Protecting Private Cloud Located within Public Cloud

Hongli Zhang[1], Lin Ye[1], Xiaojiang Du[2], and Mohsen Guizani[3]
[1]School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China,
{zhanghongli, hityelin}@hit.edu.cn
[2]Dept. of Computer and Information Sciences, Temple University, Philadelphia, PA, USA, dxj@ieee.org
[3]Qatar University, Doha, Qatar, mguizani@ieee.org

***Abstract*—Many studies use cryptographic technologies to protect sensitive data in public cloud. However, these approaches may introduce large overheads. Recently, hybrid cloud started to gain a lot of attentions. A hybrid cloud consists of a private cloud and a public cloud. Hybrid cloud allows users to store sensitive data in their private cloud and hence enables efficient and secure data outsourcing. In this paper, we consider a new hybrid cloud model "Cloud-in-Cloud" (CIC). Our CIC model uses a new architecture to form a hybrid cloud: placing a small number of private computers (i.e., a small private cloud) within a public cloud. The private cloud can be used to store sensitive user data. Furthermore, it is within the public cloud, so the communications between private and public clouds have small overhead. And then we study how to protect a private cloud that locates within a semi-trusted environment. We present two methods that can detect attacks that try to obtain data and information in the private cloud. Our methods are able to efficiently detect physical attacks, such as the cold boot attack and the USB autorun attack. Experimental results show that our methods have small overhead.**

***Index Terms*—hybrid cloud, attack, heartbeat, hook**

I. Introduction

Cloud computing offers dynamically provisioned resources as a service over the Internet. Users can easily scale their applications or services with elastic computing and storage resources in a pay-as-you-go way. Major industrial companies (such as Amazon [1], Google [2], IBM [3] and Microsoft [4]) have started offering cloud platforms. Though low-cost and flexible resource usage of cloud computing brings many benefits, serious concerns on privacy risks are also on the rise and discourage potential customers to migrate their systems into the cloud. First, organizational data usually contains highly sensitive information (e.g., personal contacts, health records, etc.), which needs the highest security assurance that existing clouds cannot offer. Second, recent incidents significantly aggravate customers' worry about their data stored in commercial cloud. For example, a subset of the Amazon elastic block store volumes became unable to service read and write operations [5]. Third, there is no corresponding regulations or standards to clarify the responsibilities of customers and providers when an incident happens. Similar documents like service level agreement only provide a weak service commitment for customers to remedy any unavailability or other failures in Amazon EC2 in the term of service credit [6], which means customers have to pay for their loss. Actually, it is difficult to build a solid trust between cloud service providers (CSPs) and their customers because of potential attacks or system failures even if applying cryptographic techniques.

Recently, hybrid cloud is proposed to make a tradeoff between system performance and security consideration. A hybrid cloud generally consists of a private cloud and a public cloud, which are bound together. Hybrid cloud offers the benefits of multiple deployment models [7]. Hybrid cloud allows customers to keep highly sensitive data in-house while utilizing the cost-effective infrastructure of public cloud. Most existing works on hybrid cloud assume that the private cloud is separate from the public cloud, and they may be far away physically. This may incur long network delay and high cost on network bandwidth due to the long-haul communications, especially for data-intensive computation.

In this paper we propose a different hybrid cloud model: Cloud-in-Cloud (CIC), which let a cloud user to build a small private cloud inside the public cloud. This may be achieved by exclusively renting some servers of the public cloud. Similar practices have already been implemented. For example, NASA signed a contract with Amazon to rent a few servers, which only allows NASA's virtual machines (VMs) to run on these servers.

Compared with traditional IaaS model that mainly provides VMs, in CIC the customers have complete control of the servers. The CIC model avoids vulnerabilities due to the multi-tenant environment, and hence it can provide much better security and privacy protections. Moreover, the communication cost (e.g., delay) between private and public clouds is much smaller because the two clouds are in the same network. Many cloud service providers (CSPs) do not charge communications within the cloud.

