Ensuring Audit Log Accountability through Hash Based Techniques

Rashmita Jena, M. Aparna, Chinmaya Sahu, Rajeev Ranjan, and Rajesh Atmakuri

***Abstract*—Audit logs are now considered good practice and a standard approach for business systems. The integrity of the auditing records themselves is critical. By simply storing all the interactions in a separate audit log does not guarantee the integrity of the log. Data tampering can be done through unauthorized access and in some cases through authorized users. Results of such action can be unpleasant for business and their clients. Therefore, a demand for audit log security is needed more than ever. This paper describes a mechanism based on cryptographic hash functions and trusted timestamping that prevents an outsider or inside intruder from silently corrupting the audit log. In addition it is shown that the proposed mechanism can be realized in database systems with little overhead and that the hash based techniques and trusted timestamping can be used efficiently and correctly to determine if the audit log has been compromised.**

***Index Terms*—Audit logs, hashing, database.**

I. Introduction and Motivation

The objective of data security can be divided into three separate, but interrelated, areas as secrecy, integrity and availability. It is important to understand that the threat posed by a corrupt authorized user is quite different in the context of correctness of the data as compared to secrecy. Recent experience has shown that data tampering can be done with unauthorized access and in some cases through authorized access. Corrupt authorized users can leak the internal secrets by using the computer to access confidential information, and then passing on this information to any other destination by some non-computer means of communication (e.g., a telephone call). It is impossible for the computer systems to know whether or not first step was followed by second step. There is no choice other than assuming that insiders are honest. There is only one direction to provide valid forensic analysis of database transaction which requires an audit log of all aspects of information system. Based on that, database forensics process can reconstruct what has really happened.

Let us first discuss a few examples to see what the users concerns might be and why one might want to protect users from malicious audit log. Consider a sales company, wherein the intruder is an insider rather than someone hacking in from the outside, could be any employee at a large company who is trying to meet his sales requirements for a fiscal year. He might attempt to change the transaction dates to make it appear that they had transpired within the previous fiscal year when, in reality, they had not. Consider a school database where a student who receives a "F" in one of his subjects, in which he needs at least a "B", could be highly tempted to try to dishonestly change his grade to a "B" in the database. This would be an example of a student who would have to hack into the system, unless of course the student somehow had access to the database containing the grade.

The above discussed examples provide just a few of the reasons why someone might want to tamper with a database. These fraudulent acts can be punishable by law and result in severe consequences if the intruder is caught. These examples also give a clear message that when users have full access to audit logs in performing auditing of interactions with the data (modification, exposure) as well as of the base data itself, it is difficult to prove the integrity of the audit logs.

The requirements of recent regulations, to ensure trustworthy long-term retention of their routine business documents, have led to a huge market for compliance storage servers, which ensure that data are not shredded or altered before the end of their mandatory retention period. Meanwhile, there are many commercially available tools to assist forensics but these tools are not applicable to tamper detection of database audit logs by intruders having full access over audit logs. With the recent development of electronic commerce, time stamping is now widely recognized as an important technique used to ensure the integrity of digital data for a long time period. In real-world applications, sensitive information is kept in log files on an untrusted machine. In the event of an intrusion into this machine, we would like to guarantee that no corrupt log file goes undetected. In this paper we present a mechanism based on digital timestamps and hash functions that prevent an outsider or inside intruder from silently corrupting the audit log. In addition to the theory, we present performance analysis and results with an implementation on a high-performance storage engine. Finally we show that the overhead for using hash based techniques and digital timestamps for audit log security is low and that the hash based techniques can be used to determine if the audit log has been compromised.

II. Tamper Detection Methodology

The security of our audit log file comes from fact that each log entry contains an element in a hash chain that serve to authenticate the values of all previous log entries. It is this

Manuscript received April 16, 2012; revised June 2, 2012.

Rashmita Jena, M. Aparna, Chinmaya Sahu, and Rajeev Ranjan are with the Department of Computer Science National Institute of Science and Technology Palur Hills, Berhampur Orissa, India (e-mail:rashmitaj@nist.edu; maparna@nist.edu; chinmayas@nist.edu; rajeev@nist.edu).

Rajesh Atmakuri is with Wipro InfoTech Electronic city Bangalore Karnataka, India (e-mail: rajesh.atmakuri@wipro.com).

DOI: 10.7763/IJFCC.2012.V1.88

value that is actually authenticated, which makes it possible to verify all previous log entries by authenticating a single hash value. We have added two special columns for storing the Hash Code and the Chain_ID (Most Recent Digital Timestamp Chain ID).We have also added another table to store the chain of digital timestamps generated by the Timestamping Authority or the TSA. Each entry in the log file stores the most recent Chain_ID generated in the timestamp chain table. Since the hashcode is based upon previous tuples it is important to hash the tuples in the same order during validation. For this purpose we have added a tuple sequence number *Sn* which is incremented within a chain.

