Cloud Security Auditing based on Behavioral Modeling

Zachary Birnbaum, Bingwei Liu, Andrey Dolgikh, Yu Chen, Victor Skormin
Dept. of Electrical and Computer Engineering
Binghamton University
Binghamton, NY USA
{zbirnba1, bliu, adolgik1, ychen, vskormin}@binghamton.edu

Abstract— **Multi-tenancy is one of the most attractive features of cloud computing, which provides significant benefits to both clients and service providers by supporting elastic, efficient, and on-demand resource provisioning and 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. The timely prevention of intrusive behavior and malicious processes using signature based intrusion detection technologies, or system call level anomaly analysis is a very challenging task due to a high rate of false alarms. In this work, a behavioral modeling scheme is proposed to audit the behaviors of client VMs and to detect suspicious processes on the highest semantic level. Our preliminary results have validated the effectiveness and efficiency of this novel approach.**

Keywords-Cloud Security Auditing (CSA), 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, supporting the pay-as-you-go service model, and omni-accessibility [3]. This paradigm not only enables users to enjoy 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 [15, 19, 24] and escape-to-hypervisor attacks [18]. The success of such attacks will compromise the confidentiality of user information including data and operations. Researchers have demonstrated that modern complex malwares can successfully exploit multi-tenancy for information stealing, even on modern highly dynamic and unpredictable, Symmetric Multiprocessing Platforms (SMP) [15, 19, 24].

An efficient Cloud Security Auditing (CSA) mechanism is highly desired because of the un-trustworthy nature of the cloud computing environment. On one hand, it is critical for service providers to catch runtime attacks launched by malicious users; on the other hand, an attestable hypervisor will reduce clients' security concerns. In addition, a well maintained auditing log provides sufficient evidence when a disputation arises between service provider and clients, or among clients.

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, harming their reputation and leading to eventual financial losses. From the clients' perspective, lack of trust in the cloud environment necessitates the capability to monitor 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 including code packers and metamorphic engines, and leads to a dead end due to an exponentially growing database of binary signatures. In addition, it is inherently incapable of addressing targeted, zero-day malware attacks not represented by a binary sample in the database.

Behavioral analysis offers a more promising approach to malware detection because behavioral signatures are more obfuscation resilient than 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 a 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 behavior modeling based CSA scheme. This technology is well aligned with the cloud computing environment and the behavior monitoring function module can be flexibly deployed either in the hypervisor or client VM level. The hypervisor level implementation enables the service provider to monitor all OS level system calls in all virtual machines. In contrast, deployment in the OS kernel of client VMs allows each

client to audit the working status of the assigned VM, the execution of user application programs, and the detection suspicious processes. A private cloud computing platform has been built and Gephi graph visualization software [25] utilized, which allows us to show system call data in real time and visualize previously captured and saved trace files. The results are very encouraging.

The rest of this paper is organized as follows. Section II gives a brief survey of the recent achievement in cloud auditing. Section III discusses the threat model in the cloud computers. Section IV introduces the principle of our behavioral modeling based cloud auditing technology. The experimental results and some discussions are reported in Section V, and Section VI concludes this paper with our ongoing work.

II. RELATED WORK

The reported CSA schemes can be divided into two categories: data security auditing, which keeps track of accesses and operations on stored data, and software security auditing, which focuses on suspicious activities monitoring and logging.

A. *Auditing for Data Integrity*

Juels and Kaliski [9] proposed a sentinel-based scheme which they called a Proof Of Retrievability (POR) for archived files. The major computation work happens in the client side. The file F is applied an error-correction code and encrypted before pre-defined number of sentinels are embedded. The new file with sentinels is then permutated and sent to the server. The advantage of this protocol is the server cannot delete any block and pretend that it possess the whole file because it doesn't know which blocks are sentinels. The down side, of course is that the client can only challenge the server limited number of times. This fatal limit makes the protocol not suitable for services that require flexible verification and frequent modifications.

Ateniese et al. [1] first proposed the concept of Provable Data Possession (PDP), a statistical scheme based on homomorphic verifiable tags. In order to achieve 95% confidence that the server possesses the client's data, the client only needs to check 300 blocks, while checking 460 blocks is sufficient to guarantee 99% confidence.

Shacham and Waters [17] proposed two extended POR schemes, with private and public verifiability respectively. The main advantage of SW PORs is the unlimited number of queries. The private verification scheme, based on Pseudo-Random Functions (PRFs), has the shortest response of any POR scheme (20 bytes) with the cost of a longer query. The second scheme used short signatures introduced by Boneh, Lynn and Shacham (BLS) [4] to verify the authentication of data in a remote server, hence assuring secure public verifiability. At an 80-bit security level, this scheme has the shortest query (20 bytes) and response (40 bytes) of any POR scheme.