On the other hand, it also causes new security concerns by placing the private cloud within the public cloud, which exposes the private cloud servers in a semi-trusted environment. A number of attacks may be launched on the servers. For example, cold boot attack [8] may be used to recovery the confidential information in memory if physical access is allowed. An attacker may leverage physical USB attack [9] to hack into the servers.

In this paper, we present two generic approaches that can protect a private cloud from the cold boot and USB attacks. Our approaches monitor the onlineness of private servers and

978-1-4799-1353-4/13/$31.00 ©2013 IEEE

the status of their hardware such that a remote owner is notified when the attacks happen. Our contributions are summarized as follows:

- We design an enhanced heartbeat mechanism to periodically examine whether a private server is still alive or not. Different from existing heartbeat mechanisms, we employ cryptographic method to strengthen our scheme for the semi-trusted cloud environment, and make sure that the cold boot attack on a private server is detected.
- We propose an authorized device management mechanism that allows remote owners to monitor/control the I/O device access. We design a scheme that can detect any unauthorized I/O device access to a private server.

The rest of this paper is organized as follows. Section II discusses the related work. Section III introduces the system and threat models. Section IV and V present the enhanced heartbeat mechanism and the authorized device management mechanism, respectively. Section VI presents the evaluations of our proposed schemes, followed by the conclusion in Section VII.

II. Related Work

Many efforts have been made to secure users' sensitive data and privacy in public cloud, such as encrypting outsourced data or applying fully homomorphic encryption schemes [10]. Though cryptographic techniques are able to provide enough protection, they also cause the overhead of decryption and some inconveniences, such as data query [11] and access control [12]. Furthermore, it seems that fully homomorphic encryption schemes have a long way to go before they can be used in practice.

Recently, hybrid cloud has been proposed to balance cloud's benefits and user security concerns. Wood et al. [13] propose a VPN-like network to provide secure and seamless resource integration between private and public clouds. Zhang et al. [14] present a user-transparent hybrid-cloud based secure data-intensive computing framework - *Sedic*, which ensures that sensitive data are not exposed to public cloud while making use of computing resources in public cloud. *Sedic* leverages the characteristics of MapReduce to automatically partition a computing job according to the security levels of the data it works on, and arranges the computation across a hybrid cloud. Study [15] proposes an execution model - *HybrEx* to support confidentiality and privacy in cloud computing. By partitioning data and computation, *HybrEx* allows an organization to utilize their own infrastructure for sensitive, private data and computation, while integrating public clouds for non-sensitive, public data and computation.

III. System and Threat Model

In the CIC model, a CSP allows cloud customers to rent physical infrastructure (such as rack space and servers) to build a small private cloud within the public cloud. The CIC model is illustrated in Fig. 1, where the private cloud locates inside the public cloud and is monitored by the outside owner. Data links are used to exchange data between public and private

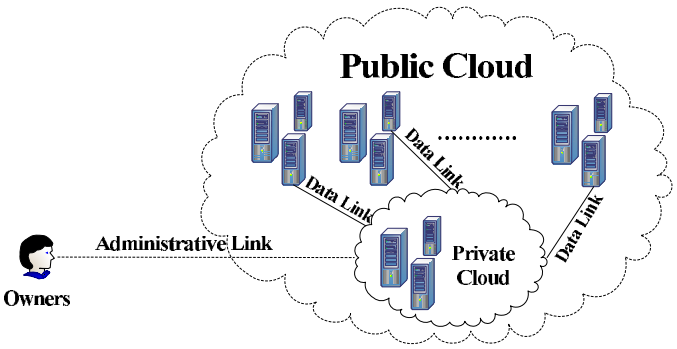

Fig. 1. CIC Model

clouds. Customers can store highly sensitive data in the private cloud. The CIC model has two advantages: 1) compared with public cloud models (IaaS, PaaS and SaaS), customers have total control of the private cloud and they don't need to worry about the security and privacy of data in the private cloud; and 2) compared with the standard hybrid cloud model, private cloud and public cloud are very close to each other, which significantly reduces network delay and communication cost.

