Inferring Secret Information in Relational Databases

Stefan Böttcher

University of Paderborn, EIM, Fürstenallee 11, D-33102 Paderborn, Germany

Abstract. We formalize the problem of finding information leaks in multi-user database systems, and we reduce this problem to the problem of inferring secret answers to database queries from other answers to database queries and a set of given Boolean integrity constraints. Furthermore, we investigate some sufficient conditions under which the answer to a query can be inferred from a previously answered set of database queries and a set of Boolean integrity constraints. Finally, show that the problem of finding information leaks is NP-hard, and we suggest a reformulation of the problem as a query composition and simplification problem.

1 Introduction

Whenever secret company information that can be accessed by multiple users is illegally leaked to a third party, it is crucial for the company to identify the information leak. We especially focus on scenarios where the information is leaked from a person that has an access right to the leaked information. We call this kind of information leakage an attack from the inside, in comparison to attacks from the outside where people who do not have an access right illegally try to access secret information. While access control helps to prevent attacks from the outside, access control is not applicable to our scenario where multiple users have an access right to the secret, but leaked information.

Instead our problem is related to inference and anti-inference, i.e. the problem is who of the users that have an access right to secret information that has been leaked by someone, did actually submit queries and did retrieve answers that are sufficient to infer the leaked information. In other words, given the knowledge an attacker can infer from his queries and his answers, can he or can't he infer the leaked secret information.

Furthermore, our problem is different from k-anonymity and l-diversity both of which regard relationships between all values given for certain attributes, whereas our secret information is an association of individual combinations of values, i.e. our secret can be uncovered even in situations where k-anonymity and l-diversity are violated.

Contributions

In comparison to related work, e.g. [2], the main contributions of this paper are the following:

- We present an introductory example that demonstrates that checking whether secret information is leaked by answering multiple queries is significantly more complicated than checking this for single queries only.

Böttcher S. (2007).
Inferring Secret Information in Relational Databases.
In *Proceedings of the 5th International Workshop on Security in Information Systems*, pages 179-187
DOI: 10.5220/0002437501790187

- We give a formal definition of secret information in databases.
- We formally define what it means that secret information can be inferred from query results.
- In contrast to related work, we discuss anti-inference attacks in scenarios where attackers can combine the knowledge retrieved from multiple queries.
- We identify anti-inference of single queries using common knowledge (like e.g. integrity constraints) as a special case of anti-inference for scenarios regarding multiple query attacks.
- We show that proving anti-inference is NP-hard.
- We provide a reformulation of the information leak identification problem as a query composition and simplification problem.

Paper organization
Section 2 presents a motivating example and summarizes the main contributions. Section 3 outlines a formal definition of the information leak identification problem, and Section 4 discusses steps towards a solution. Section 5 discusses related works and is followed by the Summary and Conclusions.

2 A Motivating Example

As an example, let us consider a relational database that contains two relations AB(account,balance) and AC(account,customer), and a secret information that "Jane has a bank account with a negative balance". When this secret has being leaked, i.e. is known to a third party, Jane may blame her bank that the leaked information originates from the bank's database. In such a case, it is important for the bank to know who inside the bank could have known about the secret information. Assuming that inside the bank the secret information is stored only in the database, the bank could analyze the queries Qi of different users in order to check which queries Qi have accessed information that is sufficient to infer the secret information and start interviews with the users submitting these queries. Or even better, the bank can prove that no query Qi has accessed information which is sufficient to infer the secret, i.e. the information leakage is not related to accessing the bank's database.
Given the relations AB and AC, the secret that "Jane has a bank account with a negative balance" can be computed by joining two tuples. For example, a query Q1

```
Q1 =  select customer from AC  where exists
      ( select * from AB where AB.account = AC.account and AB.balance < 0 )
```

would uncover the secret information.
Furthermore, let us consider two queries Q2 and Q3:

```
Q2 =  select customer from AC
Q3 =  select customer from AC  where not exists
      ( select * from AB where AB.account = AC.account and AB.balance < 0 )
```

