Overview of Database Query Auditing Techniques

Ms. Vidya Ingle, Prof. Manjusha Deshmukh

***Abstract*— Government regulations and increased awareness of security issues have increased the auditing requirements of information technology systems. The goal of an auditing system is to determine if security and other policies are being violated. It provides the way to detect intrusions into the system, including privileged users and also provide periodic report of system usage and data modifications. Auditing can also be used to detect and recover database systems in case of system failure or human errors. Thus auditing is a key part of the security infrastructure in a database system.**

While commercial database systems provide mechanisms such as triggers that can be used to track and log any changes made to "sensitive" data using UPDATE queries, they are not useful for tracking accesses to sensitive data using complex SQL queries, which is important for many applications given recent laws such as HIPAA.

This paper focuses on a framework for auditing queries and several different notions of suspiciousness for simple SQL queries. It also focuses on notion of SELECT triggers that extends triggers to work for SELECT queries in order to facilitate data auditing.

***Index Terms*— Data auditing, SQL query auditing, SELECT Triggers.**

I. Introduction

Today databases are used to store sensitive information such as health records, employee records and customer data which motivates the need for developing a security infrastructure as part of a DBMS. The security infrastructure of database systems consists of mechanisms such as access control and encryption to prevent security breaches. Despite the availability of preventive mechanisms in modern DBMSs, data breaches are common. One of the most common reasons for data breaches is insider attacks where the insider gets knowledge of sensitive data by running SQL queries and examining their results. Therefore, auditing system has become an important part of the security infrastructure of a database system to monitor various operations during production and can be used to investigate security breaches.

Manuscript received May 15,2014

Ms. Vidya Ingle, Department of Information Technology, Mumbai University/ Pillai's Institute if Information Technology, New Panvel, India, (e-mail: inglevidya@rediffmail.com).

Prof. Manjusha Deshmukh, Department of Computer Engineering, Mumbai University/Pillai's Institute of Information Technology, New Panvel,India, (e-mail: manjudeshmukh@rediffmail.com).

II. Database Auditing

Auditing is the process of examining the past actions to determine whether they were in conformance with official policies. In the context of database systems, auditing queries is the process of going over past queries that have been answered and determining whether these answers could have been pieced together by a user to deduce prohibited information [2][3].Thus need for some sort of an audit mechanism in database management systems is clear. For example, an individual who receives advertisements for a specific kind of illness might suspect that his health-care provider has leaked private information from his medical records to concerned parties. If the privacy policy of provider claims that it never release patient data to external parties, the provider must be able to prove compliance with this policy

III. Database Query Auditing

An auditing framework for SQL queries was introduced in [2]. This framework is useful to check whether any query posed in the past accessed/revealed some confidential data. With this approach, users can create audit expressions to specify data that should not be released illegally. The auditing system as shown in Fig 1 identifies all "suspicious" queries that accessed the sensitive data specified in the audit expression directly or indirectly.

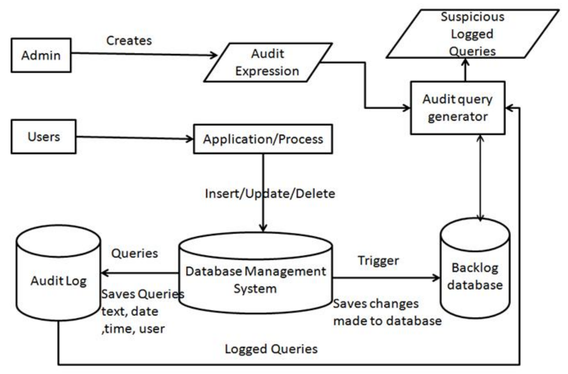

Fig 1 Basic architecture of auditing system

Fig 1 shows the overall architecture of the system. Query Log records every query processed by the database system with information such as the time of execution of query, the user who submitted the query, and the purpose of query. All updates to base tables are captured in backlog tables with the help of triggers so that any past state of the database can be recovered at any point in time. Select/Read queries; do not write any tuple to the backlog database as we cannot write trigger for select queries [1].

