ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2 Issue 8, August 2013

SECURING DISTRIBUTED ACCOUNTABILITY FOR DATA SHARING IN CLOUD COMPUTING

N. RAMESH [1], D. ANIL$_{M.E}$[2], M. KIRAN[3]

1, M.TECH Scholar, VEC, Kavali
2, Assistant Professor, VEC, Kavali
3, Assistant Professor, JCET

ABSTRACT:

The growing trend towards grid computing and cloud computing provides enormous potential for allowing dynamic, distributed and data demanding applications such as sharing and processing of large-scale scientific data. Cloud computing is the use of computing of sources (hardware and software) that are delivered as a service over a network. A main characteristic of the cloud services is that users' data are usually processed remotely in unknown machines that users do not operate. It can be converted into a significant roadblock to the wide adoption of cloud services. To deal with this problem, we suggest a highly decentralized accountability framework to keep track of the actual usage of the user's data in cloud. The Cloud Information Accountability framework projected in this work conducts automated logging and distributed auditing of relevant access performed by anything, passed out at any point of time at any cloud service provider. It has two main elements: logger and log harmonizer. The proposed methodology will take concern of the JAR file by converting the JAR into obfuscated code which will adds an additional layer of security to the communications. Apart from that we are going to enlarge the security of user's data by provable data possessions for integrity verification.

Index Terms— ***Cloud computing, information accountability framework, data sharing, Provable data possession***

1. INTRODUCTION

The Cloud Information Accountability framework proposed conduct the automated logging and distributed auditing of relevant access performed by anything, taken out at any point of time at any cloud service provider. It has two main components: logger and log harmonizer. The JAR file includes a set of simple controlling rules identifying whether and how the cloud servers and possibly other data stakeholders are authorized to access the content itself. Away from each other 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 procedure control connected with logging, or will give only logging functionality. When for the logging, every time there is an access to the data, the JAR will automatically produce a log record.

2. PROBLEM STATEMENT

A user, who registered to a certain cloud service, generally needs to send his data as well as associated access control policies to the service provider. After the data are received by the cloud service provider, the service supplier will have approved access rights, such as write, read, and copy, on the data. Using conservative access control mechanisms, once the access rights are approved, the data will be fully available at the service provider. In organize to track the actual usage of the data; we aim to develop novel logging and auditing techniques which satisfy the following requirements:

1. The logging should be decentralized in order to get used to the dynamic nature of the cloud. More specifically, log files should be strongly bounded with the equivalent data being controlled, and require minimal infrastructural maintain from any server.
2. Every access to the user's data should be properly and automatically logged. This requires integrated methods to authenticate the entity that accesses the data, verify, and record the actual processes on the data as well as the time that the data have been right to used.
3. Log files should be consistent and tamper proof to avoid illegal deletion, insertion, and alteration by malicious parties. Healing mechanisms are also desirable to restore damaged log files caused by technical problems.
4. Log files should be drive back to their data holder periodically to inform them of the current usage of their data. The log files should be retrievable at anytime by their data owners when needed in spite of the location where the files are stored.

3. CLOUD INFORMATION AND ACCOUNTABILITY

The Cloud Information Accountability framework proposed in this work conducts automated logging and distributed auditing of relevant access performed by anything, carried out at any point of time at any cloud service provider. It has two major works: logger and log harmonizer.

3.1. Data Flow

The overall CIA framework, combining data, users, logger and harmonizer at the beginning, each user generates a pair of public and private keys depend on Identity-Based Encryption. This IBE method is a Weil-pairing-based IBE scheme, using the generated key, the user will construct a logger component which is a JAR file, to store up its data items. The JAR file contains 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. Then the user sends the JAR file to the cloud service supplier that he subscribes to. To authenticate the CSP to the JAR, we utilize Open SSLbased certificates, where in a trusted certificate authority certifies the CSP. In the occasion that access is requested by a user, we make use of SAML-based validation, where in a trusted uniqueness provider issues certificates verifying the user's identity based on his username.

