OLAP Means On-Line Anti-Privacy*

Lingyu Wang, Duminda Wijesekera and Sushil Jajodia
Center for Secure Information Systems, George Mason University
Fairfax, VA 22030-4444, USA
{lwang3,dwijesek,jajodia}@gmu.edu

June 19, 2003

Abstract

In this paper we investigate the privacy breaches caused by multi-dimensional range sum queries in OLAP (Online Analytic Processing) systems. Our results show that the sensitive information stored in the underlying data warehouses can be easily compromised by malicious users with legitimate range queries. This compromise is possible even when users are restricted to some special class of range queries. We present algorithms that compromise individual tuples with only range queries. We study the conditions under which the compromises are possible. We also analyze the number of queries required for each compromise, as well as the complexity and completeness of those algorithms. Our study reveals the seriousness of the privacy issue in OLAP systems and provide better understanding of the problem for further research on the control methods.

1 Introduction

Online analytical processing (OLAP) is an important decision support tool. OLAP is intended to assist users in exploiting the large amount of data stored in underlying data warehouses for useful trends and patterns. Contrary to this initial objective, OLAP systems can be misused to gain access to the protected sensitive data. This results in the breach of individual's privacy and jeopardy organizational interest. Access control alone is insufficient to thwart such misuses, because information not directly disclosed can be inferred indirectly from the answers to legitimate queries, which is known as the *inference problem*. Unfortunately, most of today's OLAP applications do not take into account the privacy threats caused by the inference problem. It is the subject matter of this paper to demonstrate how easily privacy can be breached can be achieved and how serious the threats can be.

One feature that distinguishes OLAP systems from general-use databases is the restricted form of queries and the high efficiency in answering such queries. Typically OLAP users are only interested in certain classes of well-formed queries. For example, multi-dimensional range query [19] is one of the most important OLAP queries. Such well formed queries usually survey information about the general properties rather than the specific details. Hence they better serve the need of OLAP users in discovering know ledges than randomly formed queries do. Although those queries usually involve aggregations of a large amount

*This work was partially supported by the National Science Foundation under grant CCR-0113515.

of data, they can be answered by most OLAP systems in merely a few seconds. This is achieved through pre-computation and materialization of multi-dimensional views from which the queries can be easily derived. For example, data cubes [18] organize multi-dimensional aggregates into hierarchies for the fast computation of OLAP queries.

Because OLAP systems may work with restricted forms of queries, a natural question is to ask whether the privacy breaches caused by the inference problem may be eliminated or alleviated by such restrictions. This conjecture is strengthen by the fact that compromising general-purpose databases usually depends on the availability of randomly formed queries. For example, the so-called Tracker [10, 11] and the linear system attack [14] both need to inquire about the sum of arbitrary subsets of sensitive values, which are likely meaningless to OLAP users and are therefore restricted. However, the answer to the question is *no*. Our results show that although more sophisticated mechanisms are required, it is usually possible and easy to attack the OLAP systems with legitimate queries. Our algorithms can find trackers using merely range queries to subvert the restrictions on small queries. Even if only the range queries involving even numbers of tuples are answered, our algorithms are still capable of compromises in most cases.

Another weaker but seemingly more plausible conjecture is that at least the restricted form of queries may alleviate the damages of such attacks. For example, the compromised sensitive values may only comprise a small portion of the entire data warehouses. If so then the privacy issue can still be ignored. Unfortunately, it is untrue according to our study. We discuss the methods that can compromise the entire data warehouses in the worst case.

Finally, it seems attractive to directly apply existing inference control methods to OLAP systems. The inference problem in general-purpose statistical databases has been investigated since 70's with many inference control methods proposed. However, as discussed above, general-purpose databases are not optimized for any specific kinds of queries and inference control methods must cope with all possible queries. As a natural trade-off, the run time of those methods are usually proportional to the size of the queries. For example, the Audit Expert [6] needs $O(mn)$ time to process a single query, where m and n is the number of answered queries and tuples respectively. Moreover, most inference control methods in statistical databases are launched only after queries have arrived. Therefore, the time consumed by those methods adds to the total response time. This prohibitive overhead renders them impractical for OLAP systems as they demand instant responses to large queries.

