Volume 3, Issue 3, March 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Ensuring Distributed Accountability for Data Sharing in the Cloud

Pankaj Kumar Singh
C.S.E, Bharath University
India.

Ajit kumar
C.S.E, Bharath University
India.

R.Karthikeyan
Lecturer C.S.E. Department, Bharath university
India.

Abstract- Cloud computing is the use of computing of sources (hardware and software) that are delivered as a service over a network (typically the internet).It enables highly scalable services to be easily consumed over the Internet on an as needed basis. A major characteristic of the cloud services is that users' data are usually processed remotely in unknown machines that users do not operate. It can become a substantial roadblock to the wide adoption of cloud services. To address this problem, we propose a highly decentralized answerability framework to keep track of the actual usage of the user's data in the cloud. 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 elements: logger and log harmonizer. The proposed methodology will also take concern of the JAR file by converting the JAR into obfuscated code which will adds an additional layer of security to the infrastructure. Apart from that we are going to increase the security of user's data by provable data possessions for integrity verification.

Keywords-- Cloud computing, data sharing, information accountability framework, Provable data possession.

I. Introduction

The Cloud Information Accountability framework [1] proposed in this work conducts automated logging and distributed auditing of relevant access performed by any entity, taken out at any point of time at any cloud service provider. It has two major components: logger and log harmonizer. The JAR file includes a set of simple access control rules specifying whether and how the cloud servers and possibly other data stakeholders are authorized to access the content itself. Apart from that we are going to check the integrity of the JRE on the systems on which the logger components is initiated. This integrity checks are carried out by using oblivious hashing .The proposed methodology will also take concern of the JAR file by converting the JAR into obfuscated code which will adds an additional layer of security to the infrastructure. Apart from that we are going to enlarge the security of user's data by provable data possessions for integrity verification. Based on the configuration settings defined at the time of creation, the JAR will give usage control associated with logging, or will give only logging functionality. As for the logging, every time there is an access to the data, the JAR will automatically produce a log record.

II. Existing System

Cloud computing is the delivery of computing as a service rather than a product, by which shared resources, software, and information are given to computers and other devices as a utility like the electricity grid over a network (typically the Internet). In these days a single server deals with the multiple requests from the user. Here the server has to operate the both the request from the user simultaneously, so the processing time will be high. This may leads to deficit of data and packets may be delayed and corrupted and also the Data Management and the Services are not Trust Worthy. While enjoying the convenience brought by this new technology, users also start bothering about losing control of their own data. The data operated on clouds are often outsourced, which lead to a number of issues related to accountability, including the management of personally identifiable information. To allay users' concerns, it is necessary to provide an effective mechanism for users to monitor the usage of their data in the cloud. For example, users required to be able to ensure that their data are handled according to the service level agreements made at the time they sign on for services in the cloud. Conventional access control approaches made for closed domains such as databases and operating systems, or approaches with a centralized server in distributed environments, are not suitable, because of the following features characterizing cloud environments.

2.1. Disadvantages

Although the Cloud computing is vast developing technology, the database management system does not have a trustworthiness.

III. Proposed System

To overcome the above problems, we propose a novel method, namely Cloud Information Accountability (CIA) framework, based on the notion of information accountability. Data Owner can upload the data into the cloud server after encrypted the data. User can subscribe into the cloud server with certain access polices such as read, write and copy of the original data. The Loggers and Log Harmonizer will have a track of the access logs and reports to the data owner. This Process ensures security.

3.1. Advantages

We can share the data in a secured manner.

3.2. Modification

Automatic reporting of illegal action performance of any use to the data owner, along with data owner would generate the random numbers set for the every user. So if the user accessing the account, the user has to give the random number set and that will be verified by the server. If the verification resultant is positive then only the user will be allowed to access their account.

