the victim process, and starting a malicious one with the same name but different functionality; presumably stealing data. The process is a small c program that simply prints a counter value, its process ID, and a function name print() and sleeps for few seconds in a continues loop.*

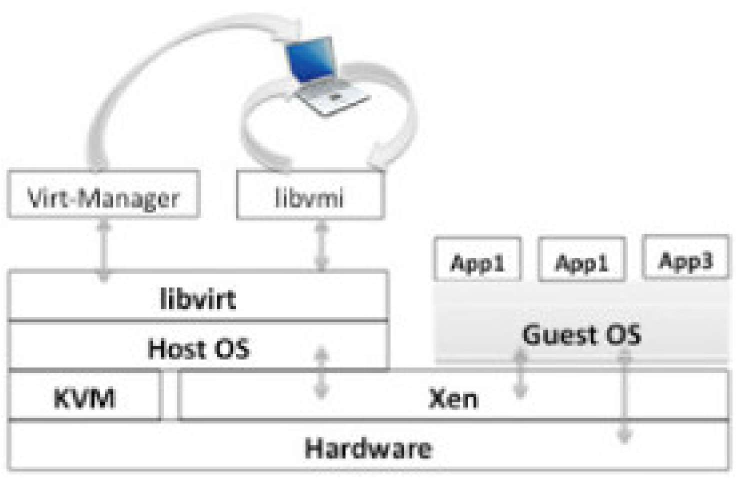


Figure 2: High level Architecture - The bottom of the stack is the hardware. On top of the hardware is the Hypervisor, a *Xen* or a *KVM*. Note, only one hypervisor can be deployed on any given node. We used *Xen* in our experiments as depicted the data follow (arrows) between the guest OS and the host. The Virtual Machine Manager (Virt-Manager) (top left) enables provisioning/de-provisioning VM resources (CPU, Memory, etc) through the *libvirt* library, a library for virtualization. We used *libvmi* to intercept calls between the guest and the host depicted in the laptop.

(2) ***Attack 2:*** *To bypass the detection scheme used for attack1, in this attack, we hijack the application by loading/injecting a shared library and diverting to make additional function calls without stopping the process, print2() in this case, that are implemented in the injected shared library. In a realistic setting, some applications have many runtime dependencies and may be orchestrated with other applications. Restarting the process causes to break the dependencies or the chain, thereby triggering an alarm.*

Since the focus of this work is to detect anomalous behavior in the cloud platforms, we will not discuss the details of the attack implementation. For those interested, the program and the application hijacking attack process used in this work are downloaded from the public domain.

4. EVALUATIONS

A typical cloud computing node consist of a host OS using a hypervisor (Xen or KVM) to virtualize the computing hardware resources, and a guest OS to deploy applications. A high-level architecture of our evaluation system model is depicted in Figure 2 above. The key objective of our use case scenario is to prevent intrusions of the Apps (*App1, App2, App3,..*) on the Guest OS through the Host OS with *libvmi*. The detection process is as follows:

