IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 1, March, 2013
ISSN: 2320 - 8791
www.ijreat.org

Decentralized Information in Cloud Computing using Distributed Technique

A.Senthamarai Selvan[1], S.Kishore Verma[2] and S.Suresh[3]

[1,2,3]Assistant Professor, Department of Computer Science and Engineering,
C.Abdul Hakeem College of Engineering and Technology, Melvisharam, Vellore.

Abstract
A cloud computing is highly scalable services over the internet. The cloud services are that users' data are usually processed remotely in unknown machines that users do not own or operate. The new emerging technology, users' fears of losing control of their own data (particularly, financial and health data) can become a significant barrier to the wide adoption of cloud services. In this paper, we propose a novel approach of cloud information accountability (CIA) framework in distributed way to keep track of the actual usage of the users' data in the cloud. The CIA framework provides end-to-end accountability in a highly distributed fashion. One of the main innovative features of the CIA framework lies in its ability of maintaining lightweight and powerful accountability that combines aspects of access control, usage control and authentication. We influence the JAR programmable capabilities to both create a dynamic and traveling object, and to ensure that any access to users' data will trigger authentication and automated logging local to the JARs. To strengthen user's control, we also provide distributed auditing mechanisms.

Key words: *Cloud computing, accountability, distributed*.

1. Introduction

Cloud computing presents a new way to supplement the current consumption and delivery model for IT services based on the Internet, by providing for dynamically scalable and often virtualized resources as a service over the Internet. While enjoying the convenience brought by this new technology, users also start worrying about losing control of their own data. The data processed on clouds are often outsourced, leading to a number of issues related to accountability, including the handling of personally identifiable information conventional access control approaches developed for closed domains such as databases and operating systems, or approaches using a centralized server in distributed environments, are not suitable, due to the following features characterizing cloud environments. Unlike privacy protection technologies which are built on the hide-it-or-lose-it perspective, information accountability focuses on keeping the data usage transparent and track able.

Our proposed CIA framework provides end-to-end accountability in a highly distributed fashion. The design of the CIA framework presents substantial challenges, including uniquely identifying CSPs, ensuring the reliability of the log, adapting to a highly decentralized infrastructure, etc. Our basic approach toward addressing these issues is to leverage and extend the programmable capability of JAR (Java ARchives) files to automatically log the usage of the users' data by any entity in the cloud. Our experiments demonstrate the efficiency, scalability and granularity of our approach. In addition, we also provide a detailed security analysis and discuss the reliability and strength of our architecture in the face of various nontrivial attacks, launched by malicious users or due to compromised Java Running Environment. We have made the following new contributions. First, we integrated integrity checks and oblivious hashing (OH) technique to our system in order to strengthen the dependability of our system in case of compromised JRE. We also updated the log records structure to provide additional guarantees of integrity and authenticity. Second, we extended the security analysis to cover more possible attack scenarios.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 lays out our proposed CIA framework and Sections 4 describe the detailed algorithms for automated logging mechanism and auditing approaches, respectively. Section 5 presents a security analysis of our framework, followed by an experimental study in Section 6. Finally, Section 7 concludes the paper.

2 Related Works

The review related works addressing the privacy and security issues in the cloud. Cloud computing has raised a range of important privacy and security issues [2], [3], [4]. Such issues are due to the fact that, in the cloud, users' data and applications reside at least for a certain amount of time on the cloud cluster which is owned and maintained by a third party. Concerns arise since in the cloud it is not

always clear to individuals why their personal information is requested or how it will be used or passed on to other parties. Their basic idea is that the user's private data are sent to the cloud in an encrypted form, and the processing is done on the encrypted data. The output of the processing is de obfuscated by the privacy manager to reveal the correct result. However, the privacy manager provides only limited features in that it does not guarantee protection once the data are being disclosed. To the best of our knowledge, the only work proposing a distributed approach to accountability is from Lee and colleagues [6]. The authors have proposed an agent-based system specific to grid computing. Distributed jobs, along with the resource consumption at local machines are tracked by static software agents.

3. Cloud Information Accountability

We present an overview of the Cloud Information Accountability framework and discuss how the CIA framework meets the design requirements discussed in the previous section. The Cloud Information Accountability framework proposed in this work conducts automated logging and distributed auditing of relevant access performed by any entity, carried out at any point of time at any cloud service provider. It has two major components: logger and log harmonizer.