However, the CIC model also causes other security concerns. The traffic between the private cloud and the outside owner could be captured and analyzed. Since the I/O devices of the private cloud servers can be physically accessed, an attacker is able to perform cold boot attack [8], which retrieves memory contents that remain readable from seconds to minutes after power is shutdown. Hard disk can be removed from a server and copied. In addition, an attacker may launch a USB autorun attack [9]. There are some trivial ways to protect the private cloud, such as using locked-up cabinet or specialized hardware. However, locked-up cabinet increases the cost and may waste the rest of space in the rack. In addition, the attacker may be able to get a copy of the key. Specialized hardware also increases the cost.

Therefore, effective security schemes need to be designed for the CIC model, which can defense or detect the attacks on private servers. In this paper, we study two physical level attacks:

- **Disassembly Attack**. A disassembly attack is a brutal way to recover or copy users' data in temporary and persistent storage. An attacker can disassemble a computer into parts, such as removing memory (or hard disk) from the computer, and then he can recover the memory content or make a disk-to-disk copy to obtain sensitive data. Since the private servers locate in the public cloud, an attacker can easily access the servers and disassemble components without owners' awareness, and the data will be exposed even if the whole disk has been encrypted [8].
- **Attached Attack**. An attached attack exploits potential weaknesses of I/O devices. Some Linux desktop applications provide friendly user interfaces, such as auto-mounting USB devices, which may give attackers an opportunity to launch virus in a USB disk [9].

IV. The Enhanced Heartbeat Mechanism

The key to fight against the disassembly attack is to prove that a private server works continuously and normally as expected since the disassembly attack disrupts normal operations of the private server. Heartbeat mechanism is a common method used to guarantee the high availability of key services in distributed computing, which allows clients to know about the presence (or disappearance) of peer processes on other machines and to easily exchange messages with them [16]. However, existing heartbeat mechanisms using plain text are not suitable for our case because the public cloud can eavesdrop on the traffic and launch an impersonate attack (e.g., sending fake heartbeat messages). In our research, we improve the heartbeat mechanism to have the following properties:

1) **Anti-fake**. Because the private cloud runs in a semi-trusted environment, every message may be captured and analyzed. If an attacker is able to infer the content and pattern of the heartbeat communication, it can easily launch an impersonate attack in which the attacker's machine can pretend to be the private server and send fake heartbeat messages to the outside owner. Therefore, the heartbeat mechanism must be anti-fake.
2) **Low-overhead**. Since a heartbeat mechanism continuously sends heartbeat messages, the traffic volume may be large. The communication overhead should be minimized.

A. The Basic Mechanism

In order to provide anti-fake property, we use one-way hash function to generate a sequence of random values. One-way key chain is a widely-used cryptographic primitive [17]. At the beginning, the last element of the chain k_t is randomly selected, and then the rest elements of the chain are generated by repeatedly applying a one-way function H. The first element k_0 is a commitment to the entire one-way chain, which can be used to verify any element of the chain. For example, one can verify the ith element k_i by checking if $H^i(k_i) = k_0$. More generally, if $i < j$, one can verify that k_j is part of the chain by checking if $H^{(j-i)}(k_j) = k_i$, and we say that k_i commits to k_j. The elements of the chain are revealed in reverse order k_0, k_1, $\cdots$, k_{t-1}, k_t. Fig. 2 illustrates the generation/reveal process of a one-way key chain.

The enhanced heartbeat mechanism employs the one-way key chain to confirm the onlineness of private cloud servers. A server sends one key of the chain in the reveal order periodically, and this heartbeat communication is used to detect possible disassembly attack. If an attacker launches the disassembly attack and interrupts the running of the private server, the keys cannot be sent at the normal frequency and value. If the remote owner's server does not receive the expected heartbeat message, it knows that the private server is attacked. Due to the property of one-way key chain, an attacker cannot infer the future keys from the known ones, it is impossible for an attacker to fake valid future keys and perform an impersonate attack.