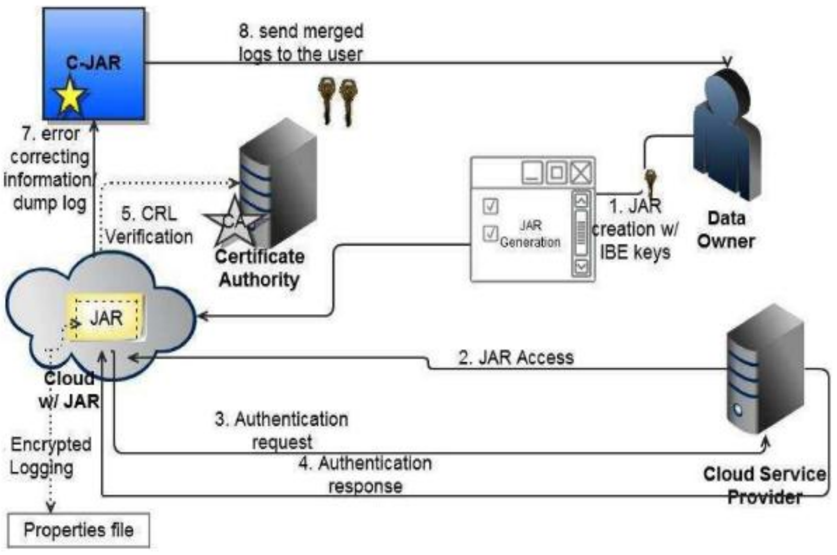

Fig.1. Overall Architecture

IV. Implementation

4.1. Algorithm Evolution

The algorithm here used is Log Retrieval Algorithm for push and pull modes.

The algorithm presents logging and synchronization steps with the harmonizer in case of Pure Log. 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 development of the JAR. The algorithm also determines whether the data owner has requested a dump of the log files. If none of these events has happened, it proceeds to encrypt the record and write the error-correction information to the harmonizer. The interaction 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 via emails, if the JAR is configured to send error notifications. Once the handshake is done, the interaction with the harmonizer proceeds, using a TCP/IP protocol. If either of the aforementioned events (i.e., there is request of the log file or the size or time exceeds the threshold) has happened, the JAR simply dumps the log files and resets all the variables, to make a space for new records. In case of Access Log, the above algorithm is modified by adding an additional check after step 6. Precisely, the Access Log checks whether the CSP accessing the log satisfies all the conditions specified in the policies pertaining to it. If the conditions are fulfilled, access is granted; otherwise, access is declined. Irrespective of the access control outcome, the attempted access to the data in the JAR file will be logged. Our auditing mechanism has two fundamental advantages. First, it guarantees a high level of availability of the logs. Second, the usage of the harmonizer minimizes the amount of workload for human users in going through long log files sent by different copies of JAR files.

4.2. Modules

- User/Data Owner
- Cloud Sever
- Certificate Authority
- Logger
- Access Privileges
- Push And Pull
- Random Set Generation And Verification

4.2.1. User/Data Owner

User is the person is going to see or download the data from the Cloud server. To access the data from the Cloud server, the users have to be registered with the cloud server. So that the user have to register their details like username, password and a set of random numbers. This is the information that will stored in the database for the future authentication.

Data Owner is the Person who is going to upload the data in the Cloud Server. In order to upload the data into the Cloud server, the Data Owner have to be registered in the Cloud Server. Once the Data Owner registered in cloud server, the space will be assigned to the Data Owner.

4.2.2. Cloud Server

Cloud Server is the area where the user going to request the data and also the data owner will upload their data. Once the user send the request regarding the data they want, the request will ne first send to the Cloud Server and the Cloud Server will forward your request to the data owner. The data Owner will send the data the data the user via Cloud Server. The Cloud Server will also manage the Data owner and Users information in their Database for future purpose.

4.2.3. Logger

The Logger is maintained by the Cloud Server. Loggers have the details of the data owner and users who are accessing the Cloud Server. So the Logger will be more useful for many purposes.

Like which user / data owner accessing the Cloud Server, accessed at the particular time and the IP address from which the data is requested by user etc.

*4.2.4. **Certificate Authority***

The Certificate Authority is used to verify the Cloud Server is recognized or not. The Cloud Server has to be recognized by the certificate authority. If not recognized, the Cloud Server is a Fraudulent Server. The data owner can check the whether the recognized or not. Because the data owner is going to upload their data in the Cloud Server.

4.2.5. ***Access Privileges***

The access privileges are set by the data owner for accessing their data. Some Owners will provide read only, some of them will allow read and download. The Cloud Server will send the dynamic intimation when the user is accessing the data beyond their limits. This increases more security while sharing the data in the Cloud.

4.2.6. Push and Pull Concept

4.2.6.1. Push

For the every periodical time the Cloud Server will send the access details of the user to the data owner. So that the Data Owner may able to know who're all the accessing their data at the particular time period. During the registration phase, the Data owner will ask by the Cloud Server whether they're choosing the push or pull method

4.2.6.2. Pull

In the Pull method, the data owner has to send the request to the Cloud Server regarding the access details of their data up to the particular time. Then the Cloud Server will send the response to the Data Owner regarding the user's access details.