IV. AUDITING SQL QUERIES

A. Perfect privacy

In [1, 3] the authors consider the problem of ensuring "perfect privacy". A database system releases various views of its data, does it disclose any information at all about a view that must be kept confidential. For example, a database contains the names and salaries of employees in an organization and we want to keep secret the entire salary column of this database. A query asking for the names of all the employees in the organization would be considered suspicious with respect to the secret view under the "perfect privacy" notion of security even though not a single salary would have been revealed by this query. This is simply because by revealing information about the size of the database, it has revealed some small amount of information about the salary column.

An audit expression very closely resembles a SQL query:

```
 AUDIT audit list
FROM table list
WHERE condition list
```

It essentially recognized the tuples of importance from the cross- product of tables in the FROM clause via predicates in the WHERE clause.

Consider the audit expression:

```
AUDIT Salary
 FROM Employee e
WHERE e.deptNo =10;
```

This expression identifies all queries that accessed the salary column of any employee in department no.10 directly or indirectly. All such queries will be considered suspicious. Now consider the SQL query:

```
SELECT deptNo
FROM employee e
 WHERE e.salary = 10000;
```

Now if any employee whose salary is 10000 works in department no 10, this SQL query will be considered suspicious with respect to the above audit expression because the Salary column of an employee working in department No 10 was accessed while answering the query. On the other hand, this SQL query would not be suspicious with respect to the audit expression given below:

```
AUDIT deptNo
 FROM employee e
 WHERE salary = 20000
```

This is because this audit expression is only interested in checking if the department no. of any employee having salary=20000 was revealed. Note that queries are taken at their face value, assuming away background knowledge.

What it means for a query to be suspicious with respect to an audit expression is formalized as follows [3].

Definition 1 (Candidate Query): A query Q is a candidate query with respect to an audit expression A, if Q accesses all the columns that A specifies in its audit list.

Note that the query should access all the columns of the audit expression to be called a candidate.

Definition 2 (Indispensable Tuple): A tuple t ∈ T is indispensable to a query Q if the presence or absence of t makes a difference to the result of Q.

Definition 3 (Suspicious Query): A candidate query Q is suspicious with respect to an audit expression A, if they share an indispensable tuple.

The idea is that an indispensable tuple for the audit expression would be one of the tuples being subjected to a disclosure evaluation. So if any one of these tuples is also an indispensable tuple for a candidate query in the query log, then that query would have accessed all the columns of the audit list for that tuple and should therefore be considered suspicious.

V. AUDITING A BATCH OF SQL QUERIES

Usually a single query in isolation may not access all the columns of the audit list; but few queries together may outlook sensitive information [2]. For example, the audit expression might be

```
 AUDIT name, Salary
FROM Employee e
WHERE e.deptNo =10
```

Here the association between names and salary of employees working in department No 10 should be kept confidential. Now consider the SQL queries:

```
SELECT deptNo
FROM employee e
```

WHERE e.salary =10000

SELECT e.name
FROM employee e
WHERE e.deptNo =10

Any of these above queries at its own does not reveal the association between name and salary of any individual working in department No 10; however the combination of the queries does disclose something. For example, if there is only one employee in department No 10 and this employee have salary=10000, then these two queries have revealed the name, salary association for that individual.

Thus, there are clever ways in which the results of queries could be combined to reveal secret information. The attackers is very intelligent to knows exactly which tuple in the results of two queries be joined together, i.e. in each query he implicitly also selects the key column [2].

Definition of suspiciousness can be extended to batches of SQL queries called *semantic suspiciousness*

Definition 4 (Semantically Suspicious Query Batch): A batch of queries, Q is said to be semantically suspicious with respect to an audit expression A if there is some subset of queries Q' ⊆ Q such that a tuple t ∈ T is indispensable to both A and every query in Q' and the queries in Q' together access all the columns of the audit list in A [3].

This definition of suspiciousness lends itself to an auditing approach where a query is executed over the database for every query in the query batch.

Note that, the query batch should access *all* the audit list columns in order to be considered for suspiciousness testing. This is because informally, we would ideally like to determine "full disclosure" of a forbidden tuple, i.e., could any combination of queries in the query batch have revealed all the audit list columns of a forbidden tuple to an adversary. This is motivated by findings where attributes such as date-of-birth, gender or zipcode on their own are not selective, however in conjunction can uniquely identify individuals. In such a case, revealing the diseases of all females in a zipcode is likely to be harmless; however revealing this in conjunction with dates-of-birth could cause significant privacy breaches.