Although both schemes almost followed the same framework, SW-PORs are more robust than PDP. Each file block is further divided into s sections and s random values are chosen for the generating and checking proof. This strategy increases the security and computation complexity. In addition, a file tag has been introduced, which is verified in each audit phase. This file tag includes random values for a check proof algorithm, the number of blocks, as well as cryptographic keys or other random values as a file identifier.

While PORs and PDPs mainly focus on static, archival storage, Dynamic Provable Data Possession (DPDP) schemes [5] have been suggested to meet the requirements of dynamic data access applications. In dynamic access, there are data updating transactions such as inserting, modifying, or deleting blocks or files. The DPDP schemes are able to verify file possession under these situations.

Cloud servers are shared by multiple users, thus multiple verifications for different files often happen. Taking advantage of aggregation property of bilinear signature scheme, extended verification schemes have been proposed to meet the challenge [20, 22]. Feng et al. [6] proposed a non-repudiation protocol that enables a fair, secure data transmission procedure by allowing peers exchange messages and non-repudiation evidence. A Trusted Third Party (TTP) is introduced, which plays the role of an arbiter when disputation happens if the sender fails to obtain non-repudiation evidence.

B. *Auditing for Suspicious Software Activities*

Figure 1 illustrates a layered hierarchy of a typical VM based cloud server. Each guest OS is separated from other VMs and authenticated by a Trusted Platform Module (TPM) on the physical machine. A hypervisor in the virtualization layer is constantly monitoring activities through different virtual machines and their OSs. In order to protect user application executions, a normal approach of CSA solutions is integrating, monitoring, or logging modules into a higher privileged software level, such as hypervisors or micro-kernels.

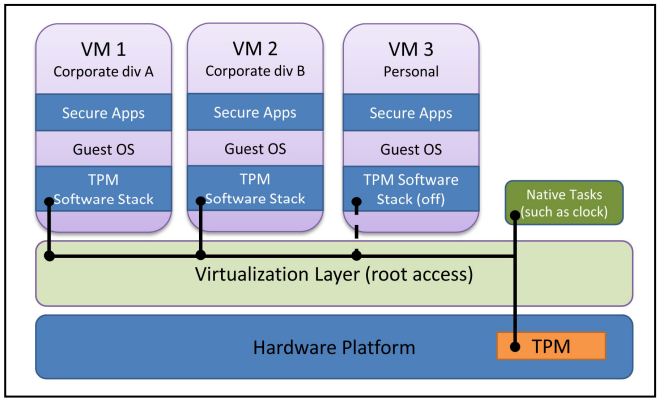

Figure 1. Virtual Machine Platform on Cloud Servers.

Wang et al. [21] proposed HyperCheck, a hardware-assisted monitor for hypervisor integrity protection. Leveraging the CPU System Management Mode (SMM), HyperCheck takes snapshot of a complete view of the machine, which includes the entire memory and CPU registers, and transmits the snapshot to a remote analyzer. An alert will be generated if suspicious changes are caught in registers or memory snapshots. While the HyperCheck

protects VM integrity, the lack of capacity for detecting dynamic data tampering makes it unsatisfactory in the cloud computing environment. The time interval between two consecutive snapshots is determined by the SMM NIC data rate [21]. This makes HyperCheck vulnerable to transient attack [7].

There are several other hardware based hypervisor or VM integrity auditing schemes similar to HyperCheck. Copilot [14] employed a special PCI device to poll the physical memory of the machine and send it to an administration station periodically. The inconsistency between the code executing on the PCI device and the system prevents the PCI device from accessing the CPU state. This opens a door for some attacks [16]. Hyperguard [23] also uses SMM of the x86 CPU to monitor the integrity of the hypervisors. However, compared to HyperCheck, which transmits the state snapshot to remote analyzer, the performance of HyperGuard is 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 [21].

Researchers have also developed TPM-based solutions that provide a minimum Trusted Code Base (TCB), such as Flicker [12], TrustVisor [11], and Hypersentry [2]. 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. They have high overhead and also vulnerable to scrubbing attacks [7]. In contrast, Hypersentry depends on an out-of-band channel and is triggered by a System Management Interrupt (SMI) on the target platform. Hypersentry is not immune to transient attacks [7].

Recently, Houlihan and Du [7] proposed a secure auditing scheme that is able to prevent transient attacks. 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. This scheme achieved low overhead according to the experimental results. One potential issue is that the attestation process uses a remote integrity verification system. The performance and security of the entire system are closely related to the security of the communication network.