3.1 Major Components

There are two major components of the CIA, the first being the logger, and the second being the log harmonizer. The logger is the component which is strongly coupled with the user's data, so that it is downloaded when the data are accessed, and is copied whenever the data are copied. The logger is strongly coupled with user's data (either single or multiple data items). Its main tasks include automatically logging access to data items that it contains, encrypting the log record using the public key of the content owner, and periodically sending them to the log harmonizer. The error correction information combined with the encryption and authentication mechanism provides a robust and reliable recovery mechanism, therefore meeting the third requirement the log harmonizer is also responsible for handling log file corruption.

3.2 Data Flow

The overall CIA framework, combining data, users, logger and harmonizer is sketched in Fig. 1. At the beginning, each user creates a pair of public and private keys based on Identity-Based Encryption [7] (step 1 in Fig. 1). The JAR file includes a set of simple access control rules specifying whether and how the cloud servers, and possibly other data stakeholders (users, companies) are authorized to access the content itself. Then, he sends the JAR file to the cloud service provider that he subscribes to. To authenticate the CSP to the JAR (steps 3-5 in Fig. 1), we use OpenSSLbased certificates, wherein a trusted certificate authority certifies the CSP. As for the logging, each time there is an access to the data, the JAR will automatically generate a log record, encrypt it using the public key distributed by the data owner, and store it along with the data (step 6 in Fig. 1). In addition, some error correction information will be sent to the log harmonizer to handle possible log file corruption (step 7 in Fig. 1). To ensure trustworthiness of the logs, each record is signed by the entity accessing the content.

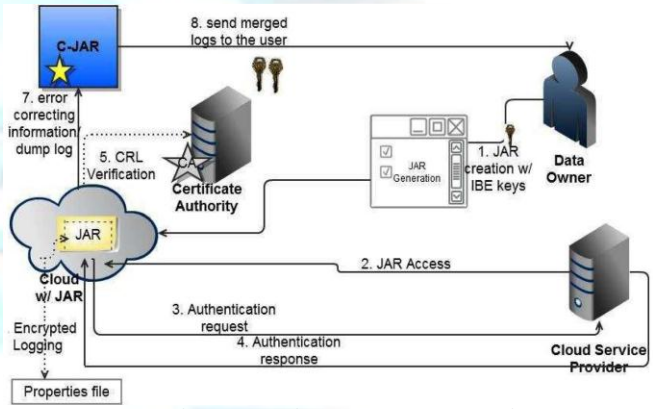

Fig1: Overview of the cloud information accountability framework

4 Automated Logging Mechanisms

First elaborate on the automated logging mechanism and then present techniques to guarantee dependability.

4.1 The Logger Structure

We leverage the programmable capability of JARs to conduct automated logging. A logger component is a nested Java JAR file which stores a user's data items and corresponding log files. The main responsibility of the outer JAR is to handle authentication of entities which want to access the data stored in the JAR file. A Java policy specifies which permissions are available for a particular piece of code in a Java application environment. The outer JAR is also in charge of selecting the correct inner JAR according to the identity of the entity who requests the data. Each inner JAR contains the encrypted data, class files to facilitate retrieval of log files and display enclosed data in a suitable format, and a log file for each encrypted item. We support two options:

1. **PureLog.** Its main task is to record every access to the data. The log files are used for pure auditing purpose.
2. **AccessLog.** It has two functions: logging actions and enforcing access control. In case an access request is denied, the JAR will record the time when the request is made. If the access request is granted, the JAR will additionally record the access information along with the duration for which the access is allowed.

4.2 Log Record Generation

Log records are generated by the logger component. Logging occurs at any access to the data in the JAR, and new log entries are appended sequentially, in order of creation LR= <r1; . . . ; ri. Each record ri is encrypted individually and appended to the log file. The checksum is computed using a collision-free hash function [5]. The component sig denotes the signature of the record created by the server. The most critical part is to log the actions on the users' data. In the current system, we support four types of actions, i.e., Act has one of the following four values: view, download, timed access, and Location-based access. We propose a specific method to correctly record or enforce it depending on the type of the logging module, which are elaborated as follows:

