Behavioral Modeling for Suspicious Process Detection in Cloud Computing Environments

Andrey Dolgikh, Zachary Birnbaum, Yu Chen, Victor Skormin

Dept. of Electrical and Computer Engineering, Binghamton University, Binghamton, NY 13902, USA
{adolgik1, zbirnba1, ychen, vskormin}@binghamton.edu

Abstract- **One of the defining features of cloud computing, multi-tenancy provides significant benefits to both clients and service providers by supporting elastic on-demand resource provisioning and efficient resource allocation. However, this architecture also introduces additional security implications. Client virtual machine (VM) instances running on the same physical machine are susceptible to side-channel and escape-to-hypervisor attacks. Timely detection/mitigation of intrusive behaviors of malicious processes using signature based intrusion detection technologies or system call level anomaly analysis due to high false alarm rate presents a challenging task. In this work, a behavioral modeling scheme is proposed to detect suspicious processes on the highest semantic level. Our preliminary results have validated the effectiveness and efficiency of this novel approach.**

Keywords: Cloud Computing Security, Multi-Tenancy, Behavioral Modeling, Suspicious Process Detection.

I. Introduction

Cloud computing is a new computing paradigm that possesses many favorable features such as transparent service, good scalability and elasticity, support for the pay-as-you-go service model, and omni-accessibility [2]. This paradigm not only enables users to enjoy the convenient, versatile, and efficient services, but also relieves the burden of maintenance. One of the defining characteristics of cloud computing is multi-tenancy: a hardware/software architecture in which a single server provides services to multiple clients through virtual, rather than physical, partitioning. Multi-tenancy provides significant benefits both to clients, through elastic on-demand resource provisioning, and to service providers, through more efficient resource allocation.

Multi-tenancy architecture also introduces additional security implications. For example, client virtual machine (VM) instances running on the same physical machine are susceptible to side-channel [7], [10], [13] and escape-to-hypervisor attacks [9]. Success of such types of attacks would compromise confidentiality of user information including data and operations. Researchers have demonstrated that modern complex malware can successfully exploit multi-tenancy for information stealing, even on modern highly dynamic and unpredictable, symmetric multiprocessing (SMP) platforms [7], [10], [13].

Therefore, timely detection and mitigation of such attacks becomes imperative to both cloud service providers and clients. Failure to protect user data confidentiality violates the interests of service providers by jeopardizing their reputation and leading to financial losses. From the clients' perspective, lack of trust in the cloud environment necessitates the capability of monitoring the status of mission critical applications or processes that have been outsourced to the cloud in a transparent manner.

The most popular malware detection schemes are still dominated by the binary signature-based approach. Although it has many practical advantages, this technology can be evaded by using automatic tools such as code packers and metamorphic engines. This leads to a dead end due to exponentially growing database of binary signatures. In addition, it is inherently incapable of addressing targeted, zero-day malware attacks as they are not represented by a binary sample in the database.

Behavioral analysis offers a more promising approach to malware detection since behavioral signatures are more obfuscation resilient than the binary ones. Indeed, changing behavior while preserving the desired malicious functions of a program is much harder than changing only the binary structure. More importantly, to achieve its goal, malware usually has to perform some system operations (e.g. registry manipulation). Since system operations can be easily observed and are difficult to obfuscate or hide, malicious programs are more likely to expose themselves to behavioral detection. Consequently, while database of specific behavioral signatures is still to be utilized, its size and rate of increase are significantly lower than those in the case of binary signatures.

In this paper, we propose a novel approach to monitor the execution status of user application programs and detect suspicious processes in cloud servers using behavioral modeling. Since our function module can be easily integrated in the hypervisor to monitor all OS level system calls in all virtual machines, this technology is ideally aligned with the cloud computing environment. It has been tested on top of our computing cluster and demonstrated very encouraging results.