Our behavioral modeling based CSA scheme is different from these reported solutions in three aspects:

- 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;
- Instead of an abstract image of registers and memory that is not easily understandable to human operators; our system will show system call data in real time and visualize previously captured trace files.
- Our behavioral analysis module can be flexibly deployed at either the hypervisor or VM layer. This implies that not only the service providers, but individual clients can adopt our scheme focusing on specific interested user applications.

III. 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 [13] and Open and Secure Terminal Initiative (OSTI) [8]:

- Software modification threats: Logical threats that aim to modify software of user equipment.
- Software opportunistic threats: Threats that take advantage of weaknesses in the execution of software on the user equipment while exposing sensitive information.
- External hardware threats: Threats and breaches that can be introduced via ports or peripherals of the physical system.
- 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.
- Component invasive hardware threats: Threats and attacks that affect the integrity of physical components including integrated circuits, memory and printed circuit boards.
- 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 on each VM, the proposed behavioral modeling based CSA scheme aims to address the software-oriented threats in categories i) and ii).

IV. BEHAVIORAL MODELING BASED CLOUD SECURITY AUDITING

The main goal of the behavior modeling based CSA scheme is to identify and report malicious behavior in cloud computers. The behavior refers to the actions performed by the programs with respect to their environment. While the complete list of malicious actions is quite extensive a substantial part of the list can be legitimate depending on the environment.

Information sharing/leaking can be legitimate for authorized programs or malicious for unauthorized ones. File hosting functionality is legitimate in most cases but can be malicious depending on the environment and file type. As one may see, it is hard to define the malicious behavior. Very often it depends on the context and the original intent of the software designer. Therefore, the program behavior should be formalized from the point of view of the original intent of the program writer. The behavior should also be actualized to the current environment where the program operates.

Consequently, the challenge of behavioral detection is in devising a good model of behavior which is descriptive enough to allow for the discrimination between benign and malicious programs and which can be tuned to the target environment.

A. *Behavioral Representation*

The environment of any program in the computer system is controlled and managed by the operating system kernel. For example, the Windows OS organizes the program environment using well known objects including: file, memory section, thread, mutex. If the process needs to sense or modify these objects it has to request the OS to do so by making a system call. However, working with system calls unnecessarily complicates the task of application developers. Windows subsystem libraries offer more friendly, well documented APIs that provide programs with the ability to interact with OS objects. Nonetheless, these APIs still execute system calls. Therefore, monitoring the system calls is the ideal way to observe a program's behavior. System call monitoring provides a reliable and in many cases incorruptible source of raw behavior-carrying data for functionality recognition.

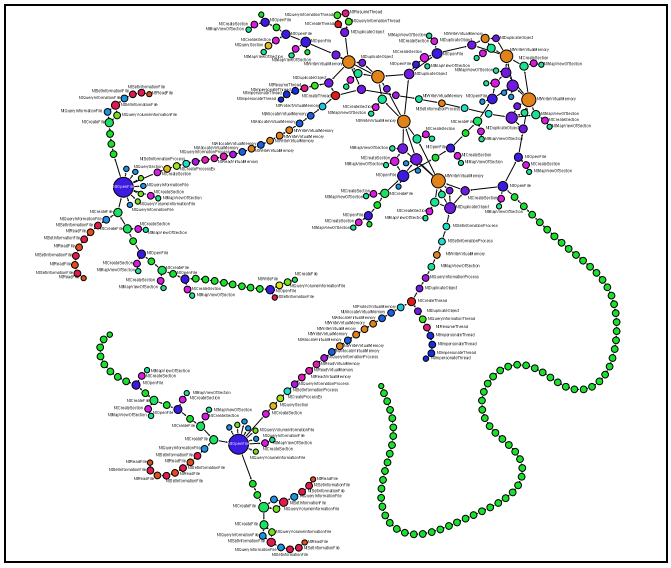

Figure 2. System call graph for the kernel32.CreatProcess.

The behavioral representation at the level of system calls may vary widely depending on the implementation. Moreover because system calls lie at the bottom of the API hierarchy it may lead to very complex and verbose manifestations of the functionalities. One Windows subsystem API call may translate into hundreds of system calls. For example, CreateProcess from kernel32.dll may expand into a sequence of hundreds of system calls, whose full graph is presented in Figure 2. It features more than 372 system calls and 399 links between them. Clearly there is no easy way for an expert to recognize functionality from the raw system call traces.