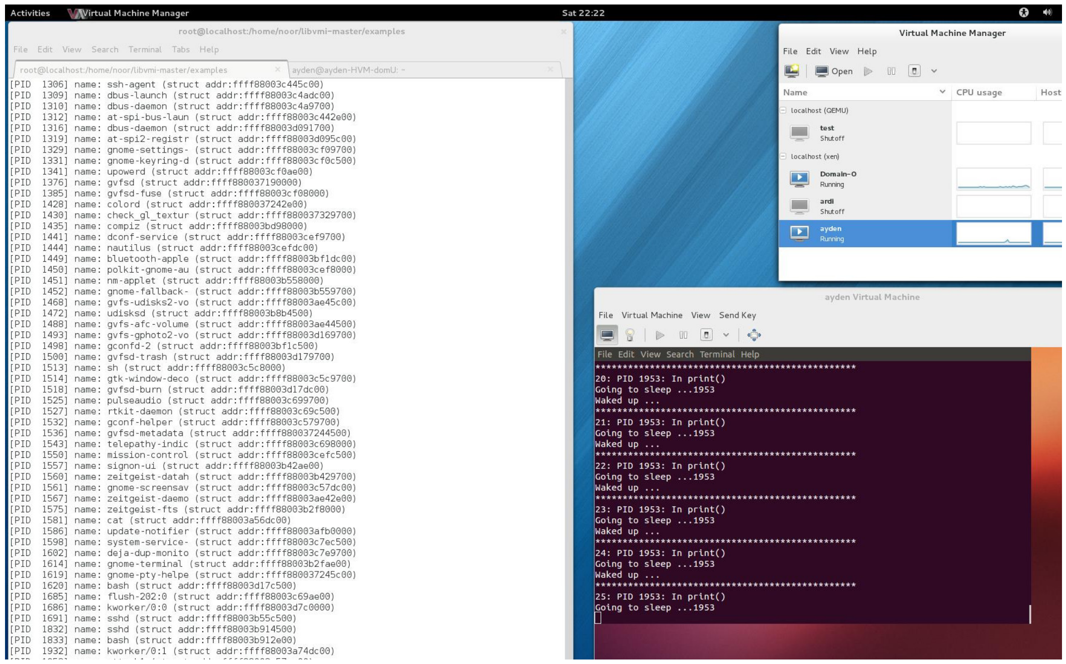
(1) *Start the application to be protected in the guest VM.*

(2) *Take memory snapshots in time intervals at the host OS.*

(3) *Analyze life memory data structure in real time and alarm any changes as an intrusion.*

4.1 Experimental Setup

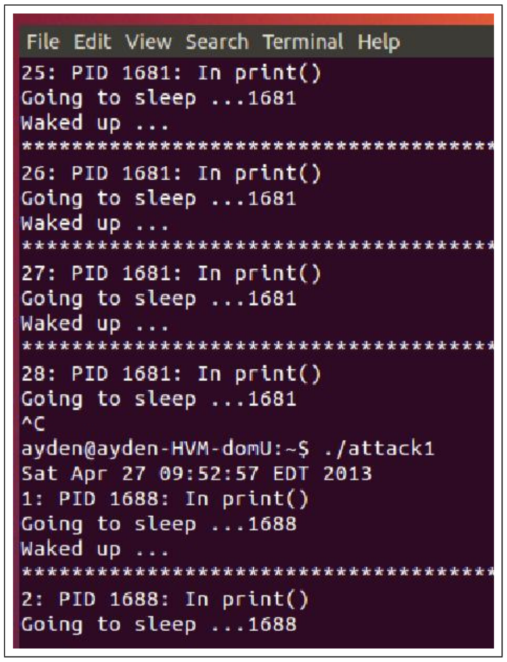
We conducted our experiments on a single machine from a private cloud built on OpenStack, open source cloud management software stack, from a cluster of Dell Z400 machines with Intel Xeon 3.2 GHZ Quad-Core with 8MB of Memory. A high level architecture of the platforms and tools used in this research is depicted in Figure 2 above, and the system-level details of the experimental Platform is depicted in Figure 3 below.