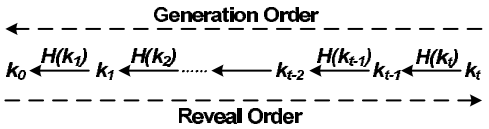

Fig. 2. One-way key chain

In the following subsections, we will discuss other aspects of the enhanced heartbeat mechanism: new key generation, interval and overhead.

B. New Key Generation

In our mechanism, SHA-1 hash function is used to generate the one-way key chain. However, the heartbeat mechanism sends lots of messages (and hence keys) during the operation of the private cloud. When the current key chain is exhausted, new keys need to be generated. In our mechanism, a pre-computed one-way key chain is stored in the private server. When these keys are going to run out, the remote owner will upload a new pre-computed key chain to the private server.

C. Interval and Overhead

It is important to use a proper time interval for the heartbeat messages. If the interval is too short, it may result in too many messages and large overhead between the private cloud and the remote owner. On the other hand, if the interval is too long, the heartbeat mechanism may not be able to detect unusual shutdown of the private server. We analyze the disassembly attack that tries to steal sensitive data in hard disk, and we find there are four main steps involving in a successful attack: 1) disassembling the hard disk from the private server, 2) copying data, 3)installing the hard disk back to private server, and 4) rebooting the machine. The total time is given in Equation (1):

$$\begin{aligned} Total\ Time &= disassembling\ time \\ &+ data\ copy\ time \\ &+ installation\ time \\ &+ reboot\ time \end{aligned} \quad (1)$$

Usually, the disassembling time and the hard disk installation time is small. The copy time depends on the volume of the data: i.e., larger data volume needs more copy time. If the volume of sensitive data is small, then the copy time is also small. In general, the reboot time is the dominating factor and it determines how long the interval of heartbeat messages should be.

The reboot time is the duration of the following events: a computer is at normal working state, shutting down, and then back to normal working state again. However, it is not easy to determine whether a computer has finished its boot process because there are many components, processes or services of operating system to be initialized. In order to estimate the reboot time, a basic network diagnosis command "ping" is used to detect whether a target machine has finished rebooting or not. Note that a quick response from "ping" command indicates that the TCP/IP stack of the operating system is

working normally. When a user types the "reboot" command, the operating system immediately begins its closure routine. In our test, after 1-2 seconds of typing the "reboot" command, the server does not response the "ping" command anymore.

In our experiment, a powerful server with 32GB memory and Intel Xeon E5620 2.4GHz CPU is used. There are about 331 processes in the operating system. Another PC is used to send the ICMP messages continuously generated by the "ping" command. The experimental results show that the reboot time of the server is stable, about 171 seconds. The interval of heartbeat messages should be smaller than the reboot time of the server, so we set the interval to 150 seconds. Since the length of a key is only 20 bytes, the overhead of heartbeat messages is about 20 bytes per 150 seconds, which is very small.

V. The Authorized Device Management Mechanism

In this section we study another attack - the attached attack, which can be exploited to hack into the private server and monitor the activities via unauthorized hardware connections. For example, if the attacker inserts a CD or connects a USB drive containing virus and the autorun mechanism is permitted, the virus will infect the system. Different from the disassembly attack, the attached attack does not interrupt the running of the private server, and hence the enhanced heartbeat mechanism cannot detect it. Furthermore, it is hard to prevent the attacker from attaching I/O devices to the server because the private server physically locates in the public cloud.

To defend against the attached attack, we propose an authorized device management mechanism that can monitor and manage I/O devices in private cloud. The basic idea is: no matter what kinds of I/O devices are attached, they must be accessed using system calls provided by the operating system, such as the *open* operation that is the only way to load (malicious) programs into the server. If an I/O device tries to connect to the private server, the *open* operation will be called. Before executing the *open* operation, the private server (and maybe also the remote owner) will verify whether such I/O connection is authorized or not. By doing this, we can prevent any unauthorized I/O connection (and hence the attached attack) to the private server.