In order to determine the behavior and detect anomalous changes in the behavior, a vertex-edge labeled graph model is proposed that allows us to capture the normal structure of operations over OS objects. Each vertex corresponds 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 3(a) have a common parameter C. In the resulting graph in Figure 3(b) nodes corresponding to calls S1, S2 are connected with the directed edge C. Nodes S2, S3 are connected with an edge labeled C.

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 because the majority of the important parameters take values from a small subset.

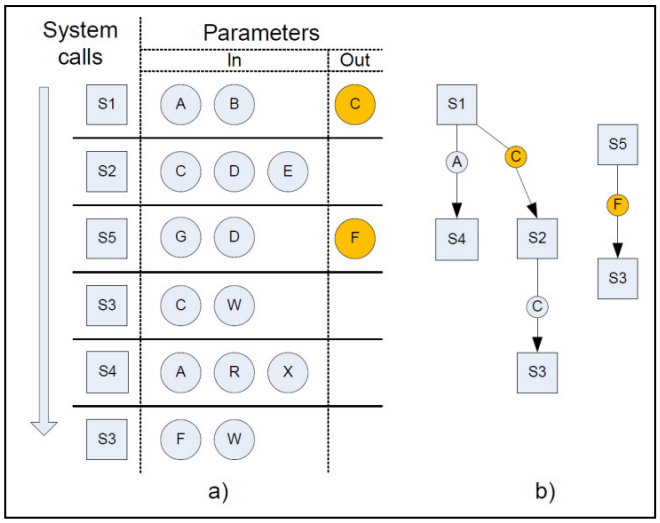

Figure 3. Illustration of Behavior Graph Construction.

B. *Behavioral Auditing based on Functionality Detection*

As illustrated in Figure 2 and Figure 3, the behavior representation at the level of system calls does not offer auditors or system administrators a convenient means for the timely assessment of hypervisor and client VMs operation. Therefore, the concept of functionality is introduced to formalize a program's behavior. A functionality can be understood as a set of operations serving nontrivial tangible goals in the digital environment. Then, the process of CSA can be accomplished in two stages.

The first stage is the detection of implemented functionalities, which actually are behavioral patterns. This stage is independent of the context and properties of the monitored computer system. At this point we only reveal what types of functionalities are executed by the programs in the system.

The second stage tags sets of functionalities as malicious or benign in the current environment. At this stage, functionalities are attributed to malicious or benign domains using current knowledge of the context and execution environment. This stage takes into account contextual information to classify the set of detected functionalities as benign or malicious.

The core of our functionality monitor is built on the Colored Petri Net (CPN) [10] simulator, which consists of several components:

- **System call interposition driver** wiretaps on all data passing the kernel boundary by means of system calls. Accumulated data is sent to user mode for analysis.
- **Low level CPN engine** is part of the interposition driver. It performs assembly of more complex events from the stream of system calls.
- **User mode data parser** receives serialized data from the kernel-mode driver and transforms the data into more usable form.
- **User mode API interposition library** wiretaps on API calls. It cross-checks with driver provided data and accelerated testing of new functionally signatures.
- **Event fusion library** accepts events from different sources and routes it to proper places in the CPN simulator.
- **CPN simulator** is the core of functionality detection. It simulates CPNs with data provided by kernel mode interposition driver and API interposition library.
- **Graph builder** assembles graphs out of the data obtained from the system call interposition driver.

With the functionality monitor, a profile of normal program operations can be constructed. Such types of functionalities possess high discriminative power against malicious software. Therefore, the functionality profiles are ideal for behavior based auditing system that prevents intrusions and timely detects suspicious processes.

Following the anomaly detection paradigm, our CSA system can be built in two phases: the training phase and the 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 "normal" behavior model. Most information attacks, even the majority of obfuscated ones, exhibit anomalous behavior and is detected as such.

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

C. *Behavioral Modeling based Cloud Security Auditing*

We envision a virtual machine-based OS system as shown in the Figure 1. This system separates user applications according to different job responsibilities. Each guest OS is separated from other VMs and authenticated by a TPM on the physical machine. A hypervisor in the virtualization layer is constantly monitoring activities through different VMs and their OSs.

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

Service providers can embed the system call monitoring and analysis module into the hypervisor, where all VMs are monitored. At this point monitored system calls are assembled into functionalities representing facets of system behavior. These functionalities can be used to verify that the user software exhibits expected behavior.