The first contribution of this paper is that it demonstrates the privacy breaches caused by the inference problem to be an important issue to OLAP systems. Although a similar issue has received attentions recently in off-line applications such as data mining [3, 2, 16, 15, 13], it is still ignored in on-line systems such as OLAP. In this paper we elaborate on this issue by showing how easy a compromise is, and how serious the threats can be. Secondly, our study provides a better understanding of the inference problem in OLAP systems. We study the inference methods that exploit only multi-dimensional range queries and data cubes. The insights will fertilize further studies on inference control methods.

The rest of the paper is organized as the follows. Section 2 reviews existing inference control methods. Section 3 reviews basic concepts of OLAP and introduces our model. Section 4 studies the inferences caused by unrestricted range queries. Section 5 discussed compromising with restricted range queries. Section 6 introduces a demo system that we have implemented. Section 7 concludes the paper.

2 Related Work

Inference control has been extensively studied in statistical databases [9, 1, 12] and the proposed methods are roughly classified into two main categories; *restriction based* techniques and *perturbation based* techniques. Perturbation-based techniques usually do not apply to on-line systems such as OLAP, and therefore we do not discuss them here. Restriction based techniques include restricting the size of *query sets* (i.e., the entities that satisfy a single query) [17], restricting the size of overlaps [14] between query sets, detecting inferences by auditing all queries asked by a specific user [6, 22, 4], suppressing sensitive data in released statistical tables [7], grouping data and treating each group as a single entity [5, 23]. These inference control methods proposed for statistical databases are designed to control random queries and hence are usually infeasible for OLAP applications which demand short response time for large queries.

Recently a variation of the inference control problem has received attentions in off-line knowledge discovery applications such as data mining [3, 2, 16, 24, 15, 13]. Those work resemble the perturbation-based methods in statistical databases, but they focus on preserving the data distribution model instead of the precision of point estimators after random perturbation [3]. Applying such approaches to online systems like OLAP encounters a fundamental difficulty. That is, it is hard to predict the purpose of online data analysis. As a decision support tool, OLAP allows users to interactively exploit the data. However, OLAP systems can not make assumptions about the knowledge users want to discover from the data. Users may be looking for association rules, but they may also be interested in classification results instead. Hence it is difficult to decide on users' behalf what is the purpose of the knowledge discovery. Without this prediction, the bias and inconsistence introduced by random perturbations may render query results imprecise and useless.

The inference control of OLAP data cubes and multi-dimensional range queries is discussed in [25, 26]. In [25] sufficient conditions are given which guarantee data cubes to be safe from inferences when the data core is full or dense (i.e., the number of known values is zero or under the given bound respectively), and only *skeleton queries* are allowed. More general multi-dimensional range queries are considered in [26]. By reducing such queries to a simpler form, efficient methods are proposed to determine whether the data core can be compromised. Those studies provide partial solutions to the inference problem of OLAP systems. However, they also reveal new vulnerabilities of OLAP systems to inferences. In this paper we study the problem from a totally different perspective. That is, the attacker's perspective. We refer to some of the theoretic results presented in [25, 26] but use them to attack on privacy.

3 Preliminaries

This section reviews the basic concepts about OLAP data cubes and their inference control. We then introduce the model and notations used in the rest of the paper.

3.1 An Example of Data Cube and Its Inference Problem

We first illustrate the basic concepts of OLAP data cubes and their inference problem by an example. Suppose a portion of the *data core* is shown as a relation in Figure 1. The attributes $Year$ and $Name$ are *dimensions*, and the attribute Adj is a *measure*.

