Detecting Insider Attacks on Databases using Blockchains

Shubham Sharma, Rahul Gupta, Shubham Sahai Srivastava and Sandeep K. Shukla

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur, India
smsharma@iitk.ac.in, grahul@iitk.ac.in, ssahai@cse.iitk.ac.in, sandeeps@cse.iitk.ac.in

Abstract. Applications relying on centralized databases are often vulnerable to insider attacks. Any user with administrative privileges to the database system or the hosting server, is capable of modifying the database entries. Furthermore, such a user might modify the corresponding log entries, making it extremely difficult to detect such an attack. Attribution of the attack to privileged users would also be challenging. In this paper, we propose a solution to this problem using the tamper-resistance property of blockchains. We have implemented and tested our protocol, using multichain [2], on a web-based online academic grading database application with a goal to make it robust to such an insider attack.

Keywords: Blockchain, Centralized Database Systems, Immutability, Insider Attack

1 Introduction

Database systems form the core of large number of applications we see today, be it web or mobile. With an ever increasing reliance on databases, protecting them from unwarranted manipulation is of utmost importance. It is a common practice to secure database servers behind layers of firewall, and define access control policies controlling the read/write access for various users of the system. The application logic not only ensures that these policies are not violated, but is also responsible for protecting the databases from SQL-injection attacks. Nevertheless, insider attacks remain a persistent threat to such systems.

Currently, access control in most of the centralized database systems and applications is based on organizational policies allowing privileged (read/write) access on data to several users. As a result, if a privileged user manipulates entries in the database, and modifies the corresponding log entries, it would be difficult to attribute any changes to such users. In extreme situations, it might be difficult to even detect such an attack. This is an example of an "insider threat" which according to 2014 U.S. State of Cybercrime Survey, constitute almost 28% of all attacks [1]. Such non-repudiability is not desirable for applications.

In this paper, we propose a technique to detect such attacks. In our approach, we exploit the immutability of blockchains, and store some metadata corresponding to each entry of the database on the blockchain. The metadata would have a very small storage overhead, but will help us in verifying the database entries when accessed.

The remainder of this paper is organized as follows: Section 2 provides a brief overview of the problem statement. Our proposed solution is presented in Section 3 and we discuss the implementation details in Section 4, where we talk about the implementation of our approach in the grade management system which is robust to the unauthorized modification of the grades. We conclude the paper in section 5.

2 Problem Statement : Security issues in centralized database systems

In this section, we illustrate the problem with the example of an online academic grade maintenance database application. This database has web front-end which allows legitimate users to access the various views of the database, submit grades and view grades for various courses offered at an institute. The user access is controlled with usual username/password credentials which can be enhanced with 2-factor authentication. However, there are IT administrators who have access to this database with administrative privileges, and if they manipulate data directly on the database, and remove all database and system logs with the help of their administrative privileges, the students might find their grades changed, but will not be able to prove that such changes were not legitimate, nor will the faculty would be able to attribute the changes to any specific administrators.

We need a protocol that would prove that changes have taken place without any doubt, and also be able to recover the original grades in case no other record had been kept. In order to do this, we need a tamper resistant logging mechanism which is not under the control of the administrators, which is publicly verifiable by each stake holder using cryptographic techniques, and which should not compromise the privacy of users by revealing the individual's grades to users other than the individual owning the grade. We solve this problem with block chain based distributed tamper resistant ledger.

Adversary Model : Any modifications in the database can be triggered via application by a user or by the database command-line console directly. For our work here, we consider any database update triggered form the application as legitimate update. However, we are concerned with the updates that bypass the authentication and logic of the application and are executed directly from the command-line console. We treat such actions as insider attacks, and assume only users having privileged access to the system will be able to execute such actions.