1. View: The entity (e.g., the cloud service provider) can only read the data but is not allowed to save a raw copy of it anywhere permanently. For this type of action, the PureLog will simply write a log record about the access, while the AccessLogs will enforce the action through the enclosed access control module. Recall that the data are encrypted and stored in the inner JAR.
2. Download: The entity is allowed to save a raw copy of the data and the entity will have no control over this copy neither log records regarding access to the copy. If PureLog is adopted, the user's data will be directly downloadable in a pure form using a link. When an entity clicks this download link, the JAR file associated with the data will decrypt the data and give it to the entity in raw form.
3. Timed_access. This action is combined with the view-only access, and it indicates that the data are made available only for a certain period of time. The Purelog will just record the access starting time and its duration, while the AccessLog will enforce that the access is allowed only within the specified period of time. The duration for which the access is allowed is calculated using the Network Time Protocol.
4. Location-based_access. In this case, the PureLog will record the location of the entities. The AccessLog will verify the location for each of such access. The access is granted and the data are made available only to entities located at locations specified by the data owner.

4.3 Dependability of Logs

We ensure the dependability of logs. In particular, we aim to prevent the following two types of attacks. First, an attacker may try to evade the auditing mechanism by storing the JARs remotely, corrupting the JAR, or trying to prevent them from communicating with the user. Second, the attacker may try to compromise the JRE used to run the JAR files.

4.3.1 JARs Availability

To protect against attacks perpetrated on offline JARs, the CIA includes a log harmonizer which has two main responsibilities: to deal with copies of JARs and to recover corrupted logs. Each log harmonizer is in charge of copies of logger components containing the same set of data items. The harmonizer is implemented as a JAR file. The log harmonizer is located at a known IP address. Typically, the harmonizer resides at the user's end as part of his local machine, or alternatively, it can either be stored in a user's desktop or in a proxy server.

4.3.2 Log Correctness

It is essential that the JRE of the system on which the logger components are running remain unmodified. To verify the integrity of the logger component, we rely on a two-step process:

1. We repair the JRE before the logger is launched and any kind of access is given, so as to provide guarantees of integrity of the JRE.
2. We insert hash codes, which calculate the hash values of the program traces of the modules being executed by the logger component. This helps us detect modifications of the JRE once the logger component has been launched, and are useful to verify if the original code flow of execution is altered.

4.4 Push and Pull Mode

To allow users to be timely and accurately informed about their data usage, our distributed logging mechanism is complemented by an innovative auditing mechanism. We support two complementary auditing modes: 1) push mode; 2) pull mode. Push mode. In this mode, the logs are periodically pushed to the data owner (or auditor) by the harmonizer. Pull mode. This mode allows auditors to retrieve the logs anytime when they want to check the recent access to their own data.

4.5 Algorithms

Pushing or pulling strategies have interesting tradeoffs. The pushing strategy is beneficial when there are a large number of accesses to the data within a short period of time. In this case, if the data are not pushed out frequently enough, the log file may become very large, which may increase cost of operations like copying data. The pushing mode may be preferred by data owners who are organizations and need to keep track of the data usage consistently over time.

5. Security Discussion

Our analysis is based on a semi honest adversary model by assuming that a user does not release his master keys to unauthorized parties, while the attacker may try to learn extra information from the log files. We assume that attackers may have sufficient Java programming skills to disassemble a JAR file and prior knowledge of our CIA architecture. We first assume that the JVM is not corrupted, followed by a discussion on how to ensure that this assumption holds true.

Require: $size$: maximum size of the log file specified by the data owner, $time$: maximum time allowed to elapse before the log file is dumped, $tbeg$: timestamp at which the last dump occurred, log: current log file, $pull$: indicates whether a command from the data owner is received.

1: Let TS(NTP) be the network time protocol timestamp
2: $pull = 0$
3: $rec := \langle UID, OID, AccessType, Result, Time, Loc \rangle$
4: $curtime := TS(NTP)$
5: $lsize := sizeof(log)$ //current size of the log
6: **if** $((cutime - tbeg) < time) \&\& (lsize < size) \&\& (pull == 0)$ **then**
7: $log := log + ENCRYPT(rec)$ // ENCRYPT is the encryption function used to encrypt the record
8: $PING$ to CJAR //send a PING to the harmonizer to check if it is alive
9: **if** $PING$-CJAR **then**
10: PUSH $RS(rec)$ // write the error correcting bits
11: **else**
12: EXIT(1) // error if no PING is received
13: **end if**
14: **end if**
15: **if** $((cutime - tbeg) > time) || (lsize >= size) || (pull \neq 0)$ **then**
16: // Check if PING is received
17: **if** $PING$-CJAR **then**
18: PUSH log //write the log file to the harmonizer
19: $RS(log) := NULL$ // reset the error correction records
20: $tbeg := TS(NTP)$ // reset the $tbeg$ variable
21: $pull := 0$
22: **else**
23: EXIT(1) // error if no PING is received
24: **end if**
25: **end if**

Fig:2. Push and pull PureLog mode.

