IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, **p**-ISSN: 2278-8719
Vol. 3, Issue 5 (May. 2013), ||V1 || PP 18-24

Data Sharing and Distributed Accountability in Cloud

Mrs. Suyoga Bansode[1], Prof. Pravin P. Kalyankar[2]
[1](Department of Computer Science & Engineering)
[2](Associate Professor in Computer Science & Engineering Department)

Abstract: In most commercial and legal transactions, the ability to hold individuals or organizations accountable for transactions is important. Accountability is an important aspect of any computer system. It assures that every action executed in the system can be traced back to some entity. Accountability is even more crucial for assuring the safety and security. A major feature of the cloud services is that users' data are usually processed remotely in unknown machines that users do not own or operate. Cloud computing enables highly scalable services to be easily consumed over the Internet on an as-needed basis, we propose a novel highly decentralized information accountability framework to keep track of the actual usage of the users' data in the cloud. In particular, we propose an object centered approach that enables enclosing our logging mechanism together with users' data and policies. We influence the JAR programmable capabilities to both create a dynamic and travelling object and to ensure that any access to user's data will trigger authentication and automated logging local to the JARs. To strengthen user's control, we also provide distributed auditing mechanism.

Keywords : *Cloud, Cloud Computing, CIA Framework, Cloud Service Provider (CSP), Java Archives (JAR).*

I. INTRODUCTION

Cloud computing is the access to computers and their functionality via the Internet or a local area network. Users of a cloud request this access from a set of web services that manage a pool of computing resources (i.e. machines, network, storage, operating systems, application development environments, application programs). When granted, a fraction of the resources in the pool is dedicated to the requesting user until he or she releases them. It is called "cloud computing" because the user cannot actually see or specify the physical location and organization of the equipment hosting the resources they are ultimately allowed to use. 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. To date, there are a number of notable commercial and individual cloud computing services, including Amazon, Google, Microsoft, Yahoo, and Salesforce. Users may not know the machines which actually process and host their data. 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. Such fears are becoming a significant obstacle to the wide acceptance of cloud services. Accountability [3] is the obligation to act as a responsible steward of the personal information of others, to take responsibility for the protection and appropriate use of that information beyond mere legal requirements, and to be accountable for any misuse of that personal information. Accountability places a legal responsibility on an organization to ensure that the contracted partners to whom it supplies data are compliant. Associated with the accountability feature, we also develop two distinct modes for auditing: push mode and pull mode. The push mode refers to logs being periodically sent to the data owner or stakeholder while the pull mode refers to an alternative approach whereby the user (or another authorized party) can retrieve the logs as needed. A Cloud Information Accountability (CIA) framework [2], based on the notion of information accountability which focuses on keeping the data usage transparent and trackable.CIA framework provides end-to end accountability in a highly distributed fashion To influence and expand the programmable capability of JAR (Java Archives) files to automatically log the usage of the users' data by any entity in the cloud. Users will send their data along with any policies such as access control policies and logging policies that they want to enforce, enclosed in JAR files, to cloud service providers. Any access to the data will trigger an automated and authenticated logging mechanism local to the JARs.

1.1 Literature Review

Here we first review related works addressing the privacy and security issues in the cloud.

1.1.1 History

Cloud computing has raised a range of important privacy and security issues. The 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 work of A. Squicciarini, S. Sundareswaran, and D. Lin [6] a Java-based approach is provided to prevent privacy leakage from indexing, which could be integrated with the CIA framework proposed in this work since they build on related architectures. We do not rely on IBE to bind the content with the rules. Instead, we use it to provide strong guarantees for the encrypted content and the log files, such as protection against chosen plaintext and cipher text attacks.