4.2.7. Random Set Generation and Verification

When the user request the data to be downloaded from the Cloud Server, the user have to enter the Random number set. If it is matched, the user is allowed to download the data. The Random number sets will be provide to the user during the registration Phase itself. Each and Every time the Random number set will vary. This ensures security while downloading the data.

V. Experiment And Result

We first bring out the settings of the test environment and then present the performance study of our system.

5.1. Experimental Settings

We tested our CIA framework by setting up a small cloud, using the Emu lab test bed In particular, the test environment consists of several Open SSL-enabled servers:

1. Notice that we do not consider the attack on the log harmonizer component, since it is saved separately in either a secure proxy or at the user end and the attacker typically cannot access it. As a result, we consider that the attacker cannot extract the decryption keys from the log harmonizer. 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 on the basis of Amazon EC2, so bringing the powerful functionalities of Amazon EC2 into the open source domain. We set to work Linux-based servers running Fedora 10 OS. Each server has a 64-bit Intel Quad Core Xeon E5530 processor, 4 GB RAM, and a 500 GB Hard Drive. Each of the servers is arrayed to run the Open JDK runtime environment with IcedTea6 1.8.2.

5.2. Experimental Results

In the experiments, we first examine the time taken to create log file and then measure the overhead in the system. With respect to time, the overhead can occur at three points: at the time of the authentication, during encryption of a log record, and at the time of the merging of the logs. Also, with respect to storage overhead, we notice that our architectures very lightweight, in that the only data to be stored are provided by the actual files and the associated logs. Further, JAR appear as a compressor of the files that it handles. In particular, as proposed, multiple files can be managed by the same logger component. To this extent, we checked whether a single logger component, used to managed more than one file, results in storage overhead.

5.2.1 Log Creation Time

In the first round of experiments, we are concerned in
finding out the time taken to create a log file when there are entities continuously accessing the data, causing continuous logging. Resultants are shown in Fig. 3. It is not surprising to identify that the time to create a log file increases linearly with the size of the log file. Specifically, the time to develop 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 figure out the amount of time to be specified between dumps, keeping other variables like space constraints or network traffic in mind.

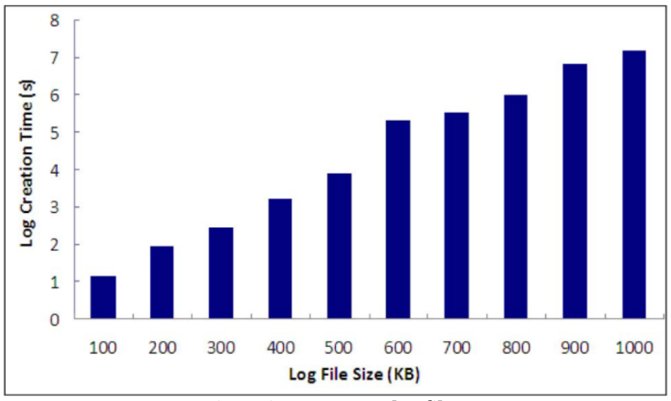

Fig.2.Time to merge log files.

5.2.2 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 investigated. Considering one access at the time, we got that the authentication time averages around 920 ms which proves that not too much overhead is added during this phase. As of present, the authentication takes place each time the CSP needs to access the data. The performance can be further improved by caching the certificates. The time for authenticating an end user is about the same when we consider only the actions required by the JAR, viz. acquiring a SAML certificate and then evaluating it. This is due to both the OpenSSL and the SAML certificates are handled in a similar fashion by the JAR. When we consider the user actions (i.e., submitting his username to the JAR), it averages at 1.2 minutes.

5.2.3 Time Taken to Perform Logging

This set of experiments studies the effect of log file size on the logging performance. We measured the average time taken to allow an access plus the time to write the corresponding log record. The time for allowing 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. Every access is just a view request and hence the time for executing the action is negligible. As a resultant, the average time to log an action is about 10 seconds, which involves the time taken by a user to double click the JAR or by a server to run the script to open the JAR. We also took the log encryption time which is about 300 ms (per record) and is seemingly unrelated from the log size.

5.2.4 Log Merging Time

To check if the log harmonizer can be a bottleneck, we have taken the amount of time required to merge log files. In this experiment, we confirmed that each of the log files had10 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. We tested the time to merge up to 70 log files of 100 KB, 300 KB, 500 KB, 700 KB, 900 KB, and 1 MB each. The results are shown in Fig.3. We can see that the time increases almost linearly to the number of files and size of files, with the least time being acquired for merging two100 KB log files at 59 ms, while the time to merge 70 1 MB files was 2.35 minutes.

