Towards Achieving Secured and Decentralized Accountability in Cloud Computing

Drishya S G [#1], Kavitha Murugeshan*[2]

[#]*Student, MTech Computer science and Engineering, Vedavyasa Institute of Technology, Malappuram , Kerala, India*

[*]*Head of the Department of Computer Science and Engineering, Vedavyasa institute of technology, Malappuram , Kerala,India*

Abstract:-Cloud computing is an internet-based service where you can acquire networked storage space and computer resources as per your demand. An underlined feature in cloud computing is that users' data are usually processed in a remote computer which they cannot access. This creates a fear of losing data in users which is a major obstacle in wide range use of cloud services. To attend to this problem, we propose an object oriented concept to protect a user data in a cloud network and also after delivery of a data to a client. Here we also provide security to cloud users by rendering JAR authentication.

Keywords: Cloud Computing, JAR(Java Archive), Java Running Environment, Accountability, cloud Service Provider.

I.INTRODUCTION

Several trends are opening up the epoch of Cloud Computing, which is an Internet-based development using computer technology. The powerful processors which is cheaper together with the software as a service (SaaS) computing architecture, are metamorphosing data centres into pools of computing service on a huge scale. The broadening network bandwidth and impeccable yet flexible network connections make it even possible that users can now endorse high quality services from data and software that reside solely on remote data centers. Storing data into the cloud offers great convenience to users since they don't have to care about the complexities of direct hardware management.

Users may be unaware of the machines which actually process and host their data. So even when enjoying the convenience brought by this new technology, the users have worry about losing control of their own data. The data processed and stored on clouds are often outsourced, which leading to a number of issues related to security of data, including the handling of bank details and personal health details. Such fears are becoming a significant barrier to the wide embracement of cloud services. To avert these problems it is essential to provide an effective mechanism for users to monitor the usage of their data stored in the cloud. For example, users want to be able to ensure that their data that are stored in cloud are manipulated according to the service level agreements made at the time they sign on for services in the cloud.

Basic approach we use to look upon these issues is by using JAR (Java Archives) files to automatically log the usage of the users' data by any entity in the cloud. Users will send their data via any policies such as access control policies and logging policies they want to enforce, to cloud service providers. Cloud service provider encloses data in JAR files. Any access to the data will trigger an automated and authenticated logging mechanism local to the JARs. Since the policies and the logging mechanism travel with the data we refer to this type of enforcement as "strong binding". This type of strong binding exists even when copies of the JARs are created; thus, the users will have control over their data at any location we provide the JARs with a central point of contact which forms a link between them and the users. When a user copy the data from downloaded system to another then the JAR automatically create a log file which shows that the system is changed by tracking the IP address of the system. It records the error correction information sent by the JARs, which allows it to

track the loss of any logs from any of the JARs. Moreover, if the JAR is not able to contact its central point, access to the enclosed data will be denied.

The platform-independent Java Archive (JAR) file format allows you to compress and bundle multiple files associated with a Java application, Web Start application, or Applet into a single file. JAR is based on the ZIP algorithm. All files in a Java package can be placed in a single file to aid distribution. Furthermore, transfer of one big file over the network instead of many small files is faster and more efficient as it involves less overhead. Since we use JAR file, to provide security to client who access the data enclosed in it the authentication of jar file is substantial. This is needed because some malicious entity produces some JAR files which include programs capable of retrieving all the information stored in the client system. Here JAR file can be signed digitally by its author to provide JAR authentication.

II RELATED WORKS

Cloud computing has raised a range of important privacy and security issues [13], [15]. Main reasons of these issues are, in the cloud, users' data and applications reside—at least for a certain amount of time—on the cloud cluster which is maintained by a third party. To date, some work has been done in this area, in particular with respect to accountability. Pearson et al. [15] have proposed accountability mechanisms to address privacy concerns of end users and then develop a privacy manager [16]. Their proposed idea is that the user's private data are sent to the cloud using encryption technique, and the processing is done on the encrypted data. The result of the processing is deobfuscated by the privacy manager to get the correct result. However, privacy manager provides only limited features where it does not guarantee protection once the data are being disclosed. In [3], B. Chun and A.C. Bavier present a layered architecture for addressing the end-to-end trust management and accountability problem in federated systems. The authors' view is very different from ours, in that they mainly focus trust relationships for accountability, along with anomaly detection and authentication. Moreover, their proposed solution requires third-party services to complete the monitoring and focuses on lower level monitoring of system resources.