1.1.2 Review of Papers

Jagadeesan et al. recently proposed logic for designing accountability-based distributed systems [4]. To the best of our knowledge, the only work proposing a distributed approach to accountability is from Lee and colleagues. 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. The notion of accountability policies by W. Lee, A. Cinzia Squicciarini, and E. Bertino [5] is mainly focused on resource consumption and on tracking of sub jobs processed at multiple computing nodes, rather than access control. Accountability is an important aspect of any computer system. Because every action executed in the system can be traced back to some entity. It also assures the safety and security in grid systems, given the very large number of users active in these sophisticated environments. Their work addresses such insufficiency by developing a comprehensive accountability system driven by policies and supported by accountability agents. In this paper they first discuss the requirements that have driven the design of their accountability system and then present some interesting aspects related to their accountability framework. They describe a fully working implementation of their accountability system, and conduct extensive experimental evaluations. Their experiments, carried out using the Emulab testbed, demonstrate that the implemented system is efficient and it scales for grid systems of large number of resources and users. In particular, S. Sundareswaran, A. Squicciarini, D. Lin, and S. Huang [7], leverage the programmable capability of Java JAR files to enclose logging mechanism together with user's data and policies.

II. CLOUD SCENARIO

We consider a cloud computing scenario as a significant context where to describe Electronic Data Sharing Agreements i.e. e-DSAs [1], related policy definitions and their enforcement. We believe that e-DSAs and their automated enforcement are keys to further enable business interactions and information flows within the Cloud, by providing more assurance and control on data. Fig. 1 shows an example of involved players in the Cloud and related information flows. In this scenario, multiple Cloud Service Providers (CSP) are available in the Internet. A customer uses the services supplied by a specific CSP to access online travelling, printing, office applications, etc. In order to access these services, customers need to register and disclose personal data, inclusive of address, financial details, etc. In order to provide the required functionalities, a Cloud Service Provider might need to interact with other Service Providers and share relevant data to enable the business transaction. For example, a travelling service might need to interact with an external billing service and flight reservation service in order to supply the required service to the customer. In all these interactions, personal and confidential data needs to be gathered; it can potentially be analyzed, processed and exchanged with other parties. A key issue highlighted by this scenario is that both the customer and service providers might quickly lose control on data when this data is exchanged between different parties in chains of interactions. Customers might desire to retain control about: how their data should be used; who can access it, etc. They might want to: dictate the purposes for which data can be disclosed to third parties; impose constraints on the retention time, notifications, etc. Similar comments apply to a service provider disclosing information to third parties.

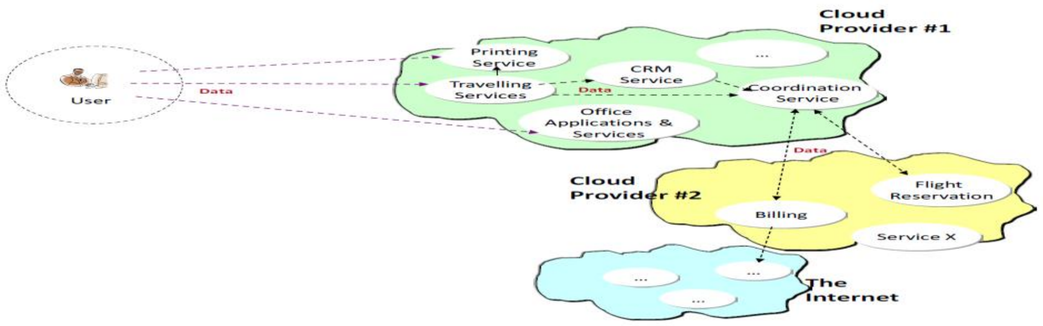

Fig. 1 Cloud Scenario

In other words, the entities that disclose information would like to express preferences including privacy preferences on how their personal and confidential data should be handled along with access control and obligation constraints. Some specific examples of authorization policies for access control and obligation policies follow:

• Authorization Policies for Access Control

(a) Data of my credit card can be accessed by Service Provider 1(SP1) only for Business Transaction purposes;
(b) My email address can be shared with SP2 and SP3 only for business transaction and goods delivery purposes;
(c) My email address details must not be shared with SP4.