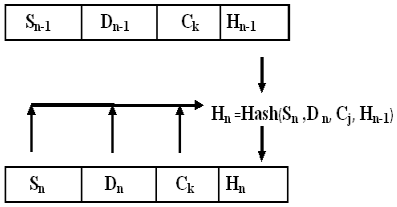

Fig. 2.1. Hash chain.

The hashcode Hn for nth log entry as in Fig. 2.1 is the hash of *Sn* which is the tuple sequence number within the particular Chain_ID, Dn which is the Data to be entered in the nth log entry of the audit file where hash is the one-way hash, using an algorithm such as SHA-1 [NIST 1993] or RIPE-MD [Dobbertin et al. 1996], of X, Ck is the most recent Chain_ID generated in the timestamp chain and the hashcode Hn-1of the n-1th entry in the log. Periodically (may be once a day) we can suspend the transaction execution and the most recent hashcode in the audit log table combined with the recent Chain_ID in the timestamp chain table hashed together and sent to the TSA requesting for a digital timestamp as shown in Fig. 2.2. The TSA concatenates a generated timestamp to this hash value and calculates the hash *Tj* of this concatenation. This hash *Tj* is in turn digitally signed with the private key of the TSA to compute *Zj*. This signed hash *Zj* and the generated timestamp *Yj* is stored with the TSA.The hash *Tj* and the timestamp *Yj* is added as a new entry in the timestamp chain table (Fig. 2). Since the original data cannot be calculated from the hash (because the hash function is a one way function), the TSA never gets to see the original data, which allows the use of this method for confidential data.

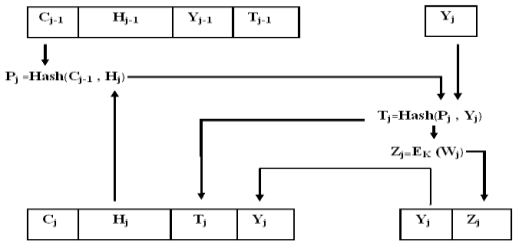

Fig. 2.2. Generating digital timestamps.

Even change in a single byte will not generate the same hash codes. This makes the base of the correctness of the proposed mechanism. Any change or a modification in any given rows will result in a mismatch of the hash value and therefore can be detected. The interwoven hashing mechanism will ensure that if one particular row is deleted or modified from the Audit table, the detection algorithm can find a mismatch by the existence of other rows preceding and following the deleted or modified row. If anybody tries to modify the data or tries to backdate or postdate the transaction timestamps, the hash value which is calculated from the audit data and transaction timestamps will not match. If any insider has full access to the base data and audit log, he can simply calculate the hash values and restore the database with his changes but during the forensic analysis this can be detected through the digital timestamps which will not match with the TSA. Hence it is impossible for any insider to hide the corrupted log file.

III. Performance

We now turn to a more detailed, empirical analysis of our implementation. We studied the performance of the auditing system and the various database parameters impact on the auditing system performance.

A. Experiment Design

We simulated a university database scenario. The database was populated with tuples inserted in random order. Each tuple represented the attendance of students in different subjects. For the normal experiments the tuple size was 250 bytes and each transaction carried 4 tuples. The computations were done on a 2.66GHz Intel Core2Duo running Red Hat Enterprise Linux version 5.1 with Oracle DBMS version 11g. In the initial approach the third party timestamping service response time (from the DBMS sending the timestamp request until receiving the response back from the third party service, a quite conservative estimate) was less than a second since the we had used local timestamping service. The timestamping service was called initially every five seconds and then on per day basis.

B. Run Time Overhead

Fig. 3.1 depicts that the hashing overhead was around 9% on an average in all the experiments that were run. This is a small price to pay for the protection of highly critical data. The overhead of hashing was analyzed by only hashing the tuples values and timestamp for each tuple individually. The tuple size was fixed to 250 bytes and each transaction carried 4 tuples for the experiments carried out to analyze the hashing overhead.

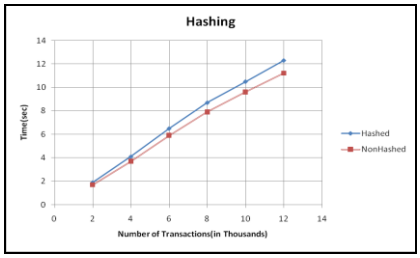

Fig. 3.1. Hashing overhead.

When timestamping service was done every five seconds it reflects the worst case, because in the real world application, the timestamping will not be done that frequently. It would usually be around one request per day, which imposes much less timestamping overhead than what we did here. In the initial approach five seconds was chosen here just to

accommodate the experiments, which run for a relatively short amount of time (much less than a day). Local digital timestamping service was implemented, rather than using one of the commercial services. The timestamping service response time does not affect the auditing system performance. The result was same as shown in Fig. 3.1. All the experiments were based on SQL triggers to store the audit logs for DML operations in transactions. These triggers which add as overhead in the proposed mechanism since now a days audit trails are implemented in many commercially available DBMS such as Oracle which takes much more less time to store audit logs. If the hashing technique is added to these then the overhead will be less than 10%.