Smith Sundareswaran et al[2]. propose a novel highly decentralized information accountability framework to keep track of the actual usage of the users' data in the cloud using JAR concept. We follow the same concept as a base and extend the concept to track the cloud data even after copying using any device like pen drive. And also we extend to provide security to client by providing JAR authentication.

Researchers have investigated accountability mostly as a provable property through cryptographic mechanisms, [6], [10] particularly in the context of electronic commerce. In [5] authors propose the usage of policies attached to the data and present logic for accountability data in distributed settings. Similarly, Jagadeesan et al. , in [9], recently proposed a logic for designing accountability-based distributed systems. Crispo and Ruffo ,in [6], proposed an approach related to accountability in case of delegation.

Lee and colleagues [11] proposed an agent-based system for grid computing where static software agents used are track the distributed jobs, with the resource consumption at each local machines. Here the notion of accountability policies is mainly focused on resource consumption and on tracking of sub jobs processed at multiple computing nodes, rather than access control. In [12] authors describe a Method for Authenticating a Java Archive (jar) for Portable Devices.

We are using a Java-based techniques for security, where our methods are related to self-defending objects (SDO) [8]. Self-defending objects are an extension of the object-oriented programming paradigm, where software objects that are offer sensitive functions or hold sensitive data for protecting those function/data. Another work is by Mont et al., in [14] ,who proposed an approach for access control, using Identity-Based Encryption (IBE) .

we use integrity checks and oblivious hashing (OH) technique in our system in order to strengthen the dependability of our system in case of compromised JRE[19]

.

III. PROPOSED SYSTEM

The Cloud Accountability technique proposed in this work controls automated logging and distributed auditing of relevant access performed by any different entities, carried out at any cloud service provider at any point of time, any system, at any location. It contains major components: logger and log harmonizer.

The logger handles a particular instance or copy of the user's data and is responsible for logging access to that instance or copy. It is the component which is strongly coupled (either single or multiple data items) with the user data, so that it copied whenever the data are copied and downloaded when the data are accessed.. The responsibility of logger includes automatically providing logging access to data items that it contains, ensure that access and usage control policies associated with the data are honored ,encrypting the log record using the public key of the content owner(here we use AES algorithm), and periodically sending them to the log harmonizer. For example, a data owner can specify that user X is only allowed to view but not to write into the data. The logger will control the data access even after it is downloaded by user X.

The log harmonizer is the central component which allows the user access to the log files. The logger sends log files to logger harmonizer. The responsibility of logger harmonizer is auditing. Being the trusted component, the log harmonizer generates the master key. It holds on to the decryption key for the IBE key pair, as it is responsible for decrypting the logs. If the path between log harmonizer and the client is not trusted one, a decryption can be carried out between them wherein, the harmonizer sends the key to the client in a secure key exchange.

It supports two auditing strategies: push and pull. The push strategy is an automated approach where the log file is pushed back to the data owner periodically in an automated fashion. The pull mode works in an on-demand way. Here the log file obtained by the data owner as often as required. In case, if multiple users log for the same set of data items, the log harmonizer will merge the log records from them before sending back to the data owner. The log harmonizer is responsible for handling log file corruption. Additionally, the log harmonizer itself can carry out logging in addition to auditing. To improve the performance, separation of logging and auditing functions are advised. Both the logger and the log harmonizer are implemented as lightweight and portable JAR files.

A. Data Flow

At the beginning, users are to create their own pair of public and private keys based on AES algorithm. Main factor that we are select AES is its speed. Using the generated key, the user encrypt the data , whether and how the cloud servers and other data stake holders are authorized to access the content is abided by the access control rules specified by the data owner. Both key and the access control rules are then send to the cloud service provider that he subscribes to. CSP creates a logger component which is a JAR file, to store its data items. The JAR file contains a set of simple access control rules provided by the data owner. A digital signature is also put by the CSP in the JAR file for JAR authentication .We employ SAML-based authentication when the user requests the access. A trusted identity provider is employed to issue certificates verifying the user's identity based on his username.