Definition 5 (Syntactically Suspicious Query Batch):A query batch is said to be syntactically suspicious with respect to an audit expression if there is some possible instantiation of database tables for which it is semantically suspicious [3].

Since the current database tables form an instantiation, it follows that if a set of queries is semantically suspicious with respect to an audit expression A, then it is also syntactically suspicious.

VI. SELECT TRIGGERS

Traditionally triggers can be used to trace changes to sensitive data and are used in data auditing to enable row level auditing of DDL/DML statements. Using triggers, an administrator is able to keep track of changes made to sensitive column such as salary, disease etc [4].

But mostly data leaks occur by insider attack where insider knows the sensitive information by executing SELECT queries. To get an alert of such data read it is likely to consider extending triggers to SELECT queries for data auditing. Security admin can install a SELECT trigger that executes every time a select query is run and a record can be added in query log or immediate alarm can be set [4].

SELECT triggers also open up the option of real time feedback on access to sensitive information which is essentially important and can enable addressing scenarios such as:

1) Getting an immediate alert when some unauthorized user tries to retrieve sensitive data.
2) Help the administrator to keep eye on the authorised user's actions before security breach happens.

A. SELECT Trigger Implementation

Even though the offline system is still required in the auditing infrastructure, SELECT triggers can serve as an important filter to reduce the number of queries that the offline system must process and therefore improves the overall auditing performance [4].

This framework is prototyped by modifying a commercial database system (Microsoft SQL Server) [4]. The experimental results on queries indicate that SELECT triggers attain a low false positive rate and can scale to a large number of individuals/tuples for a small additional overhead – for instance, we can audit a large number of individuals (around a million customers in the TPC-H benchmark) at an additional overhead of 2% [4].

B. Auditing system with select triggers

There can be situations when offline auditing is not appropriate. For example, Instead of waiting for months, weeks or even days for analyzing the query log, an administrator may want to know immediately when sensitive data are accessed [4].

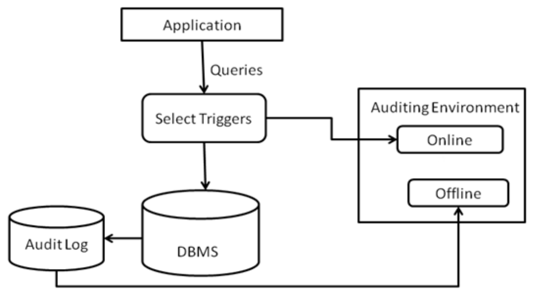

Fig 2 Overview of the auditing system with select trigger

Fig 2 shows the overall auditing system, and the relationship between SELECT triggers and the offline auditing system. SELECT triggers work as a filter for the offline system so that there are fewer queries and associated accesses to audit. Thus SELECT triggers can reduce the offline auditing effort to considerable amount.

C. Select trigger specification

The sensitive data and the trigger's action can be specified using following syntax:

on ACCESS to <SENSITIVE DATA> do <ACTION>

When the query is executed, the system records accesses to the sensitive data and stores this information in the query's ACCESSED internal state. The ACCESSED internal state is a per-query, in-memory relation that maintains access information and can be used by the trigger's action. This internal state resembles to SQL's use of the OLD and NEW variables for UPDATE triggers to mention previous value or newly created value. Once the query completes, the action is executed. The action can be specified as an SQL/T-SQL fragment, can reference the query's ACCESSED internal state and is executed as its own system transaction. The action executes even if the query is aborted to account for queries that read a subset of the result. Like traditional triggers, SELECT triggers are cascading. As a result, a SELECT trigger's action can trigger an UPDATE trigger, which in turn can trigger other SELECT triggers [4].

D. Audit Expression for SELECT Triggers

Sensitive information is specified in the form of an audit expression as follows [4].

CREATE AUDIT EXPRESSION <NAME>
AS SELECT <SENSITIVE COLUMNS>
FROM TABLES T, ..., Tn>
WHERE <PREDICATE>
FOR SENSITIVE TABLE <T>,
PARTITION BY <KEY>

In the above audit expression sensitive table specifies the table to examine for access. The partition-by key specifies what information should be stored in the ACCESSED internal state (such as the tuple's primary key).Values from the partition-by key are referred to as IDs.