• Obligation Policies

(a) I want to be notified by email every time my data is accessed;
(b) I want to be notified every time my credit card is disclosed to another Service Provider;
(c) I want my data to be deleted after 1 year if not accessed/used.

Interestingly, the stated constraints might need to be enforced by all the entities involved in a chain of data disclosures, e.g. in the example, by the Travelling Service, the Printing Service, the Flight Booking Service, etc. Furthermore, the customer might, over time, periodically change their mind and modify some of their preferences and constraints. These changes should be propagated through the chain of disclosures as well.

III. CLOUD INFORMATION ACCOUNTABILITY

Cloud Information Accountability framework conducts computerized 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, logger and the log harmonizer. The logger is the component 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. It handles a particular instance or copy of the user's data and is responsible for logging access to that instance or copy. The log harmonizer forms the central component which allows the user access to the log files. 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 logger requires only minimal support from the server (e.g., a valid Java virtual machine installed) in order to be deployed. The tight coupling between data and logger, results in a highly distributed logging system.

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 (step 1 in Fig.2). Using the generated key, the user will create a logger component which is a JAR file, to store its data items. 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.2), we use OpenSSLbased certificates, wherein a trusted certificate authority certifies the CSP. In the event that the access is requested by a user, we employ SAML-based authentication, wherein a trusted identity provider issues certificates verifying the user's identity based on his username.

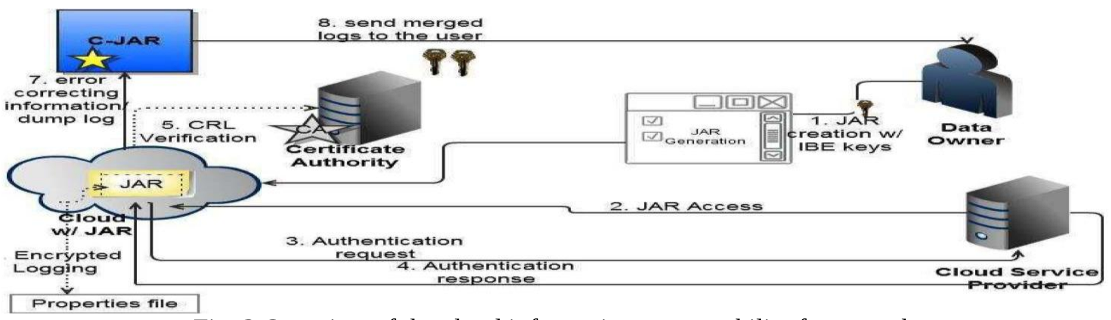

Fig. 2 Overview of the cloud information accountability framework

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.2). Using separate keys can enhance the security without introducing any overhead except in the initialization phase. In addition, some error correction information will be sent to the log harmonizer to handle possible log file corruption (step 7 in Fig.2). To ensure trustworthiness of the logs, each record is signed by the entity accessing the content. The encrypted log files can later be decrypted and their integrity verified. They can be accessed by the data owner or other authorized stakeholders at any time for auditing purposes with the aid of the log harmonizer (step 8 in Fig.2). This proposed framework prevents various attacks such as detecting illegal copies of users' data.

IV. COMPUTERIZED MECHANISM FOR LOGGING

In this we first highly structured on the computerized logging mechanism and then present techniques to guarantee dependability.

4.1The 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. As shown in Fig. 3, our proposed JAR file consists of one outer JAR enclosing one or more inner JARs. 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.

