**CLASS: Cloud Log Assuring Soundness and Secrecy Scheme for Cloud Forensics**

**Abstract**

User activity logs can be a valuable source of information in cloud forensic investigations; hence, ensuring the reliability and integrity of such logs is crucial. Most existing solutions for secure logging are designed for conventional systems rather than the complexity of a cloud environment. In this paper, we propose the Cloud Log Assuring Soundness and Secrecy (CLASS) process as an alternative scheme for the securing of logs in a cloud environment. In CLASS, logs are encrypted using the individual user’s public key so that only the user is able to decrypt the content. In order to prevent unauthorized modification of the log, we generate proof of past log (PPL) using Rabin’s fingerprint and Bloom filter. Such an approach reduces verification time significantly. Findings from our experiments deploying CLASS in OpenStack demonstrate the utility of CLASS in a real-worldcontext.

**Architecture**

****

**Existing system**

SecLaaS encrypts the log(s) using the investigating agency’s public key and stores the encrypted log(s) in a cloud server. This ensures privacy and confidentiality of the cloud user, unless the particular user is subject to an investigation (e.g. via a court order). To facilitate log integrity, SecLaaS generates proof of past log (PPL) with the log chain and publishes it publicly after each predefined epoch. A trust model was also suggested that stores the PPL in other clouds to minimize the risk of a malicious cloud entity altering the log. However, in SecLaaS, it is difficult to ensure or verify that the CSP is writing the correct information to the log, or that any information pertinent to the investigation is not omitted or modified. Specifically, SecLaaS does not provide the user the ability to verify the accuracy of the log (since the log is encrypted with the agency’s public key).

**Proposed System**

Extending SecLaaS, we propose a secure cloud logging scheme, Cloud Log Assuring Soundness and Secrecy (CLASS), designed to ensure CSP accountability (i.e. writing the correct information to the log) and preserve the user’s privacy. Specifically, we include the capability for the user to verify the accuracy of their log. To do this, the log will be encrypted using the user’s public key (rather than the agency’s public key). To avoid introducing unnecessary delays to the forensic investigation, during user registration with the cloud service, both the CSP and the user will collectively choose a public/private key pair referred to as content concealing key (CC-key) for the user. The corresponding (content concealing) private key will be shared with other CSPs secret sharing schemes. This would allow the private key to be regenerated whenever necessary. We also demonstrate how we can leverage Rabin’s fingerprint and bloom filter in PPL generation to establish log veracity. We then implement CLASS in OpenStack and evaluate its performance.

**Future work**

Normally logs are low-level data and hard for the common user to understand what exactly those logs signify. Thus, we will explore leveraging big data techniques to facilitate user retrieval and visualization of information from log data. Standardization of log format is also an associated research area.

To ease searching, we kept some crucial and sensitive information in plaintext format. This makes them vulnerable to be exposure. Thus, designing secure and efficient searchable encryption would extend this work.

There is also the need for an online credibility system designed to develop trust and credibility of a cloud user so that the CSP can enable stricter auditing policies for low-trust users in comparison to high-trust users.

Designing and implementing a prototype of the proposed scheme in collaboration with a realworld CSP, with the aim of evaluating its utility(e.g. performance and scalability) in a real-world environment.

**Module implementation**

1. **Preservation Of Log &Its Proof**

Parser collects the log from log source. When a log file changes (i.e. new lines append) it triggers the parser to check the change and to start processing new log.Retrieving log from log source, the parser parses the log and gets the necessary information.Our goal is to keep log content secure given a parser that will provide the system a log message in string format, regardless of content. The format of the log is out of the scope of this work.

1. **Accumulator Design**

Bloom filter as a proof of past data possession, which is fails to account for Bloom filter’s inherent potential for false positives. When using a Bloom filter technique, there is a trade-off between the number of false positives and the size of the filter. To mitigate this problem, a cryptographic one-way accumulator could beused. However, this requires significant computational overhead. In SecLaaS, they used their own data structure Bloom Tree that reduced the number of false positive incidents but requires an increased number of instances of logs and significant computational resources at verification time. This is true regardless of the number of entries being verified. In addition, it still remains vulnerable to false positives (albeit reduced).

1. **Verification**

Only a verification process that establishes authenticity will be able to determine the presence of log contamination. There are two types of verifications in our approach. First is verification where the user checks if the CSP is writing correct log entries or not. Second is verification by any party: user, investigator, law enforcement authority (LEA) or court of law to verify PPL to check for log modification. In both cases, the CSP can provide a small utility verification software or the user/investigator can buy it from individual software vendor (ISV) to verify.

1. **Secret Key Sharing**

We propose, in CLASS, to encrypt the log with the user’s private key (CC-key). In recognition that this might lead to permanent loss of log data (even for investigative entities), as the private key of a user’s CC-key is known only to the user, we propose to share individual user's private key according to Shamir’s or Blackley’s secret key sharing strategy among multiple CSPs. This sharing scheme requires standardization. We can build sharing clouds for such a purpose when a user subscribes to a cloud service. The CSP and user together choose a pair of public/private key that is called the content concealing key (or CC-key) because it is used to hide user’s log content.

**Algorithm**

1. **CLASS algorithms:** can be categorized into two major groups: One for Log Preservation and one for Proof Accumulation. The Log Preservation algorithm can take log entries individually or in a batch and performs processing prior to storage in a log database. This algorithm encrypts for secrecy and generates hash digest for consistency. The Proof Accumulator algorithm performs daily processing of all log entries corresponding to an IP address to prepare and publish proof of past log (PPL).
2. **Shamir’s secret sharing algorithm:**Small information will be extracted from private key following Shamir’s secret sharing algorithm and each portion will be shared to one CSP. After getting secret portions of a particular user, the host cloud can reconstruct the private key to decrypt the log of that user using Shamir’s secret sharing algorithm.

**Conclusion**

In this paper, we proposed a secure logging scheme (CLASS) for cloud computing with features that facilitate thepreservation of user privacyand that mitigate the damaging effects ofcollusion amongother parties. CLASS preserves the privacy of cloud usersby encryptingcloud logswith a public key of therespective user while also facilitating log retrieval in the event of an investigation. Moreover, it ensures accountability of the cloud server by allowing the user to identify any log modification. This has the additional effect of preventing auser fromrepudiating entries in his own log once the log has had its PPL established. Our implementation on OpenStack demonstrates the feasibility andpracticality oftheproposed scheme. The experimental results show an improvement in efficiency thanks to the features of theCLASSscheme, particularlyin verification phase.

**SYSTEM REQUIREMENTS**

➢ **H/W System Configuration:-**

➢ Processor - Pentium –IV or Later Version

➢ RAM - 4 GB (min)

➢ Hard Disk - 40 GB

➢ Key Board - Standard Windows Keyboard

➢ Mouse - Two or Three Button Mouse

➢ Monitor - SVGA

**Software Requirements:**

* Operating System - Windows XP or Later Version
* Coding Language - Java/J2EE(JSP,Servlet)
* Front End - J2EE
* Back End - MySQL