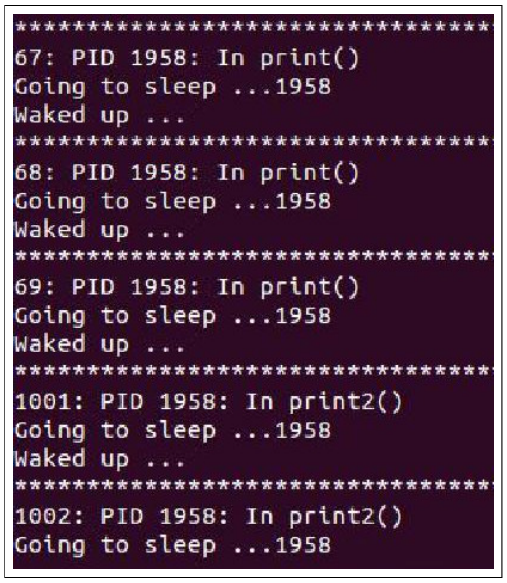
```
[PID  1306] name: ssh-agent (struct addr:ffff88003c445c00)
[PID  1309] name: dbus-launch (struct addr:ffff88003c4adc00)
[PID  1310] name: dbus-daemon (struct addr:ffff88003c4a9700)
[PID  1312] name: at-spi-bus-laun (struct addr:ffff88003c442e00)
[PID  1316] name: dbus-daemon (struct addr:ffff88003d091700)
[PID  1319] name: at-spi2-registr (struct addr:ffff88003d095c00)
[PID  1329] name: gnome-settings- (struct addr:ffff88003cf09700)
[PID  1331] name: gnome-keyring-d (struct addr:ffff88003cf0c500)
[PID  1341] name: upowerd (struct addr:ffff88003cf0ae00)
[PID  1376] name: gvfsd (struct addr:ffff880037190000)
[PID  1385] name: gvfsd-fuse (struct addr:ffff88003cf08000)
[PID  1428] name: colord (struct addr:ffff880037242e00)
[PID  1430] name: check_gl_textur (struct addr:ffff880037329700)
[PID  1435] name: compiz (struct addr:ffff88003bd98000)
[PID  1441] name: dconf-service (struct addr:ffff88003cef9700)
[PID  1444] name: nautilus (struct addr:ffff88003cefdc00)
[PID  1449] name: bluetooth-apple (struct addr:ffff88003bf1dc00)
[PID  1450] name: polkit-gnome-au (struct addr:ffff88003cef8000)
[PID  1451] name: nm-applet (struct addr:ffff88003b558000)
[PID  1452] name: gnome-fallback- (struct addr:ffff88003b559700)
[PID  1468] name: gvfs-udisks2-vo (struct addr:ffff88003ae45c00)
[PID  1472] name: udisksd (struct addr:ffff88003b8b4500)
[PID  1488] name: gvfs-afc-volume (struct addr:ffff88003ae44500)
[PID  1493] name: gvfs-gphoto2-vo (struct addr:ffff88003d169700)
[PID  1498] name: gconfd-2 (struct addr:ffff88003bf1c500)
[PID  1500] name: gvfsd-trash (struct addr:ffff88003d179700)
[PID  1513] name: sh (struct addr:ffff88003c5c8000)
[PID  1514] name: gtk-window-deco (struct addr:ffff88003c5c9700)
[PID  1518] name: gvfsd-burn (struct addr:ffff88003d17dc00)
[PID  1525] name: pulseaudio (struct addr:ffff88003c699700)
[PID  1527] name: rtkit-daemon (struct addr:ffff88003c69c500)
[PID  1532] name: gconf-helper (struct addr:ffff88003c579700)
[PID  1536] name: gvfsd-metadata (struct addr:ffff880037244500)
[PID  1543] name: telepathy-indic (struct addr:ffff88003c698000)
[PID  1550] name: mission-control (struct addr:ffff88003cefc500)
[PID  1557] name: signon-ui (struct addr:ffff88003b42ae00)
[PID  1560] name: zeitgeist-datah (struct addr:ffff88003b429700)
[PID  1561] name: gnome-screensav (struct addr:ffff88003c57dc00)
[PID  1567] name: zeitgeist-daemo (struct addr:ffff88003ae42e00)
[PID  1575] name: zeitgeist-fts (struct addr:ffff88003b2f8000)
[PID  1581] name: cat (struct addr:ffff88003a56dc00)
[PID  1586] name: update-notifier (struct addr:ffff88003afb0000)
[PID  1598] name: system-service- (struct addr:ffff88003c7ec500)
[PID  1602] name: deja-dup-monito (struct addr:ffff88003c7e9700)
[PID  1614] name: gnome-terminal (struct addr:ffff88003b2fae00)
[PID  1619] name: gnome-pty-helpe (struct addr:ffff880037245c00)
[PID  1620] name: bash (struct addr:ffff88003d17c500)
[PID  1685] name: flush-202:0 (struct addr:ffff88003c69ae00)
[PID  1686] name: kworker/0:0 (struct addr:ffff88003d7c0000)
[PID  1691] name: sshd (struct addr:ffff88003b55c500)
[PID  1832] name: sshd (struct addr:ffff88003b914500)
[PID  1833] name: bash (struct addr:ffff88003b912e00)
[PID  1932] name: kworker/0:1 (struct addr:ffff88003a74dc00)
```

```
**************************************************
20: PID 1953: In print()
Going to sleep ...1953
Waked up ...
**************************************************
21: PID 1953: In print()
Going to sleep ...1953
Waked up ...
**************************************************
22: PID 1953: In print()
Going to sleep ...1953
Waked up ...
**************************************************
23: PID 1953: In print()
Going to sleep ...1953
Waked up ...
**************************************************
24: PID 1953: In print()
Going to sleep ...1953
Waked up ...
**************************************************
25: PID 1953: In print()
Going to sleep ...1953
```

