Auditing a Batch of SQL Queries

Rajeev Motwani, Shubha U. Nabar, Dilys Thomas

Department of Computer Science, Stanford University

Abstract. In this paper, we study the problem of auditing a batch of SQL queries: given a set of SQL queries that have been posed over a database, determine whether some subset of these queries have revealed private information about an individual or group of individuals. In [2], the authors studied the problem of determining whether any single SQL query in isolation revealed information forbidden by the database system's data disclosure policies. In this paper, we extend this work to the problem of auditing a batch of SQL queries. We define two different notions of auditing - *semantic auditing* and *syntactic auditing* - and show that while syntactic auditing seems more desirable, it is in fact NP-hard to achieve. The problem of semantic auditing of a batch of SQL queries is, however, tractable and we give a polynomial time algorithm for this purpose.

1 Introduction

Auditing is the process of going over past actions to determine whether they were in conformance with official policies. In the context of database systems with data disclosure policies, 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 infer forbidden information.

The need for some sort of an audit mechanism in database management systems is clear. For instance, on receiving targeted advertisements, an individual might suspect his health-care provider of having leaked private information from his medical records to interested parties. If the provider's privacy policy stipulates that it does not release patient data to external parties, it would be in the best interests of the provider to be able to demonstrate compliance with this policy.

It is in this context that the authors of [2] introduced an auditing framework for checking whether any one query that had been posed in the past accessed/revealed some specified data. In their approach, users formulate audit expressions to specify parts of the data that they would like to ensure was not wrongfully disclosed. The audit component then returns all "suspicious" queries that accessed this data during their execution.

In general, however, it need not be any single query on its own that is the cause of a disclosure. Instead, the results of a few different queries in conjunction might enable a user to infer private data and therefore we study the problem of checking whether a batch of queries in conjunction could have caused an information leak. We restrict ourselves to the class of *select-project-join* (SPJ) queries and discover that an extension of the approach used in [2] would suffice for this class of queries. We call this approach *semantic auditing*. A problem with semantic auditing is that it requires that candidate queries actually be run against a *backlog database* which corresponds to the state of the database that existed at the time that the query was executed. A natural question to ask is whether this could be avoided. We formulate the notion of *syntactic auditing* for this purpose, but show that it is in fact NP-hard to achieve.

2 Related Work

Auditing aggregate queries: The problem of auditing queries has been extensively studied in the context of statistical databases [4, 10, 3, 6, 5, 9]. Statistical databases allow users to retrieve only aggregate statistics over subsets of its data. In this paper

we consider only SPJ queries and our work is orthogonal to the body of work on statistical databases.

Perfect privacy: In [8, 7] the authors consider the problem of ensuring "perfect privacy": as a database system releases various views of its data, does it disclose any information at all about a view that must be kept confidential. This is exactly the problem that we consider in this paper as well – the audit expression in our scenario specifies the view that must be kept confidential and the queries answered correspond to views of the data that were released. The notion of information disclosure that the authors use in [8, 7], however, is very strict in comparison, marking as suspicious many seemingly innocuous views. For example, consider a database containing the names and phone numbers of patients in a hospital and imagine that we wish to keep secret all the phone numbers listed in this database. A query asking for the names of all the patients in the hospital would be considered suspicious with respect to the secret view under the "perfect privacy" notion of security even though not a single phone number would have been reveealed 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 phone numbers column.