Note that each of the queries Q2 and Q3 alone does not uncover the secret, i.e. we can not prove from the answer of Q2 alone or from the answer of Q3 alone the secret information that "Jane has a bank account with a negative balance".
However, if an attacker has submitted both queries Q2 and Q3 to the same database state and received the results R2 of Q2 and R3 of Q3, he could externally compute the difference R2-R3 which is the answer R1 to the query Q1. In other words, the

combination of two queries, here computing the set difference of two answers is sufficient to uncover the secret.
We use a second example of two queries Q4 and Q5 that count tuples and the combination of which uncovers the secret:

Q4 = select count(*) from AC
Q5 = select count(*) from AC where AC.customer != “Jane” or not exists
(select * from AB where AB.account = AC.account and AB.balance < 0)

Again, each of the queries Q4 and Q5 alone does not uncover the secret, i.e. we can not prove from the answer of Q4 alone or from the answer of Q5 alone the secret information that "Jane has a bank account with a negative balance". However, if an attacker has submitted both queries Q4 and Q5 to the same database state and received the results R4 of Q4 and R5 of Q5, he could externally compute the difference R4-R5, and see that R4-R5>0. Therefore, the attacker can conclude that “there is a customer Jane that has a negative bank account”. Note that the secret is also uncovered by knowing the results of Q2 and Q5 because the attacker could simply count the number of answer tuples given in Q2 which is the answer to Q4, however the knowledge of the answer to Q3 and the answer to Q4 is not sufficient to uncover the secret information.

3 Formal Problem Definition

A relation schema is the cartesian product of n domains D1,…,Dn (n≥1), where each domain is an ordered finite set of elements. Typical domains are sub-intervals of in

3.1 Logical Definition of the Relational Data Model

Definition 1 (relation schema, tuple, attribute):
A relation schema is the cartesian product of `n` domains `D1,…,Dn (n≥1)`, where each domain is an ordered finite set of elements. Typical domains are sub-intervals of integers or strings up to a limited length. The elements of the relation schema `(d1,…,dn)` are called *tuples* because they take one value `di` from each domain `Di`.
An *attribute* is a function from a relation schema to one of its domains which maps each tuple `(d1,…,dn)` to one value `di`.
Let `r` be a tuple of a relation schema `R` and `A` be an attribute defined for `R`, then we write `r.A` for applying the attribute `A` to the tuple `r`.

As usually only a small subset of the tuples that are possible according to relation schema are regarded to be `true`, we use an *interpretation* of the schema to distinguish which tuples are `true` and which are `false`. Only the tuples that are regarded to be `true` are stored in the *relation* corresponding to a relation schema.

Definition 2 (interpretation of a relation schema, relation):
An interpretation `I` is a function from a relation schema to the set {`true`, `false`}. A *relation* `R` *corresponding to a relation schema* `RS` is that subset of the tuples of `RS` which are interpreted as true, i.e. `R={r∈RS | I(r)=true}`.

Definition 3 (database schema, database state):
A *database schema* `DS={RS1,…,RSn}` is a set of relation schemas `RS1,…,RSn`.

A *database state* D of a database schema DS is a set of relations D={R1,…,Rn} where Ri is a relation corresponding to the relation schema RSi.

3.2 Definition of Relational Expressions

Within the Definition 4, Definition 5, and Definition 6, we recursively define operands, Boolean relational expressions and bag-valued relational expressions.

Definition 4 (Operands):
O1: Let r be a tuple variable bound to a relation R and A be an attribute defined for the relation R, then

r.A is an operand.

O2: Each constant is an operand.
O3: If Ri is a Bag-valued relational expression, then

count(Ri) is an operand.

O4: nothing else is an operand.

Definition 5 (Boolean relational expressions):
BRE1: Let op ∈ {<,=,>,≤,≠,≥} be an operator and o1 and o2 be operands, then

o1 op o2 is a Boolean relational expression.

BRE2: true and false are Boolean relational expressions.
BRE3: If B1 and B2 are Boolean relational expressions and R is a bag-valued relational expression, then also the following are Boolean relational expressions:

B1 and B2
B1 or B2
not B1
(∃ r ∈ R) B1
(∀ r ∈ R) B1

BRE4: Nothing else is a Boolean relational expression.