Figure 3: Experimental Platform - A Virtual Machine Manager (VMM, reffered as, *VirtManager*), an open source VM management tool, is depicted in the top right window. The VMM allows us to create, destroy, and graphically login to VMs. Highlighted is a VM named *Ayden* used for the experiments shown below the VMM with the attack scenario application running. On the left window is our VMI-based IDS running on the host OS.

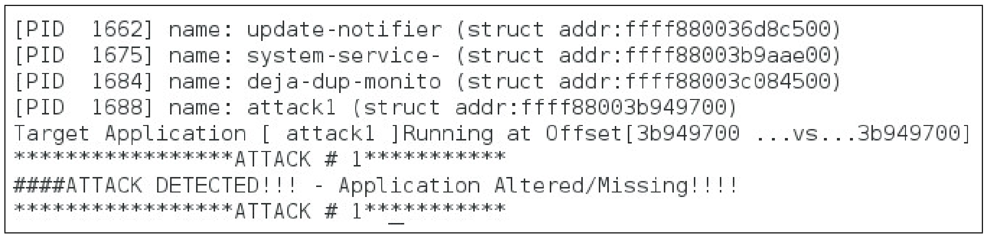


```
25: PID 1681: In print()
Going to sleep ...1681
Waked up ...
**************************************
26: PID 1681: In print()
Going to sleep ...1681
Waked up ...
**************************************
27: PID 1681: In print()
Going to sleep ...1681
Waked up ...
**************************************
28: PID 1681: In print()
Going to sleep ...1681
^C
ayden@ayden-HVM-domU:~$ ./attack1
Sat Apr 27 09:52:57 EDT 2013
1: PID 1688: In print()
Going to sleep ...1688
Waked up ...
**************************************
2: PID 1688: In print()
Going to sleep ...1688
```



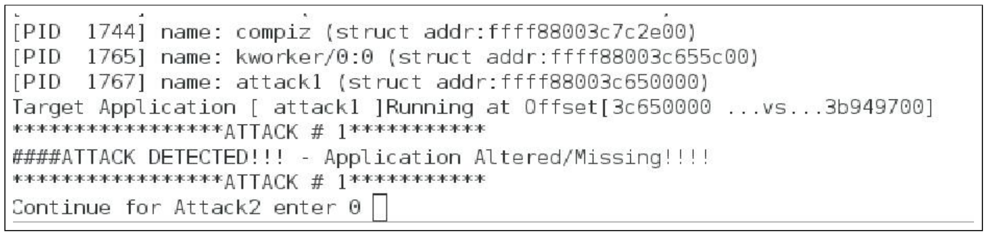
```
********************************
67: PID 1958: In print()
Going to sleep ...1958
Waked up ...
********************************
68: PID 1958: In print()
Going to sleep ...1958
Waked up ...
********************************
69: PID 1958: In print()
Going to sleep ...1958
Waked up ...
********************************
1001: PID 1958: In print2()
Going to sleep ...1958
Waked up ...
********************************
1002: PID 1958: In print2()
Going to sleep ...1958
```

Figure 4a Attack1 (left) and 4b Attack2-hijacked application (right) - *Attack1* is simply a *control-c* to terminate the program and restarting it. Note the counter also restarted. *Attack2* show the corrupted counter value and *print2()* function call made from the injected shared library while still printing it's process ID 1958. This attack is performed from another shell window that manipulate return address of the function calls of the running app.