We also assume that the application logic handles the authentication of users and is capable of bypassing attacks like SQL-Injection. However, if a user is compromised and any update is made using his or her account via the application,

our technique will attribute these changes to that user and the attacker will not be able to modify the logs to prevent detection. In case the modifications are malicious, this attribution can be further used to investigate the scenario, and decide whether the user has been compromised or has turned malicious.

3 Proposed Solution

We propose a blockchain based solution, which we have implemented using multichain [2]. It is assumed that there exists a Public Key Infrastructure (PKI), from which public key of a user can be retrieved, using some unique identifier like email-id, employee-id etc. In our protocol Blockchain is used to store the state of database. Figure 1 presents the overview of our proposed system. Section 3.1 describes the modifications required in the SQL database in order to implement our protocol. In section 3.2 we will focus on the INSERT queries. We consider the SELECT queries in Section 3.3

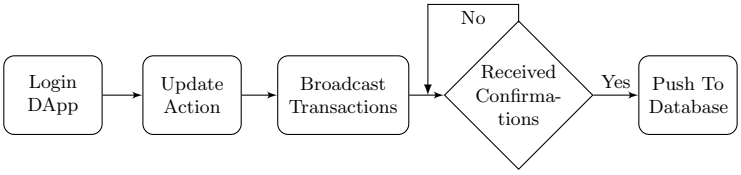

Fig. 1. Overview of the proposed protocol

3.1 Modifications required in Database Schema :

Consider a database table with k columns. Then in order to incorporate our protocol, we will need 2 extra columns. The modified database table is presented in Figure 2.

We will use these 2 columns to store the following metadata:

1. **Transaction.** Transaction id (txnid) of the transaction in first blockchain, corresponding to this database entry, is stored here.
2. **Identifier.** Unique identifier of the user who issued the INSERT/UPDATE query for this database entry, is stored here.

3.2 INSERT query

1. The user logs in the application using his credentials and requests an update action, which corresponds to INSERT query in the database.
2. The application then :
   - extracts all the fields of the INSERT query in form of tuple,

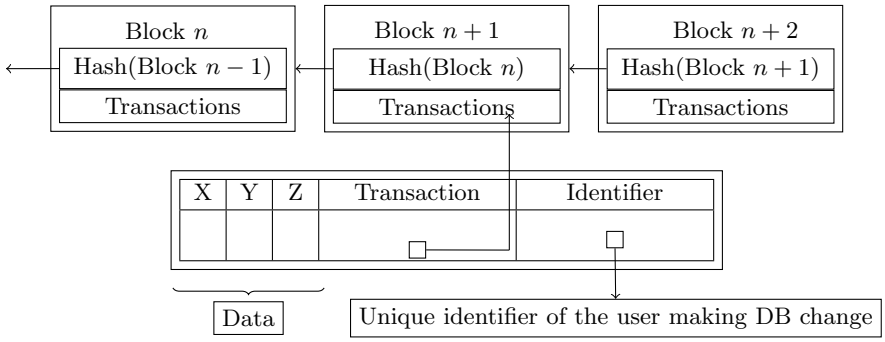

Fig. 2. Database schema modifications

- digitally signs the tuple with the user's private key,
- hashes the generated tuple, and,
- broadcasts the above hash value as a transaction to the first blockchain network.

The application stores the database query and the hashed value of the transaction in its internal log.

3. The application waits for the transaction to be mined in a block in the first blockchain and get a predetermined number of confirmations.
4. When the transaction has received the required number of confirmations,
   - the application retrieves the corresponding database query from its internal log,
   - modifies it to include the transaction id (txnid) of the transaction in blockchain and the identifier of the user generating the query, and
   - sends it to the database for execution of INSERT query.