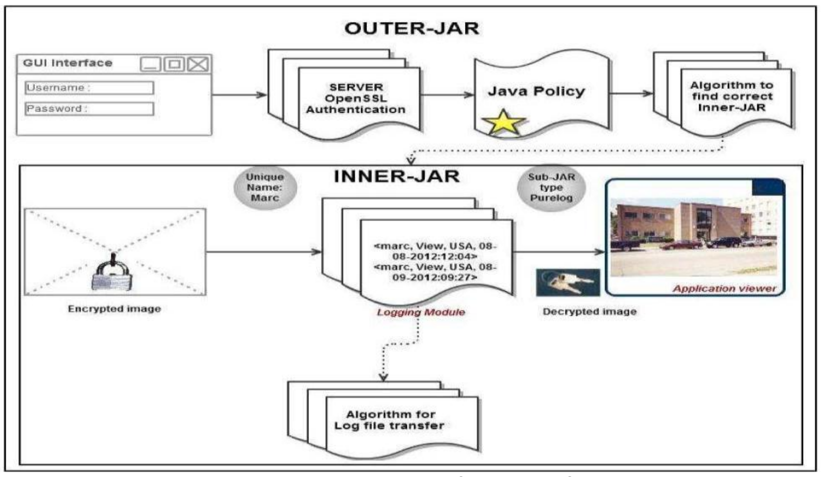
OUTER-JAR
GUI Interface
Username :
Password :
SERVER OpenSSL Authentication
Java Policy
Algorithm to find correct Inner-JAR
INNER-JAR
Unique Name: Marc
Sub-JAR type Purelog
<marc, View, USA, 08-08-2012:12:04>
<marc, View, USA, 08-09-2012:09:27>
Encrypted image
Logging Module
Decrypted image
Application viewer
Algorithm for Log file transfer

Fig. 3 The structure of the JAR file

We support two options: Pure Log - Its main task is to record every access to the data. The log files are used for pure auditing purpose. Access Log - 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= $<r_1; \dots ; r_k>$. Each record r_i is encrypted individually and appended to the log file. In particular, a log record takes the following form:

$$r_i=<ID; Act; T; Loc; h((ID; Act; T; Loc)|r_i - 1| \dots |r_1); sig>$$

Here, r_i indicates that an entity identified by I D has performed an action Act on the user's data at time T at location Loc. The component h((ID; Act; T; Loc)|r_i - 1|. . . |r1) corresponds to the checksum of the records preceding the newly inserted one, concatenated with the main content of the record itself (we use I to denote concatenation). The checksum is computed using a collision-free hash function. The component sig denotes the signature of the record created by the server. If more than one file is handled by the same logger, an additional ObjIDfield is added to each record. To ensure the correctness of the log records, we verify the access time, locations as well as actions. In particular, the time of access is determined using the Network Time Protocol (NTP) to avoid suppression of the correct time by a malicious entity. The location of the cloud service provider can be determined using IP address. 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. For each action, we propose a specific method to correctly record or enforce it depending on the type of the logging module.

V. AUDITING MECHANISM

Here we describe our distributed auditing mechanism including the algorithms for data owners to query the logs regarding their data.

5.1 Push and Pull Mode

We support two complementary auditing modes push mode and pull mode.

5.1.1 Push mode

In this mode, the logs are periodically pushed to the data owner (or auditor) by the harmonizer. The push action will be triggered by either type of the following two events: one is that the time elapses for a certain period according to the temporal timer inserted as part of the JAR file; the other is that the JAR file exceeds the size stipulated by the content owner at the time of creation.

5.1.2 Pull mode

This mode allows auditors to retrieve the logs anytime when they want to check the recent access to their own data. The request will be sent to the harmonizer, and the user will be informed of the data's locations and obtain an integrated copy of the authentic and sealed log file.

5.2 Algorithms

Supporting both pushing and pulling modes helps protecting from some nontrivial attacks.