Definition 6 (Bag-valued relational expressions):
BVRE1: Each relation name of a relation in the given database is a Bag-valued relational expression.
BVRE2: Let R1 and R2 be Bag-valued relational expressions, let B be a Boolean relational expression, and let Ai,…,Aj be attributes defined for R1. Then, the following are Bag-valued relational algebra expressions:

[t1 ∈ R1 | B] - bag-selection: select bag of tuples of R1 for which B is true
R1 × R2 - cartesian product of R1 and R2
R1 $\cup_{bag}$ R2 - bag-union of R1 and R2
R1 $-_{bag}$ R2 - bag-difference of R1 and R2
R1 $\cup_{set}$ R2 - set-union of R1 and R2
R1 $-_{set}$ R2 - set-difference of R1 and R2
<Ai,…,Aj>(R1) - bag-projection of R1 to the attributes Ai,…,Aj
rdup(R1) - removes duplicates from R1

Note that bag-union and bag-projection do not remove duplicates. We have introduced these operations which are not part of the relational algebra because an attacker can use them to get more information than he would get if he were restricted to use set union and duplicate-free projection only. Whenever duplicate-free results

are desired, the rdup-operator can be applied. Furthermore, the bag-difference $R1 -_{bag} R2$ eliminates up to as many tuples from R1 as occur in R2.

Definition 7 (Relational expressions):
Each Boolean relational expression is a relational expression.
Each Bag-valued relational expression is a relational expression.
If B is a Bag-valued relational expression, then count(B) is a relational expression.

Definition 8 (Closed relational expressions):
A *tuple variable* r *is bound to a bag-valued relational expression* R if and only if each occurrence of r appears in the scope of a binding "$(\exists r \in R)$", "$(\forall r \in R)$ B1" or "$t1 \in R1$" of r to a bag-valued relational expression R. A relational expression Q is *closed* if and only if each tuple r variable occurring in Q is bound to a bag-valued relational expression.

3.3 Definition of Queries, Answers and Secret

Definition 9 (Answers to closed relational expressions):
Let Q be a closed relational expression, and D be a given database state to which Q can be applied. If Q is a Boolean relational expression, the result R is of type Boolean and expresses whether or not, the interpretation of Q is true for D. If Q is a Bag-valued relational expression, the result R is the bag of those tuples which are interpreted to be true according to D. Whenever R is the result of Q applied to D, i.e. R=Q(D), we call Q a *query* and the R an *answer* to Q in the database state D.

Definition 10 (Secret and Secret Query):
A *secret* is the answer Rs to a closed relational expression Qs, and we call Qs the *secret query*.

Note that we describe the secret by an answer to a closed relational expression Qs, i.e. the secret query. This includes Boolean secrets like one that represents that "Jane has a bank account with a negative balance", bag-valued secrets like a secret that "the set of customers that have a bank account with a negative balance is equal to {"Jane","Bob"}, and integer-valued secrets like a secret that "the number of customers that have a bank account with a negative balance is equal to 2". Note that this is more general than other approaches (e.g. [2]) that consider only a subset of bag-valued secrets.

3.4 Problem Definition

We use Q1, …, Qn for the closed relational expressions used in user queries. The problem is whether or not the answer Rs to Qs can be derived from the knowledge of Q1,…, Qn, Qs and the answers R1, …, Rn to Q1, …, Qn. This is more formally stated in the following problem definition.

Problem definition (secret is provable):
Let Q1,…,Qn,Qs be closed relational expressions and R1,…,Rn,Rs be the answers to Q1,…,Qn,Qs applied to the same database state D. The problem is to

check whether or not Rs is provable from Q1,…,Qn,Qs,R1,…,Rn, i.e. whether or not Q1(D)=R1,…,Qn(D)=Rn ==> Qs(D)=Rs.

Note that the answer to a secret can be of type Boolean, i.e. true or false, or it can be of type integer, or of the type bag-valued relational expression, i.e. a bag of values. In the latter case, the phrase 'Rs is inferrable' means that the exact bag of values can be inferred, i.e. not only a subset or superset of the values.

3.5 Treatment of Boolean Integrity Constraints

Definition 11 (valid database state and set of integrity constraints):
Together with a database schema, it is common to define a set of closed Boolean relational expressions called the *set* SIC *of integrity constraints*. Each database state in which every integrity constraint IC of SIC is equivalent to true is called a *valid database state*. When Boolean integrity constraints are used as a kind of "background knowledge", we are only interested in valid database states, i.e. we consider each integrity constraint IC to be equivalent to true.