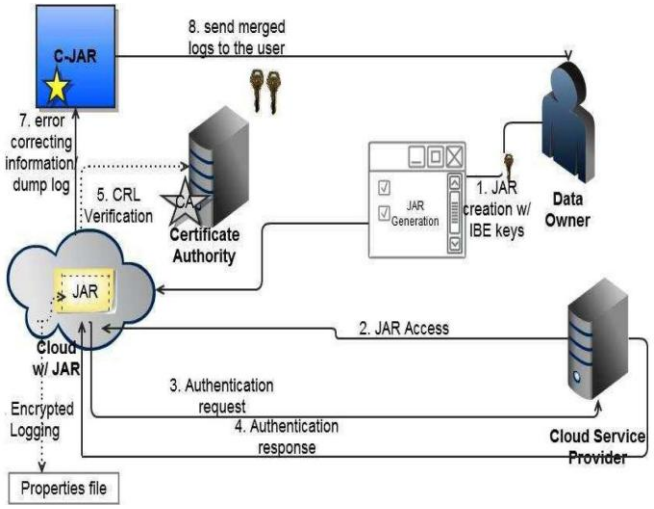

Fig.1: The cloud information accountability Framework

Once the authentication is successful, the service provider or the user will be allowed to

access the data together with this JAR. Depending on the design settings defined at the instance of establishment, the JAR will provide usage control related with logging, or will give only logging functionality. As for the logging, each instance there is an access to the data; the JAR will routinely generate a log record, encrypt it using the public key dispersed by the data owner, and pile up it along with the data. The encryption of the log file put a stop to unauthorized changes to the file by attackers. The data owner could decide on to use again the same key pair for all JARs or create particular key pairs for separate JARs. Using separate keys can improve the security without set up any overhead except in the initialization phase.

A customer supplies some perceptive data in a file inside a virtual machine (VM) hosted by a supplier that has subscribed to. Upon uploading the data, fail safe methods within the cloud will naturally back it up, and execute load balancing by generates redundancies across a number of virtual servers and physical servers in the service provider's reliance domain. From the file's construction to the backup processes, large numbers of data transport occur across virtual and physical servers and several memory read/write transactions to both virtual and physical memories are concerned. If all such transactions and the creation of new second copy files are logged, checked and accounted for, we would be able to mark out the file history and log the access history and comfortable modifications, i.e. achieving cloud accountability and audit ability

