Resisting Tracker Attacks By Query Terms Analysis

Xiaoqi Ma

School of Science and Technology, Nottingham Trent University, Clifton Campus, Nottingham NG11 8NS, England, UK

xiaoqi.ma@ntu.ac.uk

Keywords: Database security, tracker attacks.

Abstract. Tracker attacks pose a serious threat to databases, especially those used in manufactory and management in industry. These attacks can be used to infer sensitive information in databases and they are difficult to detect. This paper proposes a new approach to dealing with such attacks by analysing each disjunctive term in every query statement. Potential tracker attacks will be detected and then suppressed to avoid any further real attacks. A sample database table and a sample attack are given and analysed to show the effectiveness of the new approach.

Introduction

Database technology is more and more heavily used in manufactory and management in industry. While this technology greatly improves the performance and reliability of manufactory and management, it also faces severe risks, which, if not dealt with properly, will bring catastrophe and result in big loss.

Many databases used in industry are statistical databases, which are designed to allow query access to aggregate data. Such databases are subject to various attacks, among which *tracker attack* is one of the most powerful and dangerous attacks. With trackers, attackers can employ statistical queries to deduce sensitive information from database, while each query is legal and the result is safe, but the combination of them will cause leak of secrets.

A lot of research work has been conducted on this problem and a number of useful solutions have been proposed. One of the earliest solutions is Stonebraker's query modification method [1,2]. Stonebraker's work used old fashioned QUEL query language and only had limited support to joined queries. Denning *et al.* thoroughly described the tracker attacks in [3]. Simpson *et al.* used query modification to fight against tracker attacks [4]. Dobkin *et al.* were among the earliest researchers working on secure databases against malicious users' inference. A precise model about how attackers combined queries to infer sensitive information from database was presented in [5], although no complete countermeasure was proposed. A remarkable model called *dynamic disclosure monitor* was described by Toland *et al.* in [6]. This model maintained a history database and formed an index on it to store users' knowledge and therefore to resist inference attacks. The model was "sound and complete" but it "unnecessarily examined the entire history database in computing inferences" [6]. Byun proposed a secure anonymisation technique dealing with data anonymisation of incremental dataset [7]. This technique involves some kind of information loss or data distortion, therefore not suitable for manufactory and management in industrial context. The data sanitisation process is discussed in [8]. In this process, a hybrid approach is recommended by Amiri. This approach is effective "in terms of data utility at the expense of computational speed" [8]. Due to its heuristic nature, this approach is not as efficient as many other methods such as query modification, and the database owners are suggested to sanitise their database by themselves, making this method difficult to use. Also this approach still results in some data distortion although the author claims to have kept the distortion to a very low level, which is still not acceptable to some key industrial and business applications, as the precision and usability of data cannot be guaranteed.

A new approach is proposed in this paper. This approach involves checking each query by analysing its all disjunctive terms. If any disjunctive term identifies a single tuple in the database table, it will be regarded as a potential tracker and will be marked. Any further queries containing this disjunctive will be either suppressed or modified. A concrete example is given to illustrate this new approach. This example shows how tracker attacks can be eliminated from a real industry database.

Tracker Attacks

Data in statistical database can be retrieved without explicit access to them. Access to sensitive information in database in this way without the awareness of the database owner or administrator is dangerous and unacceptable. There are many different ways to conduct such kind of attack, some of which can be successfully detected and resisted. However, there is still no perfect solution to deal with tracker attacks [3,9].

A *tracker* is a predicate which can be used to infer or track down information about a tuple in a database. A *general tracker* is such a tracker which can be used to infer information of any tuples, especially those secret and inadmissible ones [3].

For example, Table 1 shows a database table about IT devices sales information. Suppose the columns Units and Price/Unit are individually sensitive and direct access to these individual items are not permitted, while statistical queries to the database are allowed.

Customer	Gender	Products	Units	Price/Unit
Alice	F	Laptop	5	625
Bob	M	Scanner	8	80
Catherine	F	Scanner	12	77
David	M	Printer	7	122
Elizabeth	F	Laptop	8	510
Frank	M	Printer	9	115
Gavin	M	Scanner	30	69
Helen	F	Laptop	10	495
Ian	M	Scanner	1	105

Table 1: An IT Sales Database Table

Suppose standard security mechanisms have been used upon this database. For example, all queries resulting in less than 3 tuples or more than 7 tuples will be suppressed to make sure both resultant tuples and their complements are sufficiently large. Despite these mechanisms, the attacker can create a tracker to track down any sensitive information in the database table. For an individual item, suppose R is the predicate uniquely identifying the tuple in question, and the attacker can find a tracker T so that both predicates $R \vee T$ and $R \vee \text{NOT}(T)$ are allowed. The target tuple is the only one existing in the results of both queries.