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 : = <UID,OID,AccessType,Result,Time,loc>
4.  curtime : =TS(NTP)
5.  lsize : = sizeof(log)//current size of the log
6   .if((cutime-tbeg)<time)&&(lsize<size)&&(pull==0) then
7.  l og:=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≠0)  then
16. // Check if PING is received
17. if PING-CJAR then
18.  PUSH log // while 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. 4 Push and pull PureLog mode

The log retrieval algorithm for the Push and Pull modes is outlined in Fig.4. The algorithm presents logging and synchronization steps with the harmonizer in case of PureLog. First, the algorithm checks whether the size of the JAR has exceeded a stipulated size or the normal time between two consecutive dumps has elapsed. The size and time threshold for a dump are specified by the data owner at the time of creation of the JAR. The algorithm also checks whether the data owner has requested a dump of the log files. If none of these events has occurred, it proceeds to encrypt the record and write the error-correction information to the harmonizer. The communication with the harmonizer begins with a simple handshake. If no response is received, the log file records an error. The data owner is then alerted through e-mails, if the JAR is configured to send error notifications. Once the handshake is completed, the communication with the harmonizer proceeds, using a TCP/IP protocol. If any of the abovementioned events (i.e., there is request of the log file, or the size or time exceeds the threshold) has occurred, the JAR simply dumps the log files and resets all the variables, to make space for new records. In case of AccessLog, the above algorithm is modified by adding an additional check after step 6. Precisely, the AccessLog checks whether the CSP accessing the log satisfies all the conditions specified in the policies pertaining to it. If the conditions are satisfied, access is granted; otherwise, access is denied. Irrespective of the access control outcome, the attempted access to the data in the JAR file will be logged. Our auditing mechanism has two main advantages. First, it guarantees a high level of availability of the logs. Second, the use of the harmonizer minimizes the amount of workload for human users in going through long log files sent by different copies of JAR files.

5.3 Security Discussion

We now analyse possible attacks to our framework. Our analysis is based on a semihonest 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.

5.3.1 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. However, such attack will be detected by our auditing mechanism. Recall that every JAR file is required to send log records to the harmonizer. The logger component provides more transparency than conventional log files encryption; it allows the data owner to detect when an attacker has created copies of a JAR, and it makes offline files inaccessible

5.3.2 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. We rely on the strength of the cryptographic schemes applied to preserve the integrity and confidentiality of the logs.

5.3.3 Man-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 deception as a rightful 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. The first type of attack will not succeed since the certificate typically has a time stamp which will become outdated at the time point of reuse. The second type of attack will also fail since renegotiation is banned in the latest version of OpenSSL and cryptographic checks have been added.

5.3.4 Compromised JVM Attack

An attacker may try to compromise the JVM. To quickly detect and correct these issues, we apply oblivious hashing to guarantee the correctness of the JRE and how to correct the JRE prior to execution, in case any error is detected. To further strengthen our solution, one can extend OH usage to guarantee the correctness of the class files loaded by the JVM.

VI. 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 not only audit his content but also enforce strong back-end protection if needed. Moreover, 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

Journal Papers:

[1]. Marco Casassa Mont, Ilaria Matteucci, Marinella Petrocchi, Marco Luca Sbodio, "Data Sharing in the Cloud*", Hewlett-Packard Labs,* Feb 2012

Proceedings Papers:

[2]. S. Sundareswaran, A. Squicciarini, D. Lin, " Distributed Accountability for Data Sharing in the Cloud," *Proc. IEEE Transactions on Dependable and Secure Computing, Vol. 9, No.4,* Aug. 2012

[3]. S. Pearson, "Towards Accountability in the Cloud ," *Proc. IEEE Internet Computing,* pp. 64-69, 2011

[4]. R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely, "Towards a Theory of Accountability and Audit," *Proc. 14th European Conf. Research in Computer Security (ESORICS),* pp. 152-167, 2009.

[5]. 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),* pp. 145-154, 2009.

[6]. A. Squicciarini, S. Sundareswaran, and D. Lin, "Preventing Information Leakage from Indexing in the Cloud," *Proc. IEEE Int'l Conf. Cloud Computing,* 2010.

[7]. S. Sundareswaran, A. Squicciarini, D. Lin, and S. Huang, "Promoting Distributed Accountability in the Cloud," *Proc. IEEE Int'l Conf. Cloud Computing,* 2011.