5.1 Copying Attack

The most intuitive attack is that the attacker copies entire JAR files. The attacker may assume that doing so allows accessing the data in the JAR file without being noticed by the data owner. That is, even if the data owner is not aware of the existence of the additional copies of its JAR files, he will still be able to receive log files from all existing copies. If attackers move copies of JARs to places where the harmonizer cannot connect, the copies of JARs will soon become inaccessible. This is because each JAR is required to write redundancy information to the harmonizer periodically.

5.2 Disassembling Attack

Another possible attack is to disassemble the JAR file of the logger and then attempt to extract useful information out of it or spoil the log records in it. Given the ease of disassembling JAR files, this attack poses one of the most serious threats to our architecture. Since we cannot prevent an attacker to gain possession of the JARs, we rely on the strength of the cryptographic schemes applied to preserve the integrity and confidentiality of the logs.

5.3 Men-in-the-Middle Attack

An attacker may intercept messages during the authentication of a service provider with the certificate authority, and reply the messages in order to masquerade as a legitimate service provider. There are two points in time that the attacker can replay the messages. One is after the actual service provider has completely disconnected and ended a session with the certificate authority. The other is when the actual service provider is disconnected but the session is not over, so the attacker may try to renegotiate the connection.

5.4 Compromised JVM Attack

An attacker may try to compromise the JVM. To quickly detect and correct these issues, OH adds hash code to capture the computation results of each instruction and computes the oblivious-hash value as the computation proceeds. These two techniques allow for a first quick detection of errors due to malicious JVM, therefore mitigating the risk of running subverted JARs.

6. Performance Study

The settings of the test environment and then present the performance study of our system.

6.1 Experimental Settings

We tested our CIA framework by setting up a small cloud, using the Emulab tested. In particular, the test environment consists of several OpenSSL-enabled servers: one head node which is the certificate authority, and several computing nodes. Each of the servers is installed with Eucalyptus. Eucalyptus is an open source cloud implementation for Linux-based systems. It is loosely based on Amazon EC2, therefore bringing the powerful functionalities of Amazon EC2 into the open source domain.

6.2 Experimental Results

In the experiments examine the time taken to create a log file and then measure the overhead in the system. With respect to time, the overhead can occur at three points: during the authentication, during encryption of a log record, and during the merging of the logs. Also, with respect to storage overhead, we notice that our architecture is very lightweight, in that the only data to be stored are given by the actual files and the associated logs.

6.2.1 Log Creation Time

In the first round of experiments, we are interested in finding out the time taken to create a log file when there are entities continuously accessing the data, causing continuous logging. Results are shown in Fig. It is not surprising to see that the time to create a log file increases linearly with the size of the log file. Specifically, the time to create a 100 Kb file is about 114.5 ms while the time to create a 1 MB file averages at 731 ms. With this experiment as the baseline, one can decide the amount of time to be specified between dumps, keeping other variables like space constraints or network traffic in mind.

6.2.2 Authentication Time

The next point that the overhead can occur is during the authentication of a CSP. If the time taken for this authentication is too long, it may become a bottleneck for accessing the enclosed data. To evaluate this, the head node issued OpenSSL certificates for the computing nodes and we measured the total time for the OpenSSL authentication to be completed and the certificate revocation to be checked.

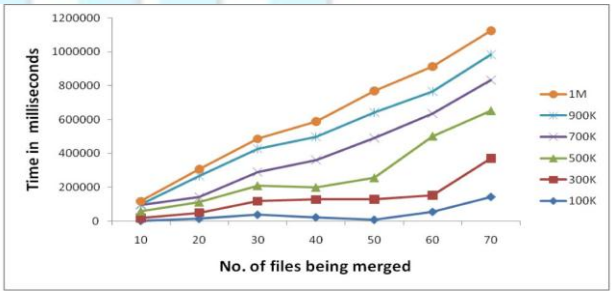

Fig: 3. Time to merge log files.