The remainder of the paper is organized as follows. Section II briefly discusses the proposed approach for detecting suspicious processes in cloud servers. Section III introduces the specifics of our behavioral modeling technology and some preliminary results. Various aspects of

its application and deployment in cloud environment are discussed in Section IV, and Section V concludes this paper describing our ongoing work.

II. Related Work

Let us consider a layered hierarchy of typical virtual machine-based cloud server. Each host OS is separated from other VMs and authenticated by a TPM (trusted platform module) on the physical machine. A hypervisor in the virtualization layer is constantly monitoring activities of different virtual machines and their OSs. In order to protect user application executions, the normal approach to monitor the behavior of programs running on cloud servers is to integrate monitoring or logging modules into a higher privileged software level, such as hypervisors or micro-kernels.

Wang et al. [11] proposed HyperCheck, a hardware-assisted monitor for hypervisor integrity protection. Leveraging the CPU System Management Mode (SMM), HyperCheck takes a complete view snapshot of the machine, which includes the entire memory and CPU registers, and transmits the snapshot to a remote analyzer. Alerts will be generated if suspicious changes are detected in registers or memory snapshot. While the HyperCheck protects VM integrity, the inablility to detect dynamic data tampering makes it unsatisfactory in cloud computing environment. Meanwhile, the time interval between two consecutive snapshots is determined by the SMM NIC data rate [11], making HyperCheck vulnerable to transient attack [3].

There are several other hardware based hypervisor or VM integrity monitoring schemes similar to HyperCheck. Copilot [6] employed a special PCI device to periodically poll the physical memory of the machine and send it to an administration station. The inconsistency between the code running on the PCI device and the code running on host system prevents the PCI device from accessing the CPU state. This gap opens a door for some attacks [8].

Hyperguard [12] is also using SMM of the x86 CPU to monitor the integrity of the hypervisers. However, compared to HyperCheck, which transmits the state snapshot to remote analyzer, the performance of HyperGuard is being decreased when the system is busy as all processing tasks are conducted locally. Such a weakness also makes HyperGuard vulnerable to Denial of Service (DoS) attacks [11].

Researchers have also developed TPM based solutions that provide a minimum Trusted Code Base (TCB), such as Flicker [5], TrustVisor [4], and Hypersentry [1]. Consisting of hardware, firmware, and software components, the TCB can detect authorized modification to the OS kernels. Flicker and TrustVisor require advanced hardware features, for example, Dynamic Root of Trust Management (DRTM) and late launch. However, in addition to high overhead, they are also vulnerable to the scrubbing attacks [3]. In contrast, Hypersentry depends on an out-of-band channel and it is triggered by a System Management Interrupt (SMI) on the target platform. Also, Hypersentry is not immune to transient attacks [3].

Recently, Houlihan and Du [3] proposed a secure monitoring scheme that is able to prevent transient attack. Besides modifying the Linux auditing daemon to generate a hash of each audit log entry, this scheme integrates SMM and TPM to achieve integrity check and attestable security. According to the experimental results, this scheme achieved reasonably low overhead. One potential issue is that the attestation process uses a remote integrity verification system. The performance and security of entire system are closely related to the security of the communication network.

Our behavioral modeling based suspicious process detection scheme is different from these reported solutions:

i) Comparing to those periodically triggered snapshot taking methods, our scheme provides a real-time, un-interrupted monitoring solution, which is able to catch dynamic data tampering attacks and be immune to transient attacks;

ii) Instead of a very abstract image of registers and memory that is not easily understandable to human operators; our system displays system call data in real time and enables the visualization of previously captured trace files.

iii) Our behavioral analysis module can be flexibly deployed at either the hypervisor layer or VM layer. This implies that not only the service providers, but also individual clients can adopt our technology focusing on specific user applications.

III. Behavioral Modeling

A. Approach

It is challenging to distinguish between malicious and benign operations that are executed by a benign program. Moreover, maliciousness of an executed functionality can often be determined only by its context or environment. Therefore, the challenge of behavioral detection is in devising a good model of behavior which is descriptive enough to allow for discrimination of benign versus malicious programs and can be tuned to the target environment.