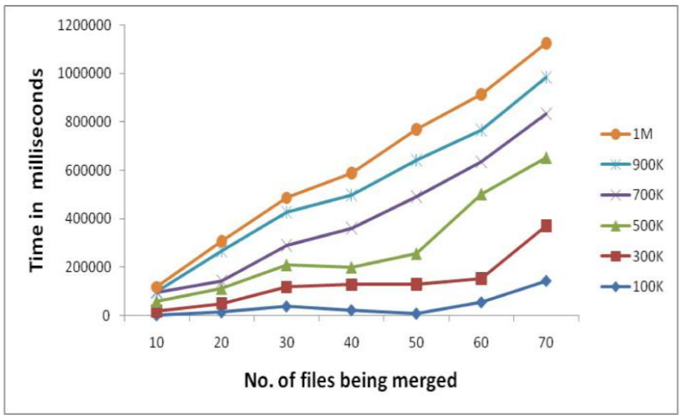

Fig.3. Time to create log files of different sizes.

5.2.5 Size of the Data JAR Files

Finally, we investigate whether a single logger, account to handle more than one file, results in storage overhead. We have figured out the size of the loggers (JARs) by varying the number and size of data items held by them. We tested the accession in size of the logger containing 10 content files (i.e., images) of the same size as the file size expands. Intuitively, in case of large size of data items held by logger, the overall logger also expand in size. The size of logger grows from 3,500 to 4,035 KB when the size of content items changes from 200 KB to 1 MB. Overall, because of the compression provided by JAR files, the size of the logger is commanded by the size of the largest files it holds. Notice that we purposely did not include large log files (less than 5 KB), so as to give attention on the overhead added by having multiple content files in a single JAR. Resultants are in Fig.4.

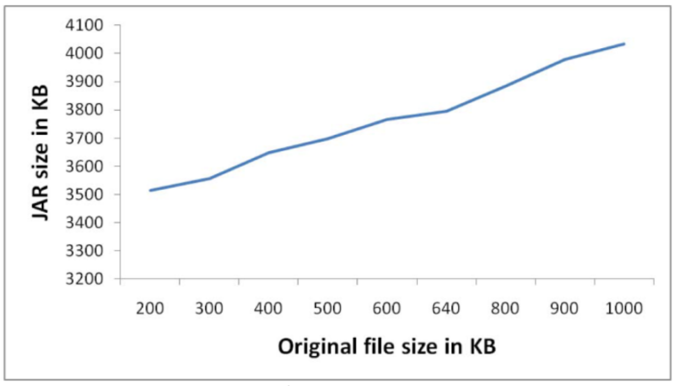

Fig.4. Size of the logger Component

5.2.6 Overhead Added by JVM Integrity Checking

We investigate the overhead added by both the Reinstallation/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 figured 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, we simply measure the time acquired for each hash calculation. This time is taken to average around 9 ms. The number of hash commands varies on the basis of size of the code in the code does not change with the content, the number of hash commands remain constant.

V. Conclusion

We introduced modern 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. Apart from that we have enclosed PDP methodology to enhance the integrity of owner's data. In future, we plan to refine our approach to verify the integrity of JRE. For that we will look into whether it is possible to leverage the advantage of secure JVM being developed by IBM and we would like to enhance our PDP architecture from user end which will allow the users to check data remotely in an efficient manner in multi cloud environment

References

1. Text Books

[1] *Cloud Computing, Principles and Paradigms* by John Wiley & Sons.

2. *Conference Proceedings*

[1] *Ensuring Distributed Accountability for Data Sharing in the Cloud Author*, Smitha Sundareswaran, Anna C.Squicciarini, Member, IEEE, and Dan Lin, *IEEE Transactions on Dependable and Secure Computing* ,VOL 9,NO,4 July/August 2012 .

[2] Hsio Ying Lin,Tzeng.W.G, **"***A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding* ",IEEE transactions on parallel and distributed systems,2012.

[3] Yan Zhu, Hongxin Hu, Gail Joon Ahn, *Mengyang Yu, "Coopera-tive Provable Data Possession for Integrity Verification in Multi-Cloud Storage*" , IEEE transactions on parallel and distributed systems,2012.

3. Generic Website

[1] *Eucalyptus Systems*, http://www.eucalyptus.com/, 2012.

[2] *Emulab Network Emulation Testbed*, www.emulab.net, 2012.