A data cube [18] can be constructed from the relation $Salary\_Adjustment$ as shown in the cross-tabular

Year	Name	Adj
2002	Alice	1000
2002	Bob	500
2002	Mary	-2000
2003	Bob	1500
2003	Mary	-500
2003	Jim	1000

Figure 1: Relation $Salary\_Adjustment$.

in Figure 2. The data cube is grouped by the dimensions $Year$ and $Name$, and the *aggregation function* is the SUM. The first column and the first row of the table corresponds to the values of the dimension $Year$ and $Name$, respectively. The values of the measure Adj is shown in the middle of the table. Each value in the last column and the last row (but not both) gives the aggregation $SUM(Adj)$ for the corresponding row and column, respectively. The value at the last column and row is the aggregation $SUM(Adj)$ for all the values of Adj in the table.

	Alice	Bob	Mary	Jim	SUM(Adj)
2002	1000	500	-2000	N/A	-500
2003	N/A	1500	-500	1000	2000
SUM(Adj)	1000	2000	-2500	1000	1500

Figure 2: An Example of a Two-dimensional Data Cube.

The SUMs shown in Figure 2 are called *skeleton queries*, because they involve whole columns (or rows) of the table. More generally, *multi-dimensional range queries* (or range queries for short) aggregate values in continuous multi-dimensional ranges of the table. For example, in Figure 2, the summation of Alice and Bob's salary adjustments in 2002 is a range query, but that of Alice and Mary's is not. Range queries involving partial columns or rows correspond to the *slicing* or *dicing* operations of data cubes.

We adopt an intuitive notation for denoting multi-dimensional ranges from now on. For example, the range $[(2002, Alice), (2002, Jim)]$ includes Alice, Bob and Mary's salary adjustments in 2002. Similarly $[(2002, Alice), (2003, Bob)]$ includes Alice and Bob's salary adjustments in 2002 and 2003.

Suupose the company invites an analyst Mallory to analyze the data, but worries that Mallory may misuses the sensitive information about each individual. Hence Mallory is not supposed to know any values of the measure Adj in Figure 2, but is allowed to access any aggregations of those values. Moreover, Mallory is free to inquiry about the dimensions $Year$ and $Name$.

Suppose Mallory asks a range sum query for $[(2002, Alice), (2003, Alice)]$. Alice's salary adjustment in 2002 is disclosed by the answer to this query, although Mallory is not supposed to know this value. Hence this value is *compromised*, and an inference occurs. This kind of trivial inferences are easy to prevent by restricting queries involving single values. An alternative way for Mallory to learn the same value is to ask two range sum queries, $[(2002, Alice), (2002, Mary)]$ and $[(2002, Alice), (2002, Bob)]$, and then to calculate their difference. It is more difficult to detect such kind of *overlap attacks* or more generally *linear*

system attacks [6, 14], because these usually require the system to audit the entire history of queries asked by each user. Moreover any control mechanism can be easily subverted if users collude.

3.2 The Model

We use $\mathbb{I}, \mathbb{R}, \mathbb{I}^k, \mathbb{R}^k$ and $\mathbb{R}^{m \times n}$ for the set of integers, reals, k-dimensional integer vectors, k-dimensional real vectors and $m \times n$, respectively. For any real vectors $u, v, t \in \mathbb{R}^k$, we use the notation $u \leq v$ to mean that $u[i] \leq v[i]$ for all $1 \leq i \leq k$, and $t \in [u, v]$ to mean that $u \leq v$ and $min\{u[i], v[i]\} \leq t[i] \leq max\{u[i], v[i]\}$ for $1 \leq i \leq k$.

Data Cubes Instead of using relational terms, it is more convenient to model a data cube as a system of sets of integer vectors, where each integer vector stands for a tuple and each set stands for a range query (we say a query we refer to both the query specification and the set of tuples aggregated by the result to the query).

