Indian Journal of Communication Engineering and Systems (IJCES)
Vol.2.No.2 2014 pp 101-106.
Available at: www.goniv.com
Paper Received: 04-04-2014
Paper Accepted: 14-04-2014
Paper Reviewed by: 1. R. Venkatakrishnan 2. R. Marimuthu
Editor: Prof. P.Muthukumar

SECURED DATA SHARING IN CLOUD USING DISTRIBUTED ACCOUNTABILITY

S.Sowmiya
M.Tech (C.S.E)-2Yr,
PRIST University.,
Thanjavur-613 403.
Email:sowmiya.rajanit@gmail.com

R.Muthuvenkatakrishnan
Associate Professor
PRIST University.,
Thanjavur-614303
muthubrillia@gmail.com

ABSTRACT

Cloud computing enables highly scalable services to be easily consumed over the Internet on an as-needed basis. 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. While enjoying the convenience brought by this 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. To address this problem, in this paper, 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-centred approach that enables enclosing our logging mechanism together with users' data and policies. We leverage 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. We provide extensive experimental studies that demonstrate the efficiency and effectiveness of the proposed approaches.

Index Terms—Cloud computing, accountability, data sharing.

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. To date, there are a number of notable commercial and individual cloud computing services, including Amazon, Google, Microsoft, Yahoo, and Sales force. Details of the services provided are abstracted from the users who no longer need to be experts of technology infrastructure. Moreover, 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 barrier to the wide adoption of cloud services. To allay users' concerns, it is essential to provide an effective mechanism for users to monitor the usage of their data in the cloud. For example, users need 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 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. First, data handling can be outsourced by the direct cloud service provider (CSP) to other entities in the cloud and theses entities can also delegate the tasks to others, and so on. Second, entities are allowed to join and leave the cloud in a flexible manner. As a result, data handling in the cloud goes through a complex and dynamic hierarchical service chain which does not exist in conventional environments.

To overcome the above problems, we propose a novel approach, namely Cloud Information Accountability (CIA) framework, based on the notion of information accountability. 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. 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. By means of the CIA, data owners can track not only whether or not the service-level agreements are being honored, but also enforce access and usage control rules as needed. 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. 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. 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. We refer to this type of enforcement as "strong binding" since the policies and the logging mechanism travel with the data. This strong binding exists even when copies of the JARs are created; thus, the user will have control over his data at any location.

Such decentralized logging mechanism meets the dynamic nature of the cloud but also imposes challenges on ensuring the integrity of the logging. To cope with this issue, we provide the JARs with a central point of contact which forms a link between them and the user. It records the error correction information sent by the JARs, which allows it to monitor the loss of any logs from any of the JARs.

Moreover, if a JAR is not able to contact its central point, any access to its enclosed data will be denied. Currently, we focus on image files since images represent a very common content type for end users and organizations (as is proven by the popularity of Flickr) and are increasingly hosted in the cloud as part of the storage services offered by the utility computing paradigm featured by cloud computing. Further, images often reveal social and personal habits of users, or are used for archiving important files from organizations. In addition, our approach can handle personal identifiable information provided they are stored as image files(they contain an image of any textual content, for example, the SSN stored as a.jpg file).

2 RELATED WORK

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. First, data handling can be outsourced by the direct Cloud Service Provider to other entities in the cloud and theses entities can also delegate the tasks to others, and so on. Second, entities are allowed to join and leave the cloud in a flexible manner.

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. It is essential to provide an effective mechanism for users to monitor the usage of their data in the cloud. An object-cantered approach that disable enclosing our logging mechanism together with users' data and policies.

Cloud computing has raised a range of important privacy and security issues . 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. To date, little work has been done in this space, in particular with respect to accountability.

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 deobfuscated 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. In the authors present a layered architecture for addressing the end-to-end trust management and accountability problem in federated systems.

The focus is very different from ours, in that they mainly leverage trust relationships for accountability, along with authentication and anomaly detection. Further, their solution requires third-party services to complete the monitoring and focuses on lower level monitoring of system resources.