6.2.3 Time Taken to Perform Logging

This set of experiments studies the effect of log file size on the logging performance. We measure the average time taken to grant an access plus the time to write the corresponding log record. The time for granting any access to the data items in a JAR file includes the time to evaluate and enforce the applicable policies and to locate the requested data items. In the experiment, we let multiple servers continuously access the same data JAR file for a minute and recorded the number of log records generated. Each access is just a view request and hence the time for executing the action is negligible. As a result, the average time to log an action is about 10 seconds, which includes the time taken by a user to double click the JAR or by a server to run the script to open the JAR.

6.2.4 Log Merging Time

To check if the log harmonizer can be a bottleneck, we measure the amount of time required to merge log files. In this experiment, we ensured that each of the log files had 10 to 25 percent of the records in common with one other. The exact number of records in common was random for each repetition of the experiment. The time was averaged over 10 repetitions.

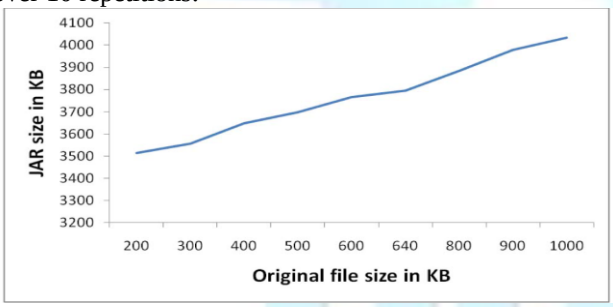

Fig. 4. Size of the logger component.

6.2.5 Size of the Data JAR Files

Finally, we investigate whether a single logger, used to handle more than one file, results in storage overhead. We measure the size of the loggers (JARs) by varying the number and size of data items held by them. We tested the increase in size of the logger containing 10 content files of the same size as the file size increases.

6.2.6 Overhead Added by JVM Integrity Checking

A investigate overhead added by both the JRE installation/repair process, and by the time taken for computation of hash codes. The time taken for JRE installation/repair averages around 6,500 ms. This time was measured by taking the system time stamp at the beginning and end of the installation/repair. To calculate the time overhead added by the hash codes, The number of hash commands varies based on the size of the code in the code does not change with the content, the number of hash commands remain constant.

7. Conclusion

We proposed innovative approaches for automatically logging any access to the data in the cloud together with an auditing mechanism. Our approach allows the data owner to not only audit his content but also enforce strong back-end protection if needed. One of the main features of our work is that it enables the data owner to audit even those copies of its data that were made without his knowledge.

References

[1]. Smitha Sundareswaran, Anna C. Squicciarini," Ensuring Distributed Accountability for Data Sharing in the Cloud" IEEE Transactions on dependable and secure computing, 2012

[2]. P.T. Jaeger, J. Lin, and J.M. Grimes, "Cloud Computing and Information Policy: Computing in a Policy Cloud," J. Information Technology and Politics, 2009.

[3]. T. Mather, S. Kumaraswamy, and S. Latif, Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance (Theory in Practice), first ed. O' Reilly, 2009.

[4]. S. Pearson and A. Charlesworth, "Accountability as a Way Forward for Privacy Protection in the Cloud," Proc. First Int'l Conf. Cloud Computing, 2009.

[5]. B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley & Sons, 1993.

[6]. W. Lee, A. Cinzia Squicciarini, and E. Bertino, "The Design and Evaluation of Accountable Grid Computing System," Proc. 29th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '09), 2009.

[7]. D. Boneh and M.K. Franklin, "Identity-Based Encryption from the Weil Pairing," Proc. Int'l Cryptology Conf. Advances in Cryptology, 2001.

[8]. R. Corin, S. Etalle, J.I. den Hartog, G. Lenzini, and I. Staicu, "A Logic for Auditing Accountability in Decentralized Systems," Proc. IFIP TC1 WG1.7 Workshop Formal Aspects in Security and Trust, 2005.

[9] B. Crispo and G. Ruffo, "Reasoning about Accountability within Delegation," Proc. Third Int'l Conf. Information and Comm. Security (ICICS), 2001.

[10] Y. Chen et al., "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive," Proc. Int'l Workshop Information Hiding, F. Petitcolas, ed., 2003.

[11] S. Etalle and W.H. Winsborough, "A Posteriori Compliance Control," SACMAT '07: Proc. 12th ACM Symp. Access Control Models and Technologies, 2007.