Following the anomaly detection paradigm, our scheme consists of two phases: training and detection phase. During the training phase we intercept and analyze a stream of system calls for a sufficient time period to cover the majority of normal system operations. This data is then used to model normal behavior of the system. During the detection phase we observe the stream of system calls and

detect any deviation from the previously defined "model of normal behavior". Most information attacks, even majority of obfuscated ones, exhibit anomalous behavior and is detected as such. Moreover, at the earlier stages of attack some malware performs benign functionalities inconsistent with normal behavior, which is instrumental for attack detection.

Our behavior modeling approach operates on the highest level of behavioral semantics, the level where behavior is directly associated with the specific goals of the software developer.

B. Behavioral Representation

The behavior refers to the actions performed by the program with respect to its environment. The environment of any program in a computer system is controlled and managed by operating system kernel. Windows OS organizes the environment using OS objects: file, memory section, thread, mutex, etc. For a program to sense or modify the environment a request to kernel must be issued. The request to OS is issued by invoking a system call with desired parameters. Monitoring system calls along with parameters provides a system-wide view of behavior.

In order to determine the behavior and specifically the anomalous behavior changes, a vertex-edge labeled graph model is proposed that allows us to capture the normal structure of operations over OS objects. Each vertex is corresponding to a system call, and any two vertices are connected by an edge if they share a parameter. The graph can be built from the stream of system calls.

For example, calls S1, S2 in Figure 1(a) have a common parameter C. In the resulting graph in Figure 1(b) nodes corresponding to calls S1, S2 are connected with the directed edge C. Nodes S2, S3 are connected with an edge labeled C.

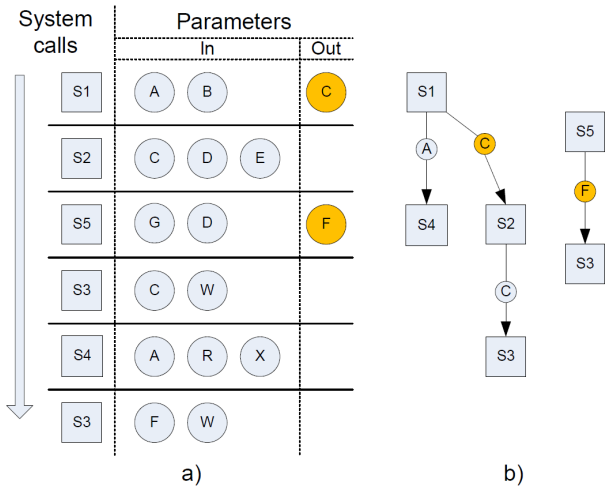

Figure 1. Illustration of Behavior Graph Construction.

It is worth noting that the set of system calls is small and well known. The set of all possible values of parameters of system calls is unknown beforehand and very extensive. Fortunately, this does not pose a difficult problem since majority of the important parameters take values from a small subset.

C. Experimental Results

To demonstrate the implemented technology, we deploy an attack against vulnerable programs to show that our system is capable of detecting attacks. The vulnerable host under attack is represented by the Metasploitable Virtual Machine (VM) [14]. The Metasploitable VM is an operating system package which comes preconfigured with many vulnerable servers and services. It is an ideal playground for testing intrusion detection systems. The Metasploit framework is used to attack the vulnerable host [15]. Metasploit comes preconfigured with tools for network identification, vulnerability scanning and penetration testing. It provides means to find exploitable vulnerabilities and mount attacks against them. Backtrack Linux is selected to host Metasploit [16].

Gephi graph visualization software [17] is adopted and

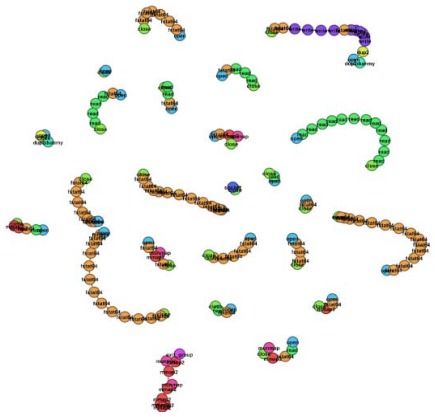

Figure 2. Component Database for Samba Server.