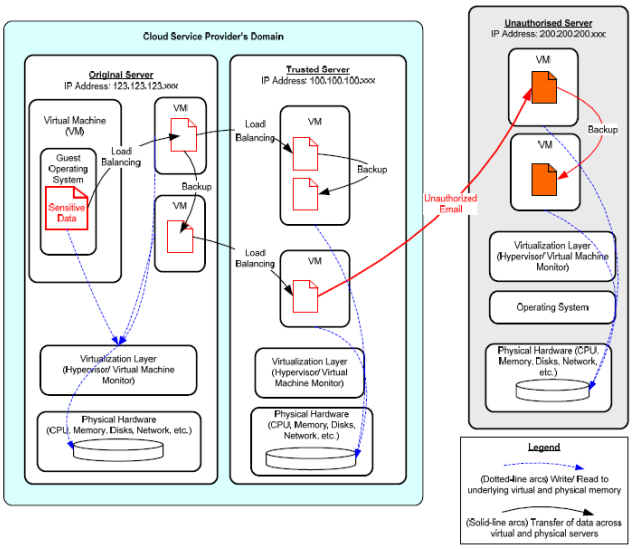

Fig.2: The importance of accountability and Audit ability

4. AUTOMATED LOGGING MECHANISM

4.1. The Logger Structure

We leverage the programmable capability of JARs to conduct programmed logging. A logger constituent is a nested Java JAR file which stores a user's data items and equivalent log files. The projected JAR file consists of one outer JAR surrounding one or more inside JARs. The main dependability of the outer JAR is to handle validation of entities which want to access the data stored in the JAR file. The data owners may not know the exact CSPs that are going to handle the data and so, authentication is individual according to the server's functionality rather than the server's URL or identity. Suppose a policy may state that Server X is acceptable to download the data if it is a storage server. The outer JAR may also have the access control functionality to enforce the data owner's necessities, specified as Java policies, on the usage of the data. A Java policy identifies which permissions are available for a particular piece of code in a Java application environment. The permissions articulated in the Java policy are in terms of File System Permissions. Though, the data owner can specify

the permissions in user-centric terms as opposed to the usual code-centric security offered by Java, using Java validation and Authorization Services. In addition, the outer JAR is also in charge of selecting the correct internal JAR according to the identity of the entity who requests the data

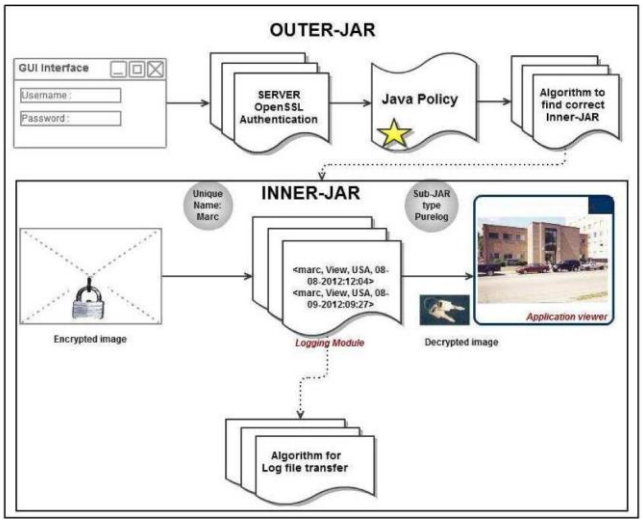

Fig.3: Structure of JAR file

4.2. Log Record Generation

Log records are generated by the logger component. Logging take place at any access to the data in the JAR, and new log entries are attached sequentially, in order of creation LR (r_1, . . . r_k). Each record r_i is encrypted individually and appended to the log file. In exacting, a log record obtains the following form

$$r_i = \langle ID, Act, T, Loc, h((ID, Act, T, Loc)|r_i - 1| \ldots |r_1), sig \rangle.$$

Here, ri indicates that an entity identified by ID 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)|….|r_1)corresponds to the checksum of the records preceding the newly introduce one, concatenated with the main content of the record itself. The checksum is calculated using a collision free hash function. The component sig indicates the signature of the record created by the server. If more than one file is switched by the same logger, an additional Obj ID field is added to each record.

5. END-TO-END AUDITING MECHANISM

End to End auditing mechanism describes our distributed auditing mechanism including the algorithms for data owners to query the logs regarding their data.

5.1. 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:

The logs are periodically pushed to the data owner (or auditor) by the harmonizer. The push achievement will be activated by either type of the following two events: one is that the time passes by 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 predetermined by the content owner at the moment of creation. After the logs are sent to the data owner, the log files will be discarded, so as to free the space for future access logs. The length of with the log files, the error accurate information for those logs is also dumped.

Pull mode:

This mode agrees to auditors to recover the logs anytime when they want to check the recent access to their own data. The pull note consists of an FTP pull command, which can be concerns from the command line. For immature users, a wizard containing a batch file can be effortlessly built. The request will be sent to the harmonizer, and the user will be up to date of the data's locations and obtain an integrated copy of the genuine and sealed log file.