Auditing SQL queries: The work most closely related to ours, and that we build on is [2]. Recall the goal here is to identify every SQL query in the query log that accessed sensitive information. Here the data being subject to a disclosure review is specified very simply through an audit expression that very closely resembles a SQL query:

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

The audit expression can be viewed as an SPJ query, specifying a certain view of the database that it wishes to ascertain was not disclosed. It essentially identifies the tuples of interest from the cross-product of tables in the `FROM` clause via predicates in the `WHERE` clause. The audit expression thus asks for all queries that accessed *all* the `audit list` columns for *any* of these tuples. We illustrate this approach with some examples from [2]. Consider the audit expression:

```
AUDIT disease
FROM Patients p
WHERE p.zipcode = 94305
```

This expression asks for all queries that accessed the disease column of any patient living in the zipcode 94305. All such queries will be considered suspicious. Now consider the SQL query:

```
SELECT zipcode
FROM Patients p
WHERE p.disease = 'diabetes'
```

If any patient who has diabetes lives in zipcode 94305, this SQL query will be considered suspicious with respect to the above audit expression. This is because, in answering the query, the disease column of a patient living in zipcode 94305 was accessed. On the other hand, this SQL query would not be suspicious with respect to the audit expression given below:

```
AUDIT zipcode
FROM Patients p
WHERE p.disease = 'high blood pressure'
```

This is because this audit expression is only interested in checking if the zipcode of any patient with high blood pressure was revealed. But what if a patient has both diabetes and high blood pressure? Although this patient's address would be revealed by the SQL query, the fact that he had high blood pressure was not relevant to the

query and so it is reasonable to deem the SQL query unsuspicious. Note that in doing so, we take queries at their face value, assuming away background knowledge. For instance, a user might know that most patients with diabetes also have high blood pressure and thus his query ought to be considered suspicious. But in this paper we assume that users do not use external information to formulate queries so as to deduce information without detection.

We now formalize what it means for a query to be suspicious with respect to an audit expression. Consider an SPJ query of the form $Q = \pi_{C_Q}(\sigma_{P_Q}(T))$ and an audit expression of the form $A = \pi_{C_A}(\sigma_{P_A}(T))$. Here $C_{Q(resp.\ A)}$ are the columns that are projected out in Q (resp. A) and $P_{Q(resp.\ A)}$ are the predicates of Q (resp. A). We also use C^*_Q to denote all the column names that appear anywhere in the query Q.

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`*, i.e. $C^*_Q \supseteq C_A$.*

Note that we require the query to access all the columns of the audit expression to be called a candidate. Previous work [2] as well as other independent notions of privacy [1] require this constraint. The much stronger notion of perfect privacy [8, 7], which does not require this would mark as candidates many innocuous views as explained in Section 2.

Definition 2 (Indispensable Tuple). *A tuple $t \in T$ is indispensable to a query Q if the presence or absence of t makes a difference to the result of Q, i.e. $\pi_{C_Q}(\sigma_{P_Q}(T)) \neq \pi_{C_Q}(\sigma_{P_Q}(T - \{t\}))$*

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 review. 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. Note that the notion of indispensability here is identical to the notion of criticality in [8, 7].

For the class of SPJ queries, this condition of sharing an indispensable tuple translates to the following: a candidate query Q is suspicious with respect to an audit expression A if and only if $\sigma_{P_A}(\sigma_{P_Q}(T)) \neq \emptyset$. So now the audit process is simple, for every candidate query, Q, in the query log if the result of the running the query $\sigma_{P_A}(\sigma_{P_Q}(T))$ is non-empty, then Q is marked as suspicious.

3 Auditing a batch of SQL queries

In general, no single query in isolation may access all the columns of the `audit list`, instead a few queries together may cause sensitive information to be disclosed. For example, the audit expression might be

```
AUDIT name, disease
FROM Patients p
WHERE p.zipcode = 94305
```

Here the data that needs to be kept secret is the association between names and diseases of patients living in the zipcode 94305. Now consider the SQL queries:

```
SELECT zipcode
FROM Patients p
WHERE p.disease = 'diabetes'
```

```
SELECT p.name
FROM Patients p
WHERE p.zipcode = 94305
```

Note that neither of these queries on their own reveal the association between name and disease of any individual living in zipcode 94305, however the combination of the queries does reveal something. For instance, if there is only one patient in zipcode 94305 and this patient has diabetes, then these two queries have revealed the name, disease association for that individual. In general, there are subtle ways in which the results of queries could be combined to reveal information. One simplifying assumption that we make here is that the adversary is very powerful and knows exactly which tuples in the results of two queries join together, i.e. we assume that in each query he implicitly also selects the key column. In this way our auditing scheme is conservative - it may at times detect suspicious batches of queries even if the results of these queries in reality could not have been easily joined. We now extend the definition of suspiciousness to batches of SQL queries. We call this notion *semantic suspiciousness* and will shortly contrast it with the notion of *syntactic suspiciousness* that we define later.