IV. Related Work

There has been related work in the field of audit log security and forensic analysis. We address each in turn. A method of forensic tamper detection in SQL Server is presented by Amit Basu [6]. Their approach is based on creating an interwoven chain of hash values to determine if a particular audit log row is modified, inserted, or deleted. This approach with its advantages, suffers from the use of non-cryptographic hash functions, and the limited forensic strength of the detection algorithm. Their approach is also not safe from the intrusion of insiders. To overcome this we have introduced the trusted Timestamping service. Note that since we send only hash values to the trusted third party service, no private data that is revealed to that external service. Peha [12] uses, as we do, one-way hash functions and a trusted timestamps to hash and store every transaction. Our approach differs from their approach in that we made no assumptions about the DBMS, or even the hardware it executes on, remaining in the trusted computing base following an intrusion. Peha [12] simply batches transactions together by hashing all the data in each and every transaction, which will undoubtedly result in increased time complexity. Schneier and Kelsey [1] address audit logs that are used for later forensic investigations into detected intrusions. Their requirement differs considerably from ours. In particular, their render approach the log entries impossible for any intruder to read. They use a hash linking in a similar way to our algorithm but encryption is done for all the log entries. They do not consider the efficiency issues, in situations where an online transactional database is being logged which is critical in our case.

V. Summary and Future Work

Motivated by audit log requirements, we have presented a new approach to transaction processing systems that can assist in detecting tampering effectively and efficiently. We based our approach on existing cryptographic techniques such as strong cryptographic hashing, partial result authentication codes, and offsite digital timestamping services. Our contributions are as follows:

We showed how hash based techniques can be used as the basis for forensic analysis in audit logs, transparent to the application.

We showed how a trusted timestamping service can be used to prevent any insider or outside intruder at the company site or Timestamping authority site from corrupting the audit log.

To minimize the expense of interacting with Timestamping Authority site, linked hashing of log records was introduced, by means of partial result authentication codes.

We showed that the Timestamping Authority service response time minimally impacts system performance.

We developed an implementation within the high performance data storage engine, and showed through experiments that the hashing overhead never exceeds 15% in all the computations and if the proposed mechanism is added in commercial DBMS then the overhead would be less than 10%.

We've focused in this paper on certifying the integrity of an audit log in a definitive fashion. If an audit log has been tampered with, then database forensic analysis algorithms would be needed to detect the tampering and should be able to determine the who, when, what and where components of the tampering. It is also possible for the DBMS and applications to log activities through additional data stored in the database to assist in analyzing tampering. Of course, such data should be stored in audited tables. The appropriate third party timestamp service request granularity (an hour, a day?) should be investigated. Finally, we want to produce and evaluate mechanisms that leverage tamper detection of audit logs to produce tamper resistant audit logs, which cannot be corrupted, yet are still accessible to the application.

References

[1] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics," ACM *Transactions on Information and System Security* vol. 2, no. 2, pp. 159-196, May, 1999.

[2] K. E. Pavlou and R. T. Snodgrass, "The Tiled Bitmap Forensic Analysis Algorithm," *IEEE Transactions on Knowledge and Data Engineering,* vol. *22*, no. 4, pp. 590-601, April, 2010.

[3] M. S. Olivier, "On metadata context in Database Forensics," *Digital Investigation* vol. 5, Issues 3- 4, pp. 115-123, March, 2009.

[4] K. Pavlou and R. T. Snodgrass, "Forensic Analysis of Database Tampering, International Conference on Management of Data," *Proceedings of the ACM SIGMOD International Conference on Management of data*, SESSION: Authentication, pp. 109 – 120, 2006.

[5] M. Malmgren, (2009) *An Infrastructure for Database Tamper Detection and Forensic Analysis*, honors thesis, Univ. of Arizona, [Online]. Available: http://www.cs.arizona.edu/projects/tau/tbdb/MelindaMalmgrenThesis.pdf.

[6] Article by A. Basu. (2006, November) Forensic Tamper Detection in SQL Server, [Online]. Available: http://www.sqlsecurity.com/images/tamper/tamperdetection.html.

[7] SQL Server Forensic Analysis by Kevvie Fowler *SQL Server Forensic Analysis*, ISBN: 9780321533203.

[8] R. S. Sandhu and S. Jajodia, "Data and database security and controls," *Handbook of Information Security Management, Auerbach Publishers*, 1993, pp. 481-499

[9] Article by David Litchfield (2011, August), [Online]. Available: www.darkreading.com/databasesecurity/167901020/ security/attacksbreaches/231300307/database-forensics-still-in-dark-ages.html

[10] K. Pavlou, (2011) *Database Forensics in the Service of Information Accountability*, [Online]. Available: http://www.cs.arizona.edu/projects/tau/dragoon

[11] R. T. Mercuri, "On Auditing Audit Trails," *CACM* vol. 46, no. 1, pp. 17-20, January, 2003.

[12] J. M. Peha, "Electronic commerce with verifiable audit trails," in *Proceedings of ISOC, 1999*. [Online]. Available: http://www.isoc.org/isoc/conferences/inet/99/proceedings/1h/1h_1.htm , viewed on March 26, 2003.