For example, the attacker wants to know how many laptops Elizabeth has ordered. The predicate R uniquely identifying the tuple is `Customer = 'Elizabeth'`, which is obviously not permitted in this example. The tracker T can be `Products = 'Scanner'`. The query

```
SELECT SUM(Units) FROM SALES
WHERE Customer = 'Elizabeth' OR Products = 'Scanner'
```

gives 59 and the query

```
SELECT SUM(Units) FROM SALES
WHERE Customer = 'Elizabeth' OR NOT (Products = 'Scanner')
```

gives 39. It is easy to know that the total number of units in the table is 90. Therefore the attacker can infer the number of laptops Elizabeth has ordered by calculating 59+39−90=8, without direct access to Elizabeth's individual record.

Resisting Tracker Attacks

The main characteristic of tracker attacks is that the attacker makes use of two queries in forms of $R \vee T$ and $R \vee \mathrm{NOT}(T)$. Suppose the set of whole database table is D, and the result set of a query Q is denoted as Set(Q), we always have Set(T) $\cup$ Set(NOT(T)) = D, whatever T is. Also, we have either Set(R) $\subseteq$ Set(T) or Set(R) $\subseteq$ Set(NOT(T)). As T can be regarded as NOT(NOT(T)), without losing generality, we always suppose Set(R) $\subseteq$ Set(T).

The database management system (DBMS) should be very alert on queries of the disjunctive form $R \vee T$ where T itself can be a negative predicate. For each such query, tests will be conducted on each disjunctive term to see whether any individual row of database table can be identified by one of these disjunctive terms. If this is the case, the DBMS has three choices, one of which is reject the query and do not return any results. This is radical, as it turns down all potentially suspect queries, even though they are not necessarily malicious. This strategy may significantly affect the usability of database. An alternative is query modification [2,4,9]. This way, users make queries as if there is no restriction. The DBMS (or more precisely, the access control mechanism of the DBMS) modifies the queries so that only permissible data will be returned as query results to the users. Then the DBA or the designer of the access control mechanism has the full control over the returned results. The third solution is to maintain an indicator for each tuple of the database table. If a query of the disjunctive form $R \vee T$ is made, and R is detected to be able to identify a single tuple t in the database table for the first time, the indicator of tuple t should be turned on, and this query will be allowed as normal. If some time later R appears in another query $R \vee T'$, this new query should be either rejected or modified, depending on the application. This strategy might look too strict, as in the latter query we do not necessarily have T'=NOT(T), and then the attacker cannot use $R \vee T'$ (together with $R \vee T$) to infer sensitive information of tuple t. But consider an example. The attacker queries $R \vee T$, where T is a tracker and gets the correct result. Then the attacker queries $R \vee T'$, where T'=NOT(T)$\vee X$, and X is another predicate intended to make $T' \neq$NOT(T). If the second query is accepted, the attack will then query $R \vee T''$, where T''=NOT(T)$\vee$NOT(X). Similar to the second query, this time the attacker will also be accepted and get the correct data returned to him. Then the attacker can combine the results of last two queries and have $(R \vee T') \vee (R \vee T'') = R \vee T' \vee T''$ $= R \vee (\mathrm{NOT}(T) \vee X) \vee (\mathrm{NOT}(T) \vee \mathrm{NOT}(X)) = R \vee \mathrm{NOT}(T)$. It can be seen that accepting the second query takes the risk of potential trackers. In implementation, the indicators can be stored in a file, in an internal data structure, or in an additional field to the original table.