Definition 12 (Boolean integrity constraints):
A *Boolean integrity constraint* is a closed Boolean relational expression Qic, the answer to which is equivalent to true for every valid database state.

This definition of Boolean integrity constraints includes key constraints, functional dependencies, referential integrity constraints and domain restriction constraints - or more generally all integrity constraints that are equivalent to a universally quantified Boolean relation expression "(∀r∈R)B1". Furthermore, Boolean integrity constraints include constraints of the form BVRE = Bag, where BVRE is a closed bag-valued relational expression and Bag is a bag of tuples. Finally, Boolean integrity constraints also contain constraints that are of the form of a comparison.

Sub-problem (secret is provable under Boolean integrity constraints):
The sub-problem is whether or not the answer to a secret query Qs can be proved from a set of Boolean integrity constraints that are used as a kind of "background knowledge" and a single user query Q1 with the answer R1.

The sub-problem is a special case of the problem definition given in the previous sub-section for the following reason. When Boolean integrity constraints are used as a kind of "background knowledge", we consider each integrity constraint IC to be equivalent to true. Therefore, we can regard each Boolean integrity constraint as a Boolean query Qic, the answer Ric of which is equivalent to true.

4 Steps towards a Solution

4.1 Solution Complexity

Lemma 1:
The sub-problem of whether or not the answer to a secret query Qs can be proved from a set of integrity constraints and a query Q1 with the answer R1 is NP-hard.

Proof sketch:
As a special case, we regard Q1 to be a Boolean query, and the answer R1 to Q1 to be true, and further we regard Qs to be a Boolean query with the answer true. Then, we can prove that the answer Rs to Qs is equivalent to true, if and only if Qs can be proved from Q1 and Qic1,…,Qicn. However, proving this for Boolean logic is NP-complete [5], therefore, proving the sub-problem is NP-hard.

Corollary 2:
The problem (a secret is provable from a set of pairs of query and answer) is NP-hard.

Proof:
The sub-problem (a secret is provable under Boolean integrity constraints) is just a special case of the problem. Therefore, from the sub-problem being NP-hard, we can conclude the problem to be NP-hard.

4.2 Composition and Simplification of the Given Queries

One idea towards a solution of the problem as defined above is to search for a composition of the given query expressions Q1,…,Qn that is equivalent to the secret query Qs. More precisely, Rs is inferable from Q1,…,Qn, Qs, R1,…,Rn if and only if there is a composition function f with the following properties:

1. f(Q1,…,Qn) is a closed relational expression that is an arbitrary composition of the given query expressions Q1, …, Qn by any combination of operators that may occur in relational expressions.
2. There is a substitution S = { Qi1/Ri1,…,Qin/Rin } that transforms f(Q1,…,Qn) into a relational expression S(f(Q1,…,Qn)) by replacing a subset Qi1,…,Qin of the query expressions Q1,…,Qn in f(Q1,…,Qn) with the corresponding answers, Ri1,…,Rin.
3. There is a sequence E1,…,En of equivalence transformations, i.e. query simplification steps that do not change the interpretation of a formula for any database state, which transform S(f(Q1, …, Qn)) into Qs, i.e. E1(…(En(S(f(Q1, …, Qn))))…) = Qs.

In other words, Rs is inferable, if Qs can be generated from Q1,…,Qn by composition, substitution of query expressions with results, and query simplification.
In this case, Rs is inferable because S(f(Q1, …, Qn)) is a description of how to compute Rs for the following reason. We only have to substitute the remaining query expressions Qi occurring in S(f(Q1, …, Qn)) with the corresponding answers Ri in order to get a relational algebra expression that can be evaluated by the database system and returns the secret Rs.