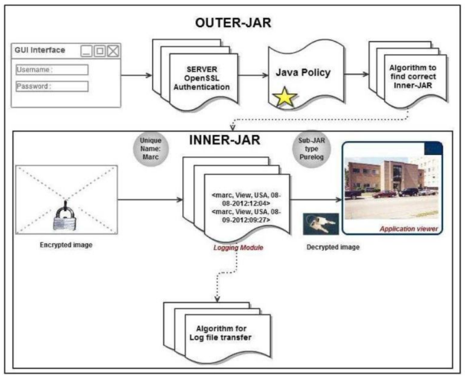
OUTER-JAR
GUI Interface
Username:
Password:
SERVER OpenSSL Authentication
Java Policy
Algorithm to find correct Inner-JAR
INNER-JAR
Unique Name: Marc
Sub-JAR type Purelog
<marc, View, USA, 08-08-2012:12:04>
<marc, View, USA, 08-09-2012:09:27>
Encrypted image
Logging Module
Decrypted image
Application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

Fig.1.Structure of JAR Files

3 PROPOSED DESIGN

To overcome the above problems, Cloud Information Accountability framework, based on the notion of information accountability. 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 trackable. 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.. By means of the CIA, data owners can track not only whether or not the service-level agreements are being honored, but also enforce access and usage control rules as needed. 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 can retrieve the logs as needed.

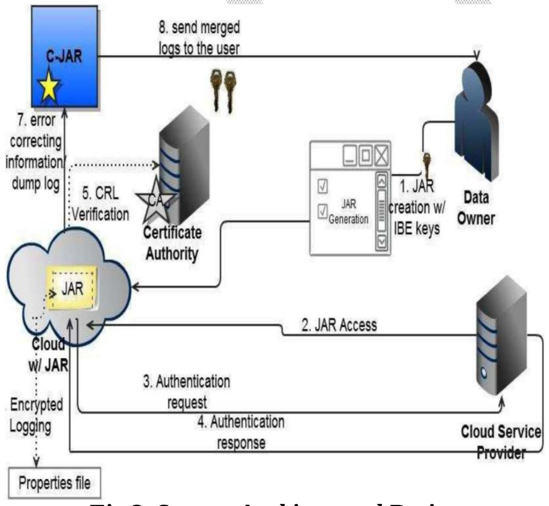

Fig.2. System Architectural Design

3.1 Login:

Direct Entry skips the Gateway and Login pages when users access the URL . When users access the system through Portal Direct Entry, they are considered Guests until they log in. The Login Module is a portal module that allows users to enter a User Name and Password to log in. This module can be placed on any Module Tab to allow users to login to the system. If the Administrator has allows users to create accounts and enabled Portal Direct Entry, a Create Account link appears in the Login Module.

3.2 New Register:

Simple registration page with First Name, Last Name, Email, Password, Address and Mobile number. The new register module page for user to amend those details. Aside from the fact that at points it doesn't actually seem to save anything into the DB the biggest issue is that neither of them seem to perform any checks on whether a user already exists. If someone tries to register using the same email address as one already registered it seems to just replace the existing user account.

3.3 Admin Module:

Maintain permanent historical PH records for customers of products and services. Control the Scheduling of Training, Resources, Facilities, Instructors and Students. Design and monitor PHR Programs for people based on their position, or a health they need to perform.

3.4 View:

The entity can only read the data but is not allowed to save a raw copy of it anywhere permanently. For this type of action, the Pure Log will simply write a log record about the access, while the Access Logs will enforce the action through the enclosed access control module. Recall that the data are encrypted and stored in the inner jar. When there is a view-only access request, the inner JAR will decrypt the data on the fly and create a temporary decrypted file. The decrypted file will then be displayed to the entity using the Java application viewer in case the file is displayed to a human user. Presenting the data in the Java application, viewer disables the copying functions using right click or other hot keys such as Print Screen. Further, to prevent the use of some screen capture software, the data will be hidden whenever the application viewer screen is out of focus.

3.5 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 Pure log will just record the access starting time and its duration, while the Access Log will enforce that the access is allowed only within the specified period of time.

3.6 Time Protocol:

To enforce the limit on the duration, the Access Log records the start time using the ntp, and then uses a timer to stop the access. Naturally, this type of access can be enforced only when it is combined with the View access right and not when it is combined with the Download.

3.7 Location-based access:

In this case, the Pure Log will record the location of the entities. The Access Log 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.

3.8 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 Pure Log 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. In case of Access Logs, the entire jar file will be given to the Entity.

Algorithm :

```
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: 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)
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//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
```

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.

4 RESULTS AND DISCUSSIONS

In the experiments, we first 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. Further, JAR act as a compressor of the files that it handles. In particular, as introduced in, multiple files can be handled by the same logger component. To this extent, we investigate whether a single logger component, used to handle more than one file, results in storage overhead.