6. EXPERIMENTAL RESULTS

In the experiment examine the time taken to create a log file and then measure the overhead in the system. Through respect to time, the overhead can occur at three points: during the validation, in encryption of a log record, and in the merging of the logs and with respect to storage overhead, we observe that our architecture is very light weight, in that the barely data to be stored are given by the real files and the connected logs. Additionally, JAR act as a compressor of the files that it handles. In many files can be handled by the same logger component. To this level, we examine whether a single logger component, used to handle extra than one file, the outcome in storage overhead.

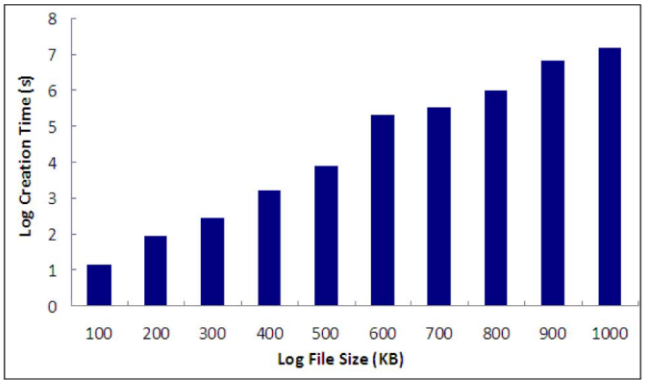

Fig.4: Time to create log files of different sizes.

6.1. Log Creation Time

In this, 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. It is not surprising to see that the time to create a log file increases linearly with the size of the log file. particularly, the time to generate a 100 Kb file is about 114.5 ms though the time to create a 1 MB file averages at 731 ms. With this trial as the baseline, one can make a decision the amount of time to be specified between dumps, maintenance other variables like space restrictions or network passage in mind.

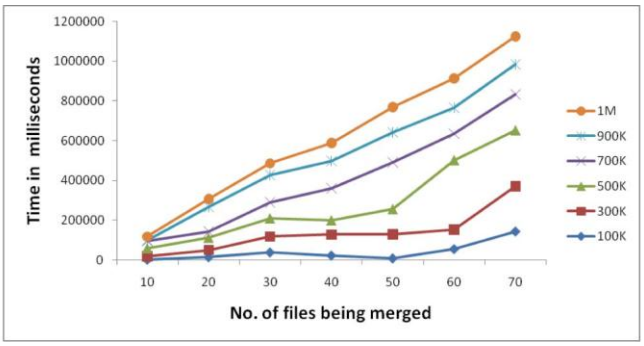

Fig. 6: Time to merge log files.

6.2. Time Taken to Perform Logging

This set of experiments studies the effect of log file size on the logging performance. We compute the average time taken to grant an access plus the time to write the corresponding log record. The time for giving way 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.

7. CONCLUSION

We initiate modern move toward automatically logging any access to the data in the cloud mutually with an auditing mechanism. Our approach allows the data holder to not only audit his content but also enforce strong back-end protection if needed. At a distance from that we have enclosed PDP methodology to improve the integrity of owner's data and we plan to refine our approach to verify the integrity of JRE. For that we will seem to be into whether it is possible to control the advantage of secure JVM being developed by IBM and we would like to enhance our PDP architecture from consumer end which will allow the consumer to check data distantly in an efficient manner in multi cloud environment.

VIII. REFERRENCES

[1] B. Chun and A.C. Bavier, "Decentralized Trust Management and Accountability in Federated Systems," Proc. Ann. Hawaii Int'l Conf. System Sciences (HICSS), 2004

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. ACM Conf. Computer and Comm. Security, pp. 598-609, 2007.
[3] E. Barka and A. Lakas, "Integrating Usage Control with SIP-Based Communications," J. Computer Systems, Networks, and Comm., vol. 2008, pp. 1-8, 2008.
[4] D. Boneh and M.K. Franklin, "Identity-Based Encryption from the Weil Pairing," Proc. Int'l Cryptology Conf. Advances in Cryptology, pp. 213-229, 2001.
[5] R. Bose and J. Frew, "Lineage Retrieval for Scientific Data Processing: A Survey," ACM