We use closed integer intervals $[1, d_i]$ ($1 \leq i \leq k$ for some fixed k) for the *dimensions* by assuming a one-to-one mapping between the integers and the dimension values. A *tuple* is an integer vector of the form $(x_1, x_2, \ldots, x_k)$, where each x_i is an integer in $[1, d_i]$. The Cartesian product $\prod_{i=1}^{k}[1, d_i]$ contains all the possible tuples that may appear in any data core. In another word, a *data core* is a subset of this Cartesian product. We use the same notation to specify *range queries* as in Section 3.1. We assume SUM queries and omit the aggregation function. That is, $[t_1, t_2]$ is a range query over the data core C, and the result to the query is the set $\{t : t \in [t_1, t_2]\}$. However, we may directly refer to such a set as a query. The following example illustrates these concepts.

Example 1 *We translate the example in Section 3.1 into the stated notations. The two dimensions are* $[1, 2]$ *and* $[1, 4]$*. The Cartesian product* $[1, 2] \times [1, 4]$ *contains eight tuples, but only six of them appear in the data core* $\{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (2, 4)\}$*. In Figure 3 we model the data cube shown in Figure 2 as a system of sets of integer vectors.*

	1	2	3	4	
1	(1,1)	(1,2)	(1,3)		[(1,1),(1,4)]
2		(2,2)	(2,3)	(2,4)	[(2,1),(2,4)]
	[(1,1),(2,1)]	[(1,2),(2,2)]	[(1,3),(2,3)]	[(1,4),(2,4)]	[(1,1),(2,4)]

Figure 3: A Data Cube Modeled as A System of Sets of Integer Vectors.

Inferences Because we denote a group of range queries over a data core as a set system, the *incidence matrix* of the set system can be used to characterize the queries (let $\mathcal{S}$ and $\mathcal{M}(\mathcal{S})$ be the set system and its incidence matrix, then $\mathcal{M}(\mathcal{S})[i, j] = 1$ if the j^{th} element is a member of the i^{th} set, and $\mathcal{M}(\mathcal{S})[i, j] = 0$ otherwise). Example 2 shows an example of incidence matrices.

Example 2 *In Figure 3, the incidence matrix $\mathcal{M}(\mathcal{S}_1)$ of the group $\mathcal{S}_1 = \{[(1,1),(1,1)]\}$ consists of a single row $[1,0,0,0,0,0]$. The incidence matrix $\mathcal{M}(\mathcal{S}_2)$ of $\mathcal{S}_2 = \{[(1,1),(1,3)],[(1,2),(1,3)]\}$ is:*

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

We introduce a *derivability* relation $\preceq$ between two groups of range queries $\mathcal{S}_1$ and $\mathcal{S}_2$ over the same data core. Let $\mathcal{M}(\mathcal{S}_1)$ and $\mathcal{M}(\mathcal{S}_\in)$ be the incidence matrix of $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Then we say that $\mathcal{S}_1$ is derivable from $\mathcal{S}_2$, denoted as $\mathcal{S}_1 \preceq \mathcal{S}_2$, if $\mathcal{M}(\mathcal{S}_1) = M \cdot \mathcal{M}(\mathcal{S}_2)$ for some $M \in \mathbb{R}^{|\mathcal{S}_1|\times|\mathcal{S}_2|}$. We say that a tuple t is *compromised* by a group of range queries $\mathcal{S}$ if $\{\{t\}\}$ (or t for short) is derivable from $\mathcal{S}$, that is, $t \preceq \mathcal{S}$. We say that $\mathcal{S}$ is safe if no tuple in the data core is compromised.

Example 3 *Following Example 2, we have that $\mathcal{S}_1 \preceq \mathcal{S}_2$, because $\mathcal{M}(\mathcal{S}_1) = [1,-1] \cdot \mathcal{M}(\mathcal{S}_2)$. Hence $\mathcal{S}_2$ is unsafe because it compromises the tuple $(1,1)$.*

We do not have measures in our model because they are not relevant for the type of inferences we discuss. For example, the inferences discussed in Section 3.1 does not depend on the specific values of the measure Adj we use in Figure 1. We deal with the general case where sensitive measures are unbounded real numbers. Hence the only way that users can learn such values is through the derivability relation. In some special cases where the measures are bounded and users happen to know the boundaries, potentially more information may be inferred [22, 21].