Example 1 Consider a bank database with tables Customers (CustomerID, Name, Age, and Zip) and Balance (CustomerID, Balance). Suppose that we wish to specify that John's records are sensitive. This can be done using the following audit expression [4]:
CREATE AUDIT EXPRESSION Audit_John AS
SELECT * FROM Customers
WHERE Name = 'John'
FOR SENSITIVE TABLE Customers,
PARTITION BY CustomerID

Example 2 Suppose that we wish to specify that the personal information pertaining to all customers having balance 100000 is sensitive. This can be done by specifying the following expression.

CREATE AUDIT EXPRESSION Audit_Balance AS
SELECT P.* FROM Customers C, Balance B
WHERE C.CustomerID=B.CustomerID
AND Balance = 100000
FOR SENSITIVE TABLE Customers,
PARTITION BY CustomerID

For writing a SELECT trigger it is required to use the audit expression's name to specify sensitive data. For example to log the information of access to John's record SELECT trigger can be written as below:

CREATE TRIGGER Log_John_Accesses
ON ACCESS TO Audit_John AS
INSERT INTO Log
SELECT now (), userID(), sql(), CustomerID
FROM ACCESSED

Each entry in database log records the time of query execution, the user who executed the query, the query's SQL text and the CustomerID that were accessed, which is John' ID for the given audit expression (now(), userID() and sql() are database methods). The ON ACCESS TO clause specifies the audit expression (i.e., the sensitive data) and the associated attributes that are available from the ACCESSED internal state for the trigger's action (i.e., the partition-by key) [4].
Now if administrator may be interested in knowing the departments where the customers having balance=100000 belongs to. This can be done with following expression:

Departments (CustomerID, DeptID).

```
        CREATE TRIGGER
        Log_Balance_Dept_Accesses
ON ACCESS TO Audit_Balance AS
INSERT INTO Log
SELECT DISTINCT now(),    userID(), sql(),D.DeptID
FROM ACCESSED A, Departments D WHERE
A.CustomerID = D.CustomerID
```

SELECT triggers can be combined with other triggers to produce more sophisticated systems.
For example, SELECT triggers that write to the log can be combined with an INSERT trigger to notify the administrator if a user accesses more than ten sensitive Customers in a single day as follows.

```
CREATE TRIGGER Notify ON Log AFTER INSERT
AS
   IF (SELECT count(DISTINCT CustomerID) > 10
   FROM Log WHERE Date = NEW.Date
   AND UserID = NEW.UserID)
   SEND EMAIL
```
That write to the log can be combined with an INSERT

VII. Conclusion

This paper takes overview of several different notions of suspiciousness that vary in their disclosure detection assurance. Out of two broad classes of auditors semantic auditors have to execute the queries in the query log to determine suspiciousness of a query batch with respect to an audit expression. This is called an instance-dependent approach, where a query is said to access a tuple if deleting the tuple changes the query result on the database instance where the query was originally run. Syntactic auditors have certain desirable properties - the auditing task is independent of the underlying database instance and so in addition to the kind of auditing discussed here, such auditors could also be used for the task of online auditing. Under the instance independent approach, a query is said to have accessed the tuple if there is *some* database instance where deleting it changes the query result.

There are many scenarios that require row-level auditing support for SELECT queries that cannot be supported by existing database triggers. This paper gives overview of SELECT triggers as an enabler for data auditing. SELECT triggers also open up the possibility of real-time feedback on access to sensitive data. Other benefit of SELECT triggers is that they reduce the overall auditing run time by filtering queries and their associated accesses that must be analyzed by the offline system.

References

[1] R. Agrawal, R. J. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant “Auditing compliance with a hippocratic database.”, In VLDB, 2004.
[2] R. Motwani, S. U. Nabar, and D. Thomas “Auditing SQL queries”, In ICDE, 2008.
[3] Rajeev Motwani, Shubha U. Nabar, Dilys Thomas, “Auditing a Batch of SQL Queries”, Data Engineering Workshop (ICDE Wkshp), 2007 IEEE 23rd International Conference
[4] Daniel Fabbri #1, Ravi Ramamurthy _2, Raghav Kaushik _“SELECT Triggers For Data Auditing”, 2013 IEEE, ICDE Conference 2013
[5] D. Fabbri, K. LeFevre, and Q. Zhu-.”Policy Replay: Misconfiguration response queries for data breach reporting”, In VLDB, 2010.