3.3 Detecting unauthorized modifications in the database :

The application performs the following check after it receives some rows as a response from the database for a SELECT query :

1. It takes the value of all the columns of each row (except the last two i.e. the first k columns in Figure 2), and forms a tuple.
2. It queries the blockchain for the transaction using the pointer present in the Transaction field ($(k+1)^{th}$ column). As a result it receives the corresponding transaction.
3. As the transaction is digitally signed by the user, it then checks the digital signature using the public key retrieved from an existing PKI against the unique identifier present in the $(k+2)^{th}$ column.
4. If the signature matches, it verifies the transaction value, which is nothing but the hash value of the tuple generated in Step 1.

In case, an error is detected in Step 3 or Step 4 above, it would mean the database entry was modified in an unauthorized way.

4 Implementation

We have implemented and tested our protocol for the following scenario. Consider an academic institution where we have to distribute grades to the students. There are several courses hosted by institution in each semester. Each course is taught by one or more professors and multiple students are enrolled.

4.1 Implementaion Details

Relational Database : To store the enrolment data and grades we can use any relational database. We have used MySQL for this purpose. The tables in the database correspond to entities : professors, students and courses. The relationship between them is marked by enrolment table which stores information about students enrolled in multiple courses taken by a certain professor. The enrolment table stores details about grades of the students which is to be protected in our scenario. So, as described in previous section, we have modified enrolment table to include columns for transaction id and identifier of the professor, as in our case, only the professor is allowed to make grade changes.

Restricting access of specific parts of the table to different entities is done using MySQL views. This is done to prevent any unauthorized access to the information stored in the database.

Web Application : We needed a web app to enable interaction with the users. We have used Django for implementing the DApp. A local Django web server runs for each student and professor which connects to the MySQL database with provided username and password to fetch and populate the data. Each student is able to see only his own grades, while professors have the ability to change the grades for any student.

Blockchain : For a distributed blockchain, we are using Multichain Streams which provide a natural abstraction for permissioned blockchain use cases, and focus on general data retrieval, timestamping and archiving, rather than the transfer of assets between participants. We are calling our stream : log-stream. We expect each participant to install multichain and then use our script to connect to a central multichain node. Each participant is then assigned a burn address for the chain which can be used to assign permissions.

The permissions for the stream are distributed according to professors' burn addresses as retrieved from PKI which allows them to append transactions and mine blocks while allowing everyone to view the blockchain, which enables the students to verify their grades according to the protocol. The verification is done by the DApp as described in following sections.

4.2 Handling INSERT Query

On the backend, whenever a professor inserts grades for any student (INSERT query), the hash of tuple (Course id, Student id, Professor id, grade) is signed

and the signature is broadcasted on log-stream. The DApp stores the actual query in its internal log. When this transaction gets mined in a block and a certain number of confirmations are received, then the DApp retrieves the actual information about the grade, and push the query to the database after appending the transaction id and identifier of the professor.

4.3 Verification of the Data

Whenever a student views his grades, the server fetches information about the grades from the central database which contains transaction id and identifier as well. The DApp the retrieves the transaction from the log-stream using the transaction id present in the row. The DApp fetches the public key from the PKI using the identifier. The signature of the transaction is then verified by using this public key. If the signature is verified successfully and the hash of the data matches the hash present in the transaction, then the student is able to see his grades. In case there is a failure in the above steps, an alarm is raised on student's end corresponding to that grade entry and a notification is sent to concerned authorities to look into the unwarranted changes on the database.

5 Conclusion

In this paper we have presented an approach based on blockchains to detect an insider attack or any unauthorized modification of the database entries. The system presented has a low storage overhead, as in addition to the usual data, it also stores a hash value and a public key in each row. It will also involve some latency, as instead of directly modifying the entries of the database, the application waits for some required number of confirmations on the blockchain corresponding to that query. But, at the cost of this small latency and low storage overhead, we were able to detect any modifications on the database, that were not authorized via our DApp.

References

1. 2014 U.S. state of cybercrime survey. URL: https://resources.sei.cmu.edu/asset_files/Presentation/2014_017_001_298322.pdf.
2. Multichain. URL: https://www.multichain.com/.