To avoid any circumvention of the attacker on the above scheme, the security mechanism must be intrinsic in the operating system, which makes it difficult to compromise. System calls are the set of native functions for a program to request a service from the kernel, which includes hardware related services, creating and executing new processes, and communicating with integral kernel services. System calls provide an essential interface between a process and the operating system. In Unix, Unix-like and other POSIX-compatible operating systems, popular filesystem calls are *open*, *read* and *write*, which correspond to the kernel functions *sys_open*, *sys_read* and *sys_write* etc. When a file is accessed, the operating system initially makes a call to:

$int\ open(const\ char\ *pathname,$
$int\ flags,$
$mode\_t\ mode);$

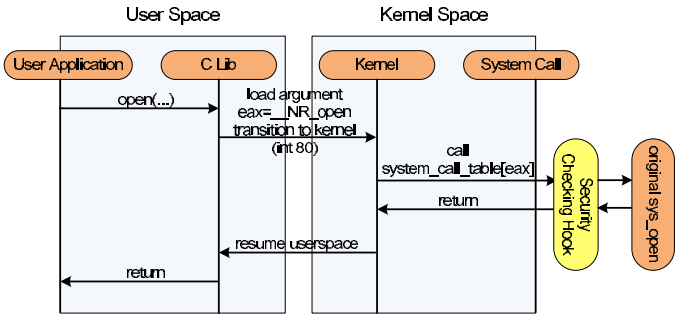

Fig. 3. The hook of system call *sys_open*

This in turn calls:

$asmlinkage\ long\ sys\_open($
$const\ char\_\_user\ *filename,$
$int\ flags,$
$int\ mode);$

which is responsible for the actual operation of *open*.

Since every device has its own directory in the filesystem, the file path will indicate whether the device is accessed or not. However, original system calls do not have this functionality to satisfy the above security requirements. Therefore, we encapsulate the original system calls that file operations depend on to examine the file/directory path pattern before these calls are actually invoked. The process is illustrated in Fig. 3. We install a security checking hook function to the original *sys_open* of the operating system, which has the following steps:

1) when an *open* operation is invoked, it abstracts file path from the pass-in parameters.
2) it examines the prefix of the file path to check whether there are unexpected requests generated by unauthorized I/O devices.
3) it directly omits these file operations or asks for authorization according to the predefined security policy.

VI. Experimental Results

We implement a prototype of our security mechanisms on a virtual Ubuntu machine, which has a Intel(R) Core(TM)2 Duo T5850 @ 2.16GHz CPU and 1GB RAM. Our prototype consists of two main components: 1) the enhanced heartbeat mechanism, and 2) the authorized device management mechanism. The modules are presented in Section IV and V, respectively.

We run a set of experiments to evaluate the overhead of the security mechanisms on the target platform. The overhead mainly comes from two aspects: the traffic of heartbeat messages and the execution of security checking on file operations. Section V has discussed the overhead of heartbeat messages. Thus, in the following we focus on the overhead of the security checking on file operations. In general, security checking on *open* should not be a frequent operation compared with *read* and *write* in the whole file process, since it is a one-time operation at the beginning. On the other hand, *open*, as a

common kernel system call, is used by all system services and applications.

Our experiment is to understand the performance overhead on the operating system when the security checking operation is running. We leverage several well-known tests in the suite of the UnixBench benchmark [18] to evaluate the overhead. We compare the overhead of running the security checking on the Ubuntu system with the overhead of the normal system without the security checking operation.

The results are plotted in Fig 4. Most of the overheads are no more than 5%, except that the overhead of the "Pipe Throughput" test is 11%. Fig 4 shows that the overhead of the security checking operation is small and acceptable. Additionally, the results also show that the overheads of different tests are different, which is mostly due to the difference in the number of *open* system call in each test.