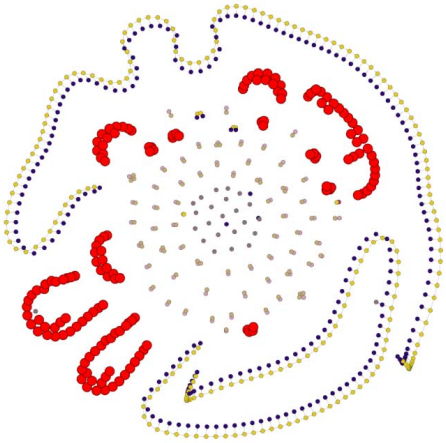

Figure 3. Detected Anomalies (enlarged red nodes).

allows us to show in real time system call data and visualize previously captured and saved trace files. The use of this facility enables us to observe both benign processes and malicious exploits developing in real time. It should be noted that the particular configuration and location of the individual graph components is the function of the visualization software and is irrelevant.

The Metasploitable VM comes with many exploitable servers, including an easily exploited SAMBA server. SAMBA is responsible for file and print sharing services and is found in most industrial or enterprise environments. In this study we expose the SAMBA server process to a typical load such that copy, move, and delete. Once the analysis of the SAMBA trace is complete a normalcy profile reflecting normal SAMBA operation contained 24 components and 118,507 nodes is established as shown by Fig. 2.

The Metasploit Framework includes an exploit for SAMBA versions 3.0.20-3.0.25rc3 allowing an attacker to execute arbitrary commands. Figure 3 features graph trace for Samba under Metasploit attack. Enlarged red nodes correspond to components not found in our normalcy profile. These components manifest additional anomalous functionality not observed before on the Samba server.

IV. Detection of Suspicious Processes in Cloud

A. Threat Model

Typically, attacks against computing devices in a cloud platform, including servers, personal computers, laptops, and mobile devices, can aim at different levels: software, hardware and service providers. Potential threats can be divided into six categories similar to Open Mobile Terminal Platform [18] and Open and Secure Terminal Initiative (OSTI) [19]:

i) Software modification threats: Logical threats that aim to modify software of user equipment.

ii) Software opportunistic threats: Threats that take advantage of weaknesses in the execution of software on the user equipment while exposing sensitive information.

iii) External hardware threats: Threats and breaches that can be introduced via ports or peripherals of the physical system.

iv) Terminal intrusive hardware threats: The treat of an attacker physically probing exposed circuitry of printed circuit boards of the physical hardware including removal of components for offline attacks.

v) Component invasive hardware threats: Threats and attacks that affect the integrity of physical components including integrated circuits, memory and printed circuit boards.

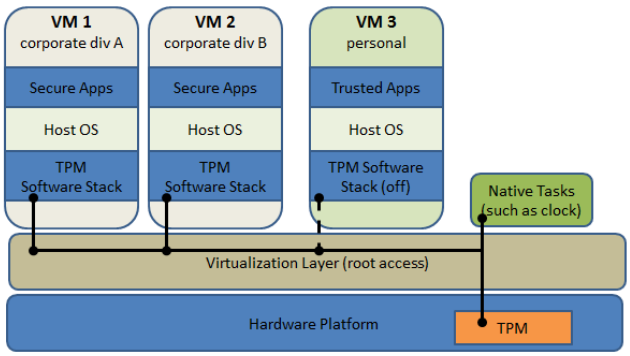

Figure 4. Virtual Machine Platform on Cloud Servers.

vi) Cloning, component replacement and addition: Threats that are introduced with the replacement of a component in the physical equipment such as integrated circuits, memory and printed circuit boards as well as copies of the physical equipment.