4.1 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. 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.

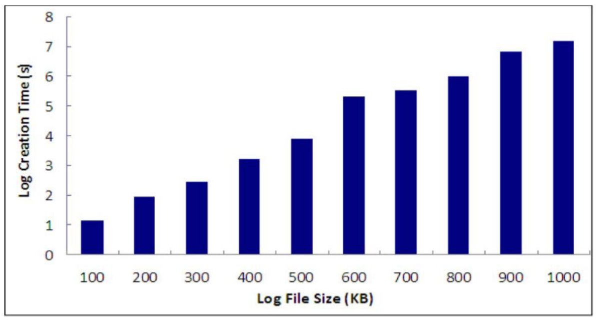

Fig.3. Time to create log files of Different sizes

4.2 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 .Considering one access at the time, we find 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. obtaining a SAML certificate and then evaluating it. This is because 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.

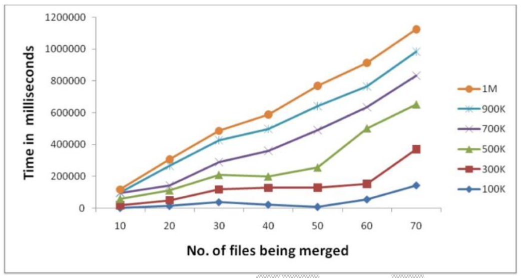

Fig.4. Time to merge log Files

4.3 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. We also measured the log encryption time which is about 300 ms (per record) and is seemingly unrelated from the log size.

4.4 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. 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. 6. We can observe that the time increases almost linearly to the number of files and size of files, with the least time being taken for merging two 100 KB log files at 59 ms, while the time to merge 70 1 MB files was 2.35 minutes.

4.5 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 (i.e., images) of the same size as the file size increases. Intuitively, in case of larger size of data items held by a logger, the overall logger also increases 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, due to the compression provided by JAR files, the size of the logger is dictated by the size of the largest files it contains. Notice that we purposely did not include large log files (less than 5 KB), so as to focus on the overhead added by having multiple content files in a single JAR.

4.6 Overhead Added by JVM Integrity Checking

We investigate the 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, we simply measure the time taken for each hash calculation. This time is found to average around 9 ms. 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.

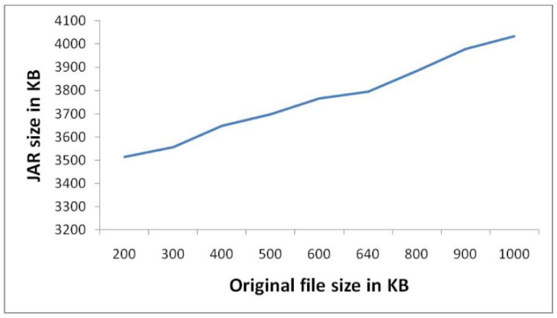

Fig 5 Size of the logger component

5 CONCLUSION

In this paper, 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 backend 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.

ACKNOWLEDGMENTS
The author would like to thank the management and staff members of the PRIST University for their help and their thoughtful comments.

REFERENCES

1. P. Ammann and S. Jajodia, (Aug.1993) "Distributed Timestamp Generation in Planar Lattice Networks," ACM Trans. Computer Systems, vol. 11, pp. 205-225.
2. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D.Song, (2007) "Provable Data Possession at Untrusted Stores," Proc. ACM Conf. Computer and Comm. Security, pp. 598-609.
3. E. Barka and A. Lakas, (2008) "Integrating Usage Control with SIP-Based Communications," J. Computer Systems, Networks, and Comm., vol. 2008, pp.
4. D. Boneh and M.K. Franklin, (2001) "Identity-Based Encryption from the Weil Pairing," Proc. Int'l Cryptology Conf. Advances in Cryptology, pp. 213-229.
5. R. Bose and J. Frew, (Mar. 2005) "Lineage Retrieval for Scientific Data Processing: A Survey," ACM Computing Surveys, vol. 37, pp. 1- 28.
6. P. Buneman, A. Chapman, and J. Cheney, (2006) "Provenance Management in Curated Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '06), pp. 539-550.
7. B. Chun and A.C. Bavier, (2004) "Decentralized Trust Management and
Accountability in Federated Systems," Proc. Ann. Hawaii Int'l Conf. System Sciences (HICSS).