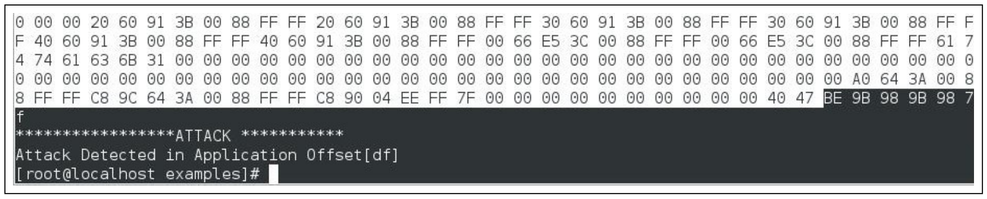
```
[PID  1662] name: update-notifier (struct addr:ffff880036d8c500)
[PID  1675] name: system-service- (struct addr:ffff88003b9aae00)
[PID  1684] name: deja-dup-monito (struct addr:ffff88003c084500)
[PID  1688] name: attack1 (struct addr:ffff88003b949700)
Target Application [ attack1 ]Running at Offset[3b949700 ...vs...3b949700]
*****************ATTACK # 1***********
####ATTACK DETECTED!!! - Application Altered/Missing!!!!
*****************ATTACK # 1***********
```

Figure 5: Attack1 Detection by process ID differences (*[PID 1688] name: attack1...*) - The process started with ID 1681 and then showed up with 1688 as depicted in Fig. 4a.



```
[PID  1744] name: compiz (struct addr:ffff88003c7c2e00)
[PID  1765] name: kworker/0:0 (struct addr:ffff88003c655c00)
[PID  1767] name: attack1 (struct addr:ffff88003c650000)
Target Application [ attack1 ]Running at Offset[3c650000 ...vs...3b949700]
*****************ATTACK # 1***********
####ATTACK DETECTED!!! - Application Altered/Missing!!!!
*****************ATTACK # 1***********
Continue for Attack2 enter 0
```

Figure 6: Attack1 Detection by the process's offset start address differences *3c650000 ..vs... 3b949700*.



```
0 00 00 20 60 91 3B 00 88 FF FF 20 60 91 3B 00 88 FF FF 30 60 91 3B 00 88 FF FF 30 60 91 3B 00 88 FF F
F 40 60 91 3B 00 88 FF FF 40 60 91 3B 00 88 FF FF 00 66 E5 3C 00 88 FF FF 00 66 E5 3C 00 88 FF FF 61 7
4 74 61 63 6B 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A0 64 3A 00 8
8 FF FF C8 9C 64 3A 00 88 FF FF C8 90 04 EE FF 7F 00 00 00 00 00 00 00 00 00 00 40 47 BE 9B 98 9B 98 7
f
****************ATTACK ***********
Attack Detected in Application Offset[df]
[root@localhost examples]#
```

Figure 7: Attack2 Detection - Detected the changes of the start of each address block within the program. This address block shift was caused by the memory block allocated for the shared library of the malicious code that implement *print2()*.

4.2 Experimental Results

For *attack1* (Figure 4a), the victim process started with a process ID *1681*, as expected, a new process ID 1688 was given to our malicious application after we restarted. A kernel rootkit can easily hide process ID from the process list, making it difficult to detect such intrusion within inside the VM. In Fig. 5, we detected the intrusion due to our target applications *attack1* process ID.

Note that, initially, the process started with ID 1681 at the offset *3b949700* and the process restarted with process ID 1688 and the same the offset *3b949700* as shown in Fig.5 above. This was a special case which clearly show that the hypervisor reassigns the address offset as soon as a process is terminated. Typically, restarting a process gets assigned with a new process ID 1767 and a new process address offset *Target App...Offset[3c650000vs...3b949700* as shown in Fig. 6. We detected such anomalies in either case; by simply verifying the existence of our applications's process ID or changes of the propram's associated starting address at the hypervisor level (below the compromised OS). However, when dealing with insidious attackers, this detection scheme can be easily bypassed by hijacking the application and altering the process's behavior without terminating the process as we showed in *attack2* in Fig. 4b.