By monitoring system calls of processes running in each virtual machine, the proposed behavioral modeling scheme aims at addressing the software-oriented threats in categories i) and ii).

B. Behavioral Modeling Deployment

We envision a virtual machine-based OS system as shown in the Fig. 4. This system separates user applications according to different job responsibilities. Each host OS is separated from other VMs and authenticated by a TPM (trusted platform module) on the physical machine. A hypervisor in the virtualization layer is constantly monitoring activities through different virtual machines and their OSs.

For the personal apps, a user can decide if he/she wants to turn on TPM software for protecting own privacy. The hypervisor is able to create partitions within the resources of the system. The operating systems can then be isolated in a way that individual OS's have isolated access to only their own resources or files, or even completely encapsulating an operating system allowing portability.

Service providers can embed the system call monitoring and analysis module into the hypervisor, where all the virtual machines are on the radar. In contrast, for an individual user the behavioral monitoring function can be integrated in the TPM software, which lies below the OS layer such that all processes running on the assigned virtual machine are supervised.

V. Conclusions

We proposed a novel approach to detect suspicious processes in computing devices in the cloud environment. This behavioral analysis scheme offers an obfuscation resilient solution to malware detection. Particularly, it can

be flexibly deployed to satisfy the requirements of service providers and individual clients.

This position paper reports our preliminary results that have validated the concept. Currently, a prototype is being implemented on a mobile cloud computing platform at Binghamton University. Experiments will be conducted to investigate the performance, robustness, and design tradeoffs both on the servers and mobile devices, e.g. Nexus 7 tablets and smartphones.

References

[1] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, N. C. Skalsky, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," in *Proc. of the 17th ACM Conference on Computer and Communications Security*, pp. 38-49, 2010.

[2] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, "NIST special publication 800-146, draft cloud computing synopsis and recommendations," *National Institute of Standards and Technology*, 2011.

[3] R. Houlihan, and X. Du, "An Effective Auditing Scheme for Cloud Computing," in *Proc. of IEEE GLOBECOM 2012*, Anaheim, CA, USA, Dec. 2012.

[4] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "TrustVisor: Efficient TCB reduction and attestation," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

[5] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki, "Flicker: an execution infrastructure for TCB minimization," in *Proc. of the ACM European Conference on Computer Systems* (EuroSys), March/April 2008.

[6] N. L. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot - a coprocessor-based kernel runtime integrity monitor," in *Proc. of the 13th USENIX Security Symposium*, pp. 13, 2004.

[7] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," In *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199-212. ACM, 2009.

[8] J. Rutkowska, "Beyond the CPU: Defeating Hardware Based RAM Acquisition Tools," *Blackhat*, February 2007.

[9] J. Szefer, E. Keller, R.B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," *ACM CCS*, 2011.

[10] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M.M. Swift, "Resource-freeing attacks: improve your cloud performance (at your neighbor's expense)," *ACM CCS*, 2012.

[11] J. Wang, A. Stavrou, and A. K. Ghosh, "HyperCheck: A hardwareassisted integrity monitor," in *Proc. of the 13th International Symposium on Recent Advances in Intrusion Detection* (RAID10), September 2010.

[12] R. Wojtczuk and J. Rutkowska, "Xen 0wning trilogy," in *Proc. Black Hat conference*, 2008.

[13] Y. Zhang, A. Juels, M.K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," *ACM CCS*, 2012.

[14] Metasploitable, http://sourceforge.net/projects/metasploitable/.

[15] Metasploit, http://www.metasploit.com/.

[16] Backtrack-Linux, http://www.backtrack-linux.org/.

[17] Gephi, Available: https://gephi.org.

[18] Open Mobile Terminal Platform Alliance, "OMTP Approved Deliverables," 2010, http://www.omtp.org/approved.html.

[19] Intel and NTT DoCoMo, "Open and Secure Terminal Initiative (OSTI) Architecture Specification V1.0," 2006.