Definition 4 (Semantically Suspicious Query Batch). *A batch of queries, $\mathcal{Q}$ is said to be semantically suspicious with respect to an audit expression A if there is some subset of queries $\mathcal{Q}' \subseteq \mathcal{Q}$ such that (1) a tuple $t \in T$ is indispensable to both A and every query in $\mathcal{Q}'$ and (2) the queries in $\mathcal{Q}'$ together access all the columns of the* `audit list` *in A.*

This definition of suspiciousness is a natural extension of the definition in [2] and as in that case, lends itself to an auditing approach where a query is executed over the database for every query in the query batch (we shall describe this approach in Section 3.2). A natural question to ask therefore is whether the overhead of executing all these queries can be avoided. To this end, we define another notion of suspiciousness for a query batch that is independent of the actual data in the database. We call this syntactic suspiciousness.

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.*

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.

Note that a semantic auditor would test suspiciousness with respect to the underlying database tables while a syntactic auditor would test suspiciousness of a query batch irrespective of the underlying database tables. Unlike semantic auditing, the hope is that syntactic auditing would require the auditor to only analyze the structure of queries in the query log, not the answers to the queries on the actual database tables. If an efficient syntactic auditor could be constructed, it could serve other purposes as well. For example, it could be used to audit queries in an online fashion - as queries are posed to the database system, the syntactic auditor could check to see if a new query in conjunction with already answered queries could cause a disclosure. If so, the query would be denied. A semantic auditor could not be used for this purpose, because as shown in [5], denials that are based on the underlying database instance can themselves leak information. Thus an efficient syntactic auditor would be of great use and we next investigate whether such an auditor can even be constructed.

3.1 Syntactic Auditing

Our main result here is that syntactic auditing is in fact NP-hard to accomplish.

Theorem 1 *Testing whether a batch of queries is syntactically suspicious with respect to an audit expression A is NP complete.*

PROOF: We provide a reduction from 3-SAT. Consider a 3-SAT formula $(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge \ldots (x_{m1} \vee x_{m2} \vee x_{m3})$. We now create a set of queries and an audit expression such that the queries are syntactically suspicious with respect to the audit expression if and only if the above 3-SAT formula is satisfiable.

Let $y_1, \bar{y_1}, \ldots y_n, \bar{y_n}$ be the literals that appear in the clauses of the 3-SAT formula. For each literal y_i create a column Y_i that can take on two possible values 0 or 1. For the j^{th} clause $(x_{j1} \vee x_{j2} \vee x_{j3})$ create a column X_j that can take on only one value. Let literal y_i occur in clauses j_1, j_2, ... j_k. We then create n query pairs where the ith pair looks like:

Q_i^+: `SELECT` X_{j_1}, X_{j_2}, X_{j_k}, Y_i
`FROM T`
`WHERE` $Y_i = 1$

and

Q_i^-: `SELECT` X_{j_1}, X_{j_2}, X_{j_k}, Y_i
`FROM T`
`WHERE` $Y_i = 0$

The audit expression is

`AUDIT` X_1, X_2, X_3, ..., X_m
`FROM` T

Now if the 3-SAT formula is satisfiable, then this batch of queries is syntactically suspicious with respect to the audit expression: Consider the queries Q_i^+ for every y_i that is set to true in the satisfying assignment and Q_i^- for every y_i that is set to false. This subset of queries and the audit expression all share an indispensable tuple - namely the tuple with $Y_i = 0$ for every y_i that is set to false and $Y_i = 1$ for every y_i that is set to true. Moreover, since every clause is satisfied, this subset of queries together selects all the columns that are in the `audit list` of the audit expression.