Dealing with *attack2*, we began evaluating entire region of the application's address space to detect anomalies. The memory data structure snapshot contains the continuous address block of the running applications. We stored the first two hex characters of each block's starting address in order to prevent performance overhead inherent in live memory analysis. We then compare the incoming scan results of our target application in time-intervals. We detected the block starting address differed when we hijacked the application, thereby, shifting the address at position df as shown in Fig. 7. This shift in address space was due to the injected library physical address space allocated by the hypervisor. Thus, this clearly shows the benefits of the simple yet effective matrix (Table 1) as a guide for deploying an adoptable targeted IDS solution.

5. RELATED WORK

Many different cloud-based IDS solutions have been proposed in recent years. For example, [Bhaduri et al. 2013] proposed a scheme for detecting abnormal machine characteristics in cloud using machine performance data integrated into ganglia (an open source distributed monitoring system). An automatic management framework for compute cloud systems using a machine learning approach in institution-wide network presented in [Smith et al. 2010].

There are wide array of research of IDS deployment techniques, a notable one closely related to our work is presented in [Mazzariello et al. 2010]. Authors investigated detecting anomalies at the network-level originated by misbehaving customers. A practical scheme of defending Denial-of-Service (DoS) attacks on SIP protocols (Session Initiation Protocol) using Eucalyptus cloud management software stack which is equally as widely adopted as the OpenStack used in our experiments is presented. Our work is complimentary to the above as we target user application threats.

A Virtual Machine Introspection (VMI) library was first introduced by authors in [Garinkel et al. 2003]. Early works include protecting Unix program such as; ls, ps, etc. In recent years, VMI-based solutions have shown an increased interest in the commercial domain to offer a layered set of security services to the cloud users for specific products like IBM Tivoli products or HP system Insights. Other notable hypervisor-based IDS schemes in the literature include detecting kernel module integrity on VMs based on windows operation systems [Ahmed et al. 2012] or using signatures generated pre-instantiation of the VM. Monitoring system calls between the VMs and its host kernel using Hidden Markov Model to classify VM behavior was proposed in [Alarifi et al. 2013]. Most recently, an efficient VMI scheme is introduced in [Kourai et al. 2014]. In this work, we expanded the VMI capabilities for anomaly-based intrusion detection in application runtime.

6. CONCLUSION

Traditional anomaly-based intrusion detection schemes fall short when confronting the challenges of dynamic, on-demand, and variable deployment and service models inherent in the cloud computing paradigm. We proposed a simple yet effective taxonomy to aid on determining IDS positioning for maximum detection in such environments. A deployment decision matrix with a VMI-based IDS solution is introduced and a realistic use case scenario is used to illustrate the proposed scheme. We investigate mapping additional IDS solutions for SaaS and PaaS deployment models and true/false and negative/positive alarm rates in our future work.

Acknowledgement

This publication was made possible by NPRP grant # [7-1113-1-199] from the Qatar National Research Fund(a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

Authors would like to thank Jim Hanna, Mike Halick, and Dr. Mark Linderman at AFRL for their continuous support and guidance, and Prof. Chris Clifton at Purdue for his valuable discussions and feedback of the initial design of this project. Special thanks to Matthew Paulini at AFRL for his editorial proofing contributions.

REFERENCES

Ahmed, I., Zoranic, A., Javaid, S., and Richard, G.G. 2006. ModChecker: Kernel Module Integrity Checking in the Cloud Environment. In *Proceedings of the 41st International Con fence on Parallel Processing Workshops (ICPPW), vol., no., pp.306,313, 10-13.*

Alarifi, S., and, Wolthusen, S. 2013. Anomaly Detection for Ephemeral Cloud IaaS Virtual Machines. In *Lecture Notes in Computer Science, Volume 7873, 2013, pp 321-335.*

Averest Research Group 2013. Enterprise Cloud Adaptation Survey 2013: Summary of Results. In *http://www.everestgrp.com/wp-content/uploads/2013/03/2013-Enterprise-Cloud-Adoption-Survey.pdf.*

Bhaduri, K., Das, K., and Matthews, B. 2011. Anomaly Detection for Ephemeral Cloud IaaS Virtual Machines. In *Proceedings of ICDMW, IEEE 11th International Conference on Data Mining Workshops. IEEE, 2011.*

Catania, C., and, Garino, C. 2012. Automatic Network Intrusion Detection: Current Techniques. In *Computer and Electrical Engineering. 38 (2012) 1062 1072.*