4 Compromising Using Random Range Queries

This section discusses inferences caused by random range queries. We discuss two kind of attacks, namely, *trivial attacks* and *tracker attacks* with range queries.

Trivial Attacks with Range Queries We begin with no restriction on the range queries users may ask. In such a case inferences are trivial, because users can simply ask a range query containing only the targeted tuple with an aggregation function such as SUM, AVG, or MAX or MIN. This is known as the *small query set attack* in statistical databases [8].

Definition 1 (Trivial Attack) *A trivial attack happens when range queries containing only the targeted tuple are answered.*

In order to compromise a tuple t, users can simply ask the query $[t,t]$. An alternative way is to ask a query $[t_1,t_2]$, where t_1,t_2 are two distinct tuples satisfying that $t \in [t_1,t_2]$ and $\mid [t_1,t_2] \mid = 1$. To satisfy $t \in [t_1,t_2]$, t_1 and t_2 can be chosen in such a way that $t_1[i] \leq t[i] \leq t_2[i]$ holds for ordered dimensions, and $t_1[i] = t[i] = t_2[i]$ for dimensions without order. The condition $\mid [t_1,t_2] \mid = 1$ can be determined by asking the query $[t_1,t_2]$ with $COUNT$.

The trivial attack can be easily prevented by executing the query and examining the cardinality of the results before releasing them to users. The query $[t,t]$ can be simply denied even without execution, because even if $\mid [t,t] \mid = 0$, answering such a query still discloses the fact that no tuple satisfies the condition

specified by t, known as *negative* inferences [8]. It may be appealing to scrutinize the $COUNT$ queries, such that users can not find two tuples t_1 and t_2 to satisfy the condition $\mid [t_1, t_2] \mid = 1$. However, $COUNT$ queries might be very difficult to control because users have the extra knowledge that each tuple contributes either zero or one to the result of a $COUNT$ query. The complexity of inference control for such binary values is very high [20]. For this reason we make a conservative assumption that users are not restricted to $COUNT$ queries and they know which tuple is present and which is absent.

Tracker Attack with Range Queries One simple mechanism to subvert the restriction that queries containing single tuples are denied is the *tracker* [10]. The basic idea of tracker is to pad the restricted queries with more tuples such that they become legitimate, and then remove the padding with subsequent queries. Tracker attacks can be used either to directly compromise tuples or as intermediate steps to obtain restricted queries required by other attacks. Finding a tracker composed of merely multi-dimensional range queries is more difficult than with random queries, but nevertheless possible.

Definition 2 *[Tracker Attack With Range Queries] Suppose only the range queries containing no less than n_t tuples are answerable. Given a targeted range query $[t_1, t_2]$ that is not answerable, a tracker attack happens when there exists a group of answerable range queries $\mathcal{S}$ satisfying $[t_1, t_2] \preceq \mathcal{S}$.*

Algorithm *Range_Tracker* shown in Figure 4 builds a tracker to derive the restricted range query $[t_1, t_2]$ by using totally $k + 2$ range queries. Intuitively, we can view $[t_1, t_2]$ as a k-dimensional *box* (some of the dimensions may have the size one) obtained by the intersection of totally $2k$ *planes*. Those planes divide the data core C into 3^k disjointed blocks including $[t_1, t_2]$ itself. The algorithm starts by searching through the $3^k - 1$ blocks around $[t_1, t_2]$ for $[t_a, t_b]$ that contains no less than n_t tuples. If the algorithm can find $[t_a, t_b]$ then it returns the following $k+1$ range queries besides $[t_a, t_b]$; $[t_c, t_d]$ is the smallest range query that contains both $[t_a, t_b]$ and $[t_1, t_2]$; and the $[u_i, v_i]$s are the largest range queries that is contained by $[t_c, t_d]$, and contain $[t_a, t_b]$ but not $[t_1, t_2]$.