5 Related Works

Related works range from audit systems, to K-anonymity to theorem proving to views and has been contributed for different data models, ranging from relational databases over predicate calculus to XML databases. All work related to relational databases or XML covers only sub-problems, e.g. privacy violation detection for single queries, and often the sub-class of the queries is restricted, e.g. [2] which is based on

hippocratic databases [1] restricts the subclass to select-project join queries. Although a partial solution for multiple XML queries is discussed in [4], this shows non-inferability of XPath query result only for a special case.
Our problem of inferring database secrets by combining the results of multiple queries is different from k-anonymity violation checking for relational views, as discussed e.g. in [9], or from l-diversity [7], for the following reason. k-anonymity and l-diversity both regard relationships between all values given for certain attributes, whereas our secret information is an association of individual combinations of values. The difference is: even if 2-anonimity and 2-diversity are provably violated for a given pair (A,B) of attributes, our secret can still be uncovered for the following reason. We can not be sure that a secret as defined in our paper, i.e., an association between two concrete values (a1,b1) of the attributes A and B, can be derived because the 2-anonymity could be violated for other pairs of values (a2,b2) only and not for the pair (a1,b1) of the secret, i.e., 2-anonymity between attributes can be violated without the concrete values of the secret being leaked.
Theorem proving for first order predicate calculus has been investigated a long time, e.g. [8], but it is not directly applicable for the following reason. A database relation contains only the tuples that are interpreted to be true, but the closed-world assumption and operations like negation, set-difference, and bag-difference may require to consider also the tuples of a relation schema that are not in the relation. When it becomes necessary to model all these facts as being false, the number of formulas will be in the order of the number of database schema tuples which is too high for today's theorem provers.
Nesting of views has been used in query optimization. However, the approaches investigated focus on fast execution plans and avoid looking into all possible combinations of views which is required here.
Finally, in contrast to all other approaches to inference on database queries that regard only a subset of the database queries, e.g. [3], [6], we regard all relational algebra expressions, including bag-valued relational expressions allowing for duplicates.

6 Summary

Whenever secret company information that could be accessed by multiple user has been illegally leaked to a third party, it is crucial for the company to find all the possible information leaks. We have provided a formalization of secret information as being the answer Rs to a secret query Qs. Second, we have shown how secret information can be inferred from a set of user queries Q1,…,Qn and known answers to these queries. Third, we reduced the problem of finding information leaks to an inference problem among database queries. Fourth, we have proven that this problem is NP-hard. Fifth, we have reduced this problem to searching a composition function f that when applied to the user queries Q1,…,Qn generates a relational expression that can be transformed into the secret query Qs by query simplification and by substitution of query expressions with results. Whenever such a composition function f can be found, the secret Rs is inferable, i.e. we have found a potential information leak. Finally, as integrity constraints are only a special case of queries, each solution to our general problem is also a solution to database inference in the presence of integrity constraints or so called "global knowledge" which can be expressed as a

relational expression. Therefore, we consider this contribution to be useful for a wide range of applications that have to reason about database inferences and privacy violation detection.

References

1. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu: Hippocratic Databases. VLDB 2002, Hong Kong, 2002.
2. Rakesh Agrawal, Roberto J. Bayardo Jr., Christos Faloutsos, Jerry Kiernan, Ralf Rantzau, Ramakrishnan Srikant: Auditing Compliance with a Hippocratic Database. VLDB 2004, Toronto, Canada, 2004.
3. Foto Afrati, Chen Li and Prasenjit Mitra: On Containment of Conjunctive Queries with Arithmetic Comparisons. EDBT 2004, Heraklion, Crete, Greece, 2004.
4. Stefan Böttcher, Rita Steinmetz. Information Disclosure by XPath Queries. 3rd International Workshop on Secure Data Management 2006 (SDM). Seoul, Korea, 2006.
5. Garey, M.R., Johnson, D.S.: Computers and intractability. Bell Labs, 1979.
6. Anthony Klug: Locking Expressions for Increased Database Concurrency. Journal of the Association for Cornputmg Machinery, Vol 30, No I, January 1983, pp 36-54.
7. Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer: ℓ-Diversity: Privacy Beyond k-Anonymity. ICDE, Atlanta, USA, 2006.
8. D. W. Loveland: Automated Theorem Proving: A Logical Basis. North Holland, 1978.
9. Chao Yao, Xiaoyang Sean Wang, Sushil Jajodia: Checking for k-Anonymity Violation by Views. VLDB 2005, Trondheim, Norway, 2005.