TABLE I. HADOOP FUNCTIONAL PROFILE

	Node 1	Node 2	Node 3	Node 4	Idle
Trace Length(S)	66034	63331	63193	63175	14334
# System Calls	23.4e6	13.5e6	13.4e6	13.4e6	9.4e6
# Functionalities	102	93	98	100	21

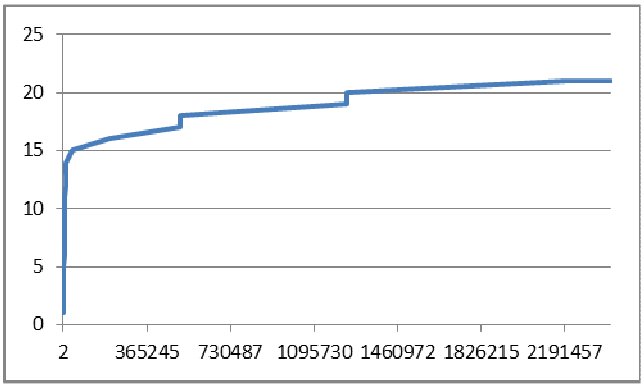

Figure 4. Stabilization of the normalcy profile size.

V. EXPERIMENTAL STUDY

A proof-of-concept prototype built at Binghamton University consists of a small scale cloud platform and a software suite capable of capturing functionalities. This section reports on the deployment of our behavioral modeling based CSA scheme and some preliminary experimental results.

To confirm that our CSA scheme is capable of extracting a stable functional normalcy profile by processing a long continuous stream of system calls data, we monitored an idling cloud node. The results are presented in Figure 4, where the Y-axis illustrates the total number of distinct functionalities captured into the normalcy profile, and the X-axis represents the number of system calls processed. With time, graphs tend to flatten thus showing that the normalcy profile converges to a certain size, which represents all functionality exercised by the system.

To confirm that a functional normalcy profile can be used to verify that the user software exhibits expected behavior we ran four Hadoop [27] nodes with similar load (Table 1). From the table one may see that the system under load exhibits additional functionalities. Figure 5 shows that our

functionality detector is able to capture additional system behaviors as they are executed by an application.

Additional tests were run to demonstrate that the same application executes previously identified functionalities. By matching functionalities found in different profiles for the same application one may identify any discrepancies. We compared the Node 1 profile against Node 2, 3 and 4 and found zero discrepancies.

The next stage of the experimentation, currently in progress, involves the deployment of typical attacks observed in the cloud environment, including "low and slow" attacks, and their detection utilizing the described technology.

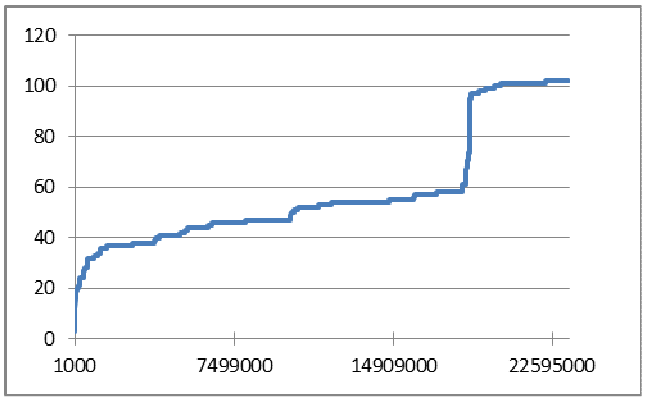

Figure 5. Number of unique functionalities executed by an application.

VI. CONCLUSIONS

We proposed a novel approach to audit programs executing in cloud computers and detects suspicious processes in VMs. 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 paper discusses the rationales and principles of our scheme along with some preliminary experimental studies conducted on a concept proof prototype. The experiment results are very encouraging. Currently, we are extending our private cloud into a mobile cloud computing platform. Further 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 smart phones.

REFERENCES

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security* (CCS), pages 598–609, New York, NY, USA, 2007. ACM.

[2] 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.

[3] 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.

[4] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptology*, 17:297–319, 2004. 10.1007/s00145-004-0314-9.

[5] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 213-222. ACM, 2009.

[6] J. Feng, Y. Chen, and D. Summerville, "A Fair Multi-Party Non-Repudiation Scheme for Storage Clouds," *the 2011 International Conference on Collaboration Technologies and Systems* (CTS 2011), Philadelphia, PA., USA, May 23 - 27, 2011.

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

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

[9] A. Juels and B. S. Kaliski Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security* (CCS), pages 584–597, New York, NY, USA, 2007. ACM.

[10] L.M. Kristensen, S. Christensen, and K. Jensen, "The practitioner's guide to coloured Petri nets," *Int. J. STTT*, (1998) 2: 98-132.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

[17] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Advances in Cryptology* - ASIACRYPT 2008.

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

[19] 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.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, march 2010.

[21] 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.

[22] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *Parallel and Distributed Systems, IEEE Transactions on*, 22(5):847 –859, may 2011.

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