Frank, D., Knahl, M., Reich, C., and Clarke, N. 2013. Anomoly Detection In IaaS Clouds. In *IEEE International Conference on Cloud Computing Technology and Science.*

Garfinkel T., and, Rosenblum M. 2003, A virtual Machine-based Architecture for Intrusion detection. In *Network and Distributed System Security Symposium, Sandiego, California USA, 2003.*

Goldszmidt, M., Woodard, D. and Bodik, P. Real-time identification of performance problems in large distributed systems. In *Machine Learning and Knowledge Discovery for Engineering Systems Health Management. 2011.*

Kourai, K., and, Nakamura K. Efficient VM Introspection in KVM and Performance Comparison with Xen. In *Proceedings of IEEE 20th Pacific Rim International Symposium on Dependable Computing (PRDC).*

Liao, H., sc Lin, C., Lin, Y., and, Tung, K. 2013. Intrusion Detection System: A comprehensive Review, In *Journal of Network and Computer Applications 36 (2103) 16-24.*

Mazzariello, C., Bifulco, R., and, Canonico, R. 2010. Integrating a Network IDS into and Open Source Cloud Computing Environment. In the *6th International Conference on Information Assurance and Security. 2010*

Modi, C., Dhiren P., Bhavesh B., Hiren P., Avi P., and, Muttukrishnan R. 2013. A survey of Intrusion Detection Techniques in Cloud. Journal of *Network and Computer Applications 36, no. 1 (2013): 42-57.*

Smith, D., Guan, Q., and, Fu, S., 2010. An Anomaly Detection Framework for Autonomic Management of Compute Cloud Systems. In the *34th Annual IEEE Computer Software and Applications Conference Workshops.*

Subashini, S., and, Kavitha, V. 2011, A survey on Security Issues in Service Delivery Models of Cloud Computing. In *Journal of Network and Computer Communication. 34.1 (2011): 1-11.*

Wang, C., Talwar, V., Schwan, K., and Ranganathan, P. 2010. Online Detection of Utility Cloud Anomalies Using Metric Distribution. In *IEEE/IFIP Network Operations and Management Symposium (NOMS) 2010: Mini Conference.*

Xu, W., Huang, L., Fox, A., Patterson, D., and, Jordan, M., 2009. Detecting Large-Scale Problems by Mining Console Logs. In the *22nd Symposium of Operating Systems Principles, (SOSP09).*

Granville, K., 2015. 9 Recent Cyberattacks Against Big Businesses. `http://www.nytimes.com/interactive/2015/02/05/technology/recent-cyberattacks.html?_r=0`.

Noor (Norman) Ahmed is a PhD candidate with the Department of Computer Science at Purdue University and a Computer Scientist at the Air Force Research Laboratory's Information Directorate (AFRL/RI). His research interests include: Security in Cloud Computing with special emphasis on Anomalous-based Intrusion Detection on virtualized cloud platforms; Security and QoS in Service Oriented Architectures; Semantic Computing; Reliability and Resiliency in Distributed Systems; and Moving Target Defense (MTD). Mr. Ahmed's PhD thesis is focussed on developing a disruption-resilient framework for distributed systems using MTD techniques.

Dr. Bharat Bhargava is a professor of computer science at Purdue University and an IEEE fellow. His research work deals with the security and privacy issues in Service Oriented Architectures and Cloud Computing, and secure Internet-scale routing and mobile networks. Professor Bhargava is the editor-in-chief of four journals and serves on over ten editorial boards of international journals. Professor Bhargava is the founder of the IEEE Symposium on Reliable and Distributed Systems, IEEE conference on Digital Library, and the ACM Conference on Information and Knowledge Management.
Professor Bhargava has graduated the largest number of PhD students in CS department and has won many awards, these include: six best paper awards, technical achievement award, and golden core award from IEEE, An outstanding Instructor Awards from the Purdue chapter of the ACM in 1996 and 1998, and was inducted in the Purdue's Book of Great Teachers. His recent work on Controlled Data Dissemination in untrusted environments under attacks received the first place in Purdues CERIAS security center symposium 2015.