Once the authentication is done, the service provider (or the user) will be allowed to access the data enclosed in the JAR. JAR is responsible for providing usage control associated with the logging, or will provide only logging functionality which depends on the configuration settings defined at the time of creation. At the time of logging, each time there is an access to the data; the JAR will automatically generate a log record, encrypt it using the public key provided by the data owner, and store it along with the data. This prevents unauthorized changes to the file by intruders. In addition, to handle possible log file corruption some

error correction information will be sent to the log harmonizer. To guarantee trustworthiness of the logs, each record is signed by the entity which accesses the content. Further, to quickly detect possible errors or missing records, individual records are hashed together to create a chain structure. The encrypted log files can later be decrypted and their integrity is verified. They can be accessed by the data owner or any authorized stakeholders at any time for auditing purposes with the help of the log harmonizer .Our proposed framework prevents attacks like detecting illegal copies of users' data. With only encryption, their logging mechanisms are neither distributed nor automatic. They require the data to stay within the boundaries of the centralized system for the logging to be possible, though it is not suitable in the cloud. Here the logging mechanism also moves with data, thus we will be able to control data at any location. The log file created is attached with the JAR file. And once it is connected to the network it send the log file to data owner through centre component.

B. *The Logger Structure*

We drag the programmable capability of JARs to conduct automated logging. A logger component is a nested JAR file which stores a user's data items and corresponding log files. Our proposed JAR file consists of one outer JAR and one or more inner JARs.

The responsibility of outer JAR is handling the authentication of entities which want to access the data stored in the JAR file. In our context, the data owners may be unaware of the exact CSPs that are going to handle the data. Hence, authentication specified according to the functionality of server, not by the server's URL or identity. Outer JAR is also responsible for selecting the correct inner JAR as per the identity of the entity who requests the data.

Each inner JAR contains the encrypted data , class files to facilitate display enclosed data in a suitable format and retrieval of log files , a log file for each encrypted item. We support two options:

- PureLog :The main task of PureLog is to record every access to the data. This log files are used for auditing purpose.
- AccessLog: Performs two functions: enforcing and access control logging actions. If an access request is denied, ie, no connection to network or unauthorized access, the JAR will record the time when the request is made. If the access requested is granted, the JAR will additionally record the access information along with the duration for which the access is allowed.

Above explained two kinds of logging modules allow the data owner to enforce certain access conditions either reactively (in case of PureLogs) or proactively (in case of AccessLogs). The inner JAR contains class file which corresponds with the log harmonizer, another class file for writing the log records, another for the encrypted data, a fourth class file for displaying or downloading the data , and the public key of the AES key pair that is necessary for encrypting the log records. The outer JAR may contain one or more inner JARs. One class file for authenticating the servers or the users, another for finding the correct inner JAR, one class file for JAR authentication, a fourth class file which checks the JVM's validity using oblivious hashing. Also, a class file is used for managing the GUI for user authentication.

C. *Log Record Generation*

Log records are generated by the logger component. Logging occurs when any access to the data in the JAR, and nlog entries are appended sequentially, in order of creation LR ¼ hr1; . . . ; rki. Each record ri is encrypted individually and appended to the log file. Generally, a log record takes the following form:

r_i = <ID, Act, T, Loc, h((ID, Act, T, Loc)| r_i -1| . . .| 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 corresponds h((ID, Act, T, Loc) |r_i - 1| . . .| r1) to the checksum of the records preceding the newly

inserted record, concatenated with the main content of the record itself (we use I to denote concatenation). Collision-free hash function used to compute the checksum. The component sig denotes the signature of the record created by the server. If same logger component handle more than one file, then an additional ObjIDfield is added to each record. To ensure the correctness of the log records, we verify the locations, access time, as well as actions. In particular, to avoid suppression of the correct time by a malicious entity we use Network Time Protocol (NTP). The location of the cloud service provider can be track using IP address. The JAR can perform an IP lookup to locate CSP. More advanced techniques for determining location can also be used. Similarly, if a trusted time stamp management infrastructure can be used to record the time stamp in the accountability log.

IV SECURITY ANALYSIS

Here we analyze possible attacks to our framework. Assume that attackers may have sufficient Java programming skills to disassemble a JAR file.