Similarly, if the batch of queries is syntactically suspicious with respect to the audit expression, then the 3-SAT formula must be satisfiable: Some subset of the queries must share an indispensable tuple with the audit expression. It cannot be the case that both Q_i^+ and Q_i^- are included in this subset for no one tuple can be indispensable to both Q_i^+ and Q_i^- as their selection predicates are contradictory. For each query in the subset of the form Q_i^+, we set y_i to be true and for each query of the form Q_i^-, we set y_i to be false. For all the y_is that are not set in the process, we set them arbitrarily to 0 or 1. Since the select columns of this subset of queries cover all the columns in the `audit list`, this ensures that every clause is set to true.

Thus we've shown that the 3-SAT formula is satisfiable if and only if the above query batch is suspicious with respect to the audit expression. □

Syntactic auditing of a batch of SQL queries is thus an NP-complete problem. We therefore continue our search for a semantic auditor.

3.2 Semantic Auditing

Theorem 2 *Testing whether a batch of queries is semantically suspicious with respect to an audit expression A is NP complete, when the underlying database table is given an implicit representation.*

PROOF: Given a $3-SAT$ formula, use the set of queries and audit expression from Theorem 1. and set the table in the input to be $\{0,1\} \times \{0,1\} \ldots \times \{0,1\} \times 1 \times \ldots \times 1$, where $\{0,1\}$ is for each of the binary columns and 1 for each of the unary columns. The set of queries is semantically suspicious with respect to the audit expression and the given table if and only if the $3-SAT$ formula is satisfiable. Note that the

implicit representation of the table is essential for the input to be of polynomial size. □

However in database systems, the database table is given explicitly. For this input representation there are polyomial time algorithms to test semantic suspiciousness.

Theorem 3 *There is a polynomial time algorithm to test semantic suspiciousness of a batch of SQL queries.*

PROOF: We first run the audit expression as a select query on the database table, modifying it slightly to select all the columns of the database. On the resulting view, we now run every single query from the query batch. Each time a query accesses a cell of the view, we mark it as accessed. If at the end of this process, there exists a row in the view all of whose `audit list` columns are marked as accessed then the batch of queries is syntactically suspicious with respect to the audit expression. To determine exactly which set of queries from the batch were involved in the disclosure, we can maintain with each marked cell, the set of queries that accessed it. The cross-product of all the query sets for each cell of each disclosed row gives us all the query sets that were involved in the disclosure. □

4 Conclusions

In this paper we introduced two broad class of auditors for auditing a batch of SQL queries – syntactic auditors and semantic auditors. 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 alse be used for the task of online auditing. We however show that syntactically auditing a query batch with respect to an audit expression is NP-hard. We therefore are forced to use semantic auditors that have to execute the queries in the query log to determine suspiciousness of a query batch with respect to an audit expression.

References

1. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *Conference on Innovative Data Systems Research*, 2005.
2. R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In *Proc. of the Intl. Conf. on Very Large Data Bases*, Sept. 2004.
3. F. Chin. Security problems on inference control for sum, max, and min queries. *J. ACM*, pages 451–464, 1986.
4. D. Dobkin, A. Jones, and R. Lipton. Secure databases: Protection against user influence. In *ACM TODS, 4(1)*, 1979.
5. K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. In *Proc. of the ACM Symp. on Principles of Database Systems*, June 2005.
6. J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. Auditing boolean attributes. In *Symposium on Principles of Database Systems*, pages 86–91, 2000.
7. A. Machanavajjhala and J. Gehrke. On the efficiency of checking perfect privacy. In *PODS*, 2006.
8. G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.
9. S. U. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards robustness in query auditing. In *VLDB*, 2006.
10. S. P. Reiss. Security in databases: A combinatorial study. *Journal of the ACM*, 26(1):45–57, 1979.