Proposition 1 justifies the correctness of the Algorithm *Range_Tracker* and gives a sufficient condition for finding a tracker. The result shows that unless the threshold n_t is very large, finding a tracker with range queries is always possible. However, choosing a large n_t may render the system useless because many drill-down queries having small cardinalities will be denied.

Proposition 1 *The result $\mathcal{S}$ of Algorithm* Range_Tracker *has the following properties:*

- $[t_1, t_2] \preceq \mathcal{S}$.
- *All range queries in $\mathcal{S}$ are answerable with respect to n_t.*
- *$\mathcal{S}$ can always be obtained if $n_t \leq \frac{|C|}{3^k}$.*

Proof:

- $[t_1, t_2]$ can be calculated from $\mathcal{S}$ with the a series of set operations as the follows.

$$[t_1, t_2] = [t_c, t_d] - \bigcup_{i=1}^{k} ([u_i, v_i] - [t_a, t_b]) - [t_a, t_b].$$

Algorithm *Range_Tracker*

Input: A data core C with k dimensions $[1, d_i]$ ($1 \leq i \leq k$), a threshold n_t, and the targeted range query $[t_1, t_2]$ with $\mid [t_1, t_2] \mid < n_t$ holds.

Output: a group of $k+2$ queries $\mathcal{S}$ if successful, ϕ otherwise.

Method:

(1) **Let** $[t_a, t_b]$ be a range query satisfying:

$\mid [t_a, t_b] \mid \geq n_t$, and for $1 \leq i \leq k$ one of the follows holds

$t_a[i] = 1$ and $t_b[i] = t_1[i] - 1$,

$t_a[i] = t_1[i]$ and $t_b[i] = t_2[i]$, or

$t_a[i] = t_2[i] + 1$ and $t_b[i] = d_i$.

(2) **Then**

Let $[t_c, t_d]$ be a range query satisfying that

$t_c[i] = min\{t_a[i], t_1[i]\}$ and $t_c[i] = max\{t_b[i], t_2[i]\}$ for $1 \leq i \leq k$.

For $1 \leq i \leq k$

Let $[u_i, v_i]$ be a range query satisfying that

$u_i[i] = t_1[i]$, $v_i[i] = t_2[i]$ and

$u_i[j] = t_c[j]$, $v_i[j] = t_d[j]$ for $j \neq i$.

Let $\mathcal{S} = \{[u_i, v_i] : 1 \leq i \leq k\} \cup [t_a, t_b] \cup [t_c, t_d]$.

(3) **Else**

Let $\mathcal{S} = \phi$.

(4) **Return** $\mathcal{S}$.

Figure 4: An Algorithm For Constructing A Tracker Using Range Queries

It then follows that $[t_1, t_2] \preceq \mathcal{S}$ because $\mathcal{M}([t_1, t_2]) = M \cdot \mathcal{M}(\mathcal{S})$, where $M = [-1, -1, \ldots, -1, k-1, 1] \in \mathbb{R}^{1 \times (k+2)}$.

- First $\mid [t_a, t_b] \mid \geq n_t$ is given by the step (2) of Algorithm *Range_Tracker*. Then because $[t_a, t_b] \subseteq [u_i, v_i]$ for $1 \leq i \leq k$, we have that $\mid [u_i, v_i] \mid \geq \mid [t_a, t_b] \mid \geq n_t$. Similarly we have $\mid [t_c, t_d] \mid \geq n_t$. Hence all the range queries in $\mathcal{S}$ can be answered with respect to n_t.