Take the above IT sales database table as an instance. Just as before, the predicate R uniquely identifying the Elizabeth's tuple is `Customer = 'Elizabeth'`. The first query

```
SELECT SUM(Units) FROM SALES
WHERE Customer = 'Elizabeth' OR Products = 'Scanner'
```

gives 59 again. All the following queries containing `Customer = 'Elizabeth'` as a disjunctive term should be rejected or modified even though it does not look like a tracker. Otherwise the attacker can query

```
SELECT SUM(Units) FROM SALES
WHERE Customer = 'Elizabeth' OR Products = 'Laptop'
```

This query seems to be all right as `Products = 'Laptop'` does not equal to `NOT (Products = 'Scanner')`. So the attacker will get the result 31 on this query. However, if the second query is accepted, the attacker can continue to query

```
SELECT SUM(Units) FROM SALES
WHERE Customer = 'Elizabeth' OR Products = 'Printer'
```

and get 16, as this query does not seem to be a tracker either. However, the attacker has already collected enough information to infer the number of units Elizabeth has ordered, which is (59+31+16−90)/2=8. The reason is that the attacker divides the predicate `NOT (Products = 'Scanner')` into two parts, namely `Products = 'Laptop'` and `Products = 'Laptop'`, and they seems not to relate to `NOT (Products = 'Scanner')`. To avoid such kind of attacks, the second and the third queries should not be allowed, as they contain the predicate `Customer = 'Elizabeth'`.

The disjunctive query can be a bit tricky, as both R and T do not have to be atomic; they can be compound formulas. It is not a problem for R. Suppose $R=R_1 \lor R_2$, and R corresponds to a single tuple of the database table, we must have $R=R_1=R_2$, since Set(R_1) $\subseteq$ Set(R) and Set(R_2) $\subseteq$ Set(R). Therefore we can easily detect disjunctive terms corresponding to single rows. If $R=R_1 \land R_2$, we can easily recognise this by considering conjunction as multiplication and disjunction as addition in ordinary arithmetic. For any conjunctive term producing a query which results in a single tuple, this term should be recognised as R. It is not a problem for T either, as we only need to detect R.

The mechanism has very low complexity. The time complexity is O(N_t), where N_t is the number of terms appearing in the query. The space complexity is O(N_r), where N_r is the number of rows in the database table.

Summary

In this paper, a new approach is proposed to resist tracker attacks. To conduct a tracker attack, the attacker needs to make two queries in disjunctive forms, and one disjunctive term of each query condition can identify a single tuple in the target database table. Therefore, every disjunctive term in each query statement should be analysed to see whether it can track down a row. If so, there are three choices: (1) reject the query; (2) modify the query; (3) mark the row, allow the query, and in the future if any query containing such term is detected again, the new query will be either rejected or modified. The third choice is recommended. Although this method seems to be too strict, an example shows the necessity of it. The tricky situation of disjunctive query is also discussed. An example shows that tracker attacks can be resisted by this method while others may have problems.

References

[1] M. Stonebraker and E. Wong: *Access Control in a Relational Data Base Management System by Query Modification*. ACM/CSC-ER Proceedings of the 1974 Annual Conference (1974)

[2] M. Stonebraker: *Implementation of Integrity Constraints and Views by Query Modification*. Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA (1975)

[3] D. E. Denning, P. J. Denning and M. D. Schwartz: *The Tracker: A Threat to Statistical Database Security*. ACM Transactions on Database Systems, 4(1):76-96 (1979)

[4] A. C. Simpson, D. J. Power and M. Slaymaker: *On Tracker Attacks in Health Grids*. Proceedings of the 2006 ACM Symposium on Applied Computing, pages 209-216, Dijon, France (2006)

[5] D. Dobkin, A. K. Jones and R. J. Lipton: *Secure Databases: Protection Against User Influence*. ACM Transactions on Database Security, 4(1):97-106 (1979)

[6] T. S. Toland, C. Farcas, C. M. Eastman: *Dynamic Disclosure Monitor (D^2Mon): An Improved Query Processing Solution*. Secure Data Management – SDM2005, volume 3674 of Lecture Notes in Computer Science, pages 124-142 (2005)

[7] J.-W. Byun, Y. Sohn, E. Bertino and N. Li: *Secure Anonymization for Incremental Datasets*. Secure Data Management – SDM2006, volume 4165 of Lecture Notes in Computer Science, pages 48-63 (2006)

[8] A. Amiri: *Dare to Share: Protecting Sensitive Knowledge with Data Sanitization*. Decision Support Systems, 43(1):181-191 (2007)

[9] D. Power, M. Slaymaker, E. Politou and A. Simpson: *Protecting Sensitive Patient Data via Query Modification*. Proceedings of the 2005 ACM Symposium on Applied Computing, pages 224-230, Santa Fe, New Mexico, USA (2005)