A Copying Attack

One of the most intuitive attacks possible in cloud is that attacker copies the entire JAR file. It is possible from the network or from the end users (cloud users) system. But it is not possible in our system to copy the JAR without notice by data owner because the JAR file is required to send log records to log harmonizer periodically. Log harmonizer checks the log IP address. If the log component not send log record (not connect to network) then the access is denied.

B Disassembling Attack

Attacker also tries to extract the log files and data enclosed in the JAR files by disassembling the JAR file. But here we have strength of cryptographic schemes. The log files are encrypted using public key algorithm whose private key stored in log harmonizer. Even if the attackers extract log file from JAR they cannot modify the log record because integrity checks, added to log record at the time of creation, revealing error at the time of verification. Furthermore attackers will not able to attach fake record to log file all log records are hashed together.

Attackers also try to extract the data attached in the JAR. This type of attach is prevented by sealing technique offered by java. Seale is a property of java which creates a signature that does not allow the data or code inside JAR files to change. Here also the data is in encrypted form which makes more security.

C.** Pharming attack:**

Pharming attack is one of the intuitive attacks in network where attackers redirect the website into other bogus site. In the case of cloud redirect to some malicious CSPs. But in our proposed system we use openSSL certificate to verify the CSP.

***D.** Gifar-Based Attack*

Gifar–based attack combining images or any other file types with Jar files. The modified file is used to carry the payloads of various attacks. These JAR file is send to our system by pretending from malicious server. Since we use JAR authentication JAR file is verified first.

V. CONCLUSION AND FUTURE WORKS

We proposed an approach for cloud data accountability by using the object-oriented concept for automatically logging any access to the data in the cloud together with an auditing mechanism. 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. So the data owner can control his data at any location. It is also determine the unauthorized access to the data by malicious entity. We also provide JAR authentication to provide security to cloud users.

In future we are plan to extend the above concept to other network like wireless sensor network for providing security to the data.

REFERENCES

[1] P. Ammann and S. Jajodia, "Distributed Timestamp Generation in Planar Lattice Networks," ACM Trans. Computer Systems, vol. 11,pp. 205-225, Aug. 1993.

[2] Smitha Sundareswaran, Anna C. Squicciarini, Member, IEEE, and Dan Lin *"Ensuring Distributed Accountability for Data Sharing in the Cloud" Proc* IEEE transactions on dependable and secure computing vol.9 no.4 year 2012

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

[4] OASIS Security Services Technical Committee, "Security Assertion Markup Language (saml) 2.0," http://www.oasis-open.org/ committees/tc home.php?wg abbrev=security, 2012.

[5] 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, pp. 187-201, 2005.

[6] B. Crispo and G. Ruffo, "Reasoning about Accountability within Delegation," Proc. Third Int'l Conf. Information and Comm. Security (ICICS), pp. 251-260, 2001.

[7] J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," Computer, vol. 34, no. 8, pp. 57-66, Aug. 2001.

[8] J.W. Holford, W.J. Caelli, and A.W. Rhodes, "Using Self-Defending Objects to Develop Security Aware Applications in Java," Proc. 27th Australasian Conf. Computer Science, vol. 26,
pp. 341-349, 2004.
.

[9] 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.

[10] R. Kailar, "Accountability in Electronic Commerce Protocols," IEEE Trans. Software Eng., vol. 22, no. 5, pp. 313-328, May 1996.

[11] 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.

[12] J.H. Lin, R.L. Geiger, R.R. Smith, A.W. Chan, and S. Wanchoo, Method for Authenticating a Java Archive (jar) for Portable Devices, US Patent 6,766,353, July 2004.

[13] 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.

[14] M.C. Mont, S. Pearson, and P. Bramhall, "Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforceable Tracing Services," Proc. Int'l Workshop Database and Expert Systems Applications (DEXA), pp. 377-382, 2003.

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

[16] S. Pearson, Y. Shen, and M. Mowbray, "A Privacy Manager for Cloud Computing," Proc. Int'l Conf. Cloud Computing (CloudCom), pp. 90-106, 2009.

[17] NTP: The Network Time Protocol, http://www.ntp.org/, 2012.

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

[19] Y. Chen et al., "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive," Proc. Int'l Workshop Information Hiding, F. Petitcolas, ed., pp. 400-414, 2003.