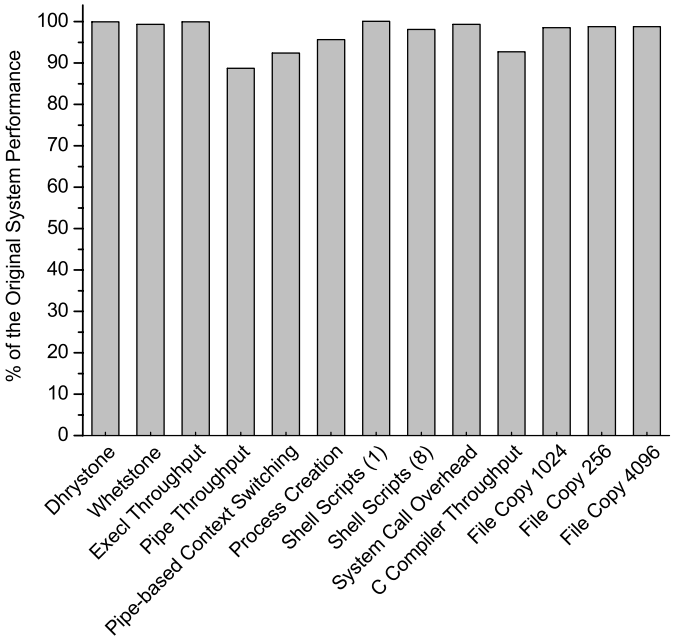

Fig. 4. Performance Impact on Original System

VII. Conclusion

In this paper we proposed a different hybrid cloud model - Cloud-in-Cloud (CIC), which places a small private cloud physically within a public cloud. The motivation is to significantly reduce the communication cost between private and public clouds in the traditional hybrid clouds. The CIC model introduces new security issues because an attacker may have physical access to the private servers. In this paper, we presented two effective security mechanisms to defend against the attacks. The enhanced heartbeat mechanism can continuously examine the onlineness of private servers and defend against the cold boot attack. The authorized device management mechanism is able to detect and prevent any unauthorized I/O access, such as the USB autorun attack. Our analysis and experiments showed that the overhead of our security mechanisms is small.

Acknowledgment

This research was supported in part by the China National Basic Research Program (973 Program) under grants 2011CB302605 and 2007CB311101, the China National High Technology Research and Development Program (863 Program) under grant 2010AA012504 and 2011AA010705; and by the US National Science Foundation under grants CNS-0963578, CNS-1002974, CNS-1022552, and CNS-1065444.

References

[1] "Amazon EC2." [Online]. Available: http://aws.amazon.com/ec2/
[2] "Google App Engine." [Online]. Available: http://www.google.com/enterprise/appengine/
[3] "IBM Cloud." [Online]. Available: http://www.ibm.com/cloud-computing/us/en/
[4] "Microsoft Azure." [Online]. Available: http://www.microsoft.com/windowsazure/
[5] "Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region," April 2011. [Online]. Available: http://aws.amazon.com/message/65648/
[6] "Amazon EC2 Service Level Agreement," 2012. [Online]. Available: http://aws.amazon.com/ec2-sla/
[7] NIST, "The NIST Definition of Cloud Computing," Special Publication 800-145, July 2011.
[8] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, May 2009.
[9] J. Larimer, "USB Autorun attacks against linux," ShmooCon 201, 2011. [Online]. Available: http://blogs.iss.net/archive/papers/ShmooCon2011-USB_Autorun_attacks_against_Linux.pdf
[10] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2009, pp. 169–178.
[11] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 393–402.
[12] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proceedings of the 29th conference on Information communications*. Piscataway, NJ, USA: IEEE Press, 2010, pp. 534–542.
[13] T. Wood, A. Gerber, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "The case for enterprise-ready virtual private clouds," in *Proceedings of the 2009 conference on Hot topics in cloud computing*. Berkeley, CA, USA: USENIX Association, 2009.
[14] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan, "Sedic: privacy-aware data intensive computing on hybrid clouds," in *Proceedings of the 18th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2011, pp. 515–526.
[15] S. Y. Ko, K. Jeon, and R. Morales, "The hybrex model for confidentiality and privacy in cloud computing," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 8–8.
[16] "Linux-HA." [Online]. Available: http://linux-ha.org/wiki/Main_Page
[17] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981.
[18] "Unixbench." [Online]. Available: http://www.tux.org/pub/tux/benchmarks/System/unixbench/