- The algorithm *Range_Tracker* is successful if it can find the range query $[t_a, t_b]$ satisfying $\mid [t_a, t_b] \mid \geq n_t$. The $4k$ values $1, t_1[i], t_2[i], d[i]$ ($1 \leq i \leq k$) divide the data core C into totally 3^k disjointed blocks with $[t_1, t_2]$ in the middle (some of the ranges may be empty). One out of the 3^k ranges must contain $\frac{|C|}{3^k}$ or more tuples. Because $\mid [t_1, t_2] \mid < n_t \leq \frac{|C|}{3^k}$, $[t_a, t_b]$ can always be found.

□

Example 4 *Figure 5 gives an example of trackers using two dimensional range queries. Suppose that* $\mid [t_1, t_2] \mid < n_t$ *holds for some* $n_t \leq \frac{|C|}{9}$. *Then one of the eight blocks around* $[t_1, t_2]$ *must contain no less than* n_t *tuples. Suppose* $\mid [t_a, t_b] \mid \geq n_t$. *Then we can calculate* $[t_1, t_2]$ *as* $[t_1, t_2] = [t_c, t_d] - ([u_1, v_1] - [t_a, t_b]) - ([u_2, v_2] - [t_a, t_b]) - [t_a, t_b]$. *Clearly all the queries at the right side of this equation have a cardinality greater or equal to* n_t.

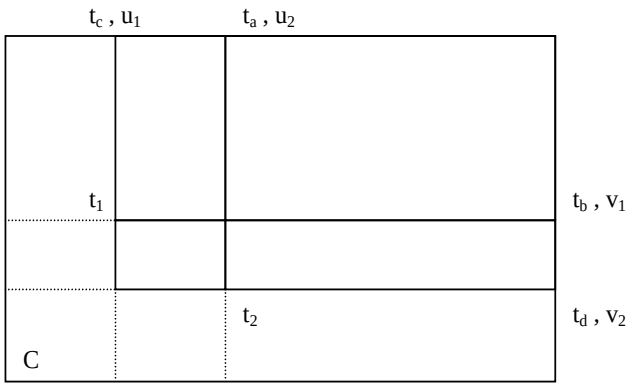

Figure 5: An Example of Trackers Using Two Dimensional Range Queries.

The tracker found by Algorithm *Range_Tracker* uses only $k + 2$ range queries to derive the restricted query. The queries are all formed in constant time. The running time of the algorithm is dominated by the searching for $[t_a, t_b]$, which can be implemented by asking at most $3^k - 1$ range $COUNT$ queries.

Tracker attack, as well as the more general *linear system attack* [14], exploits the derivability relation between groups of queries to achieve inferences. The same technique can be used to deter the attack. For example, auditing the entire history of queries asked by each user and constantly checking for any possible inferences [6]. Such an approach suffers from several inherent drawbacks. An attacker can easily achieve his objective by manipulating a few queries, but the system must check all the answered queries. Knowing this weakness, an attacker can paralyze the system by asking the desired queries one at a time, padded with a large amount of dummy safe queries involving many tuples. Such an behavior might be suspicious in general-purpose databases but just normal in OLAP systems. Another drawback is that the set of answerable queries depends on the order in which users ask queries [4]. A badly chosen order may render the number of answerable queries very small. This may not be an issue in general-purpose databases because users only ask the desired queries. However, in OLAP systems users tend to *waste* queries. More specifically, users ask queries in large batches, such as a data cube, and many of those queries are only used to stage further interactions. Under the auditing control such kind of operations may likely result in the situation where users find that they can not ask any more queries after only a few interactions.

5 Compromising with Restricted Range Queries

In the previous section we have seen that it is relatively easy to compromise the data core when no restriction is enforced on the type of range queries that users may ask. In this section we shall investigate the cases where users are only allowed to ask a special class of range queries. We show that although considerably more difficult than with random range queries, inferences are still possible under such restrictions.

5.1 Even Range Query Attack

First we consider answering only the *even range queries*. The intuition is that any union or difference of two even range queries yield another even range query. Hence compromises with such simple set operations will not succeed, since a single tuple composes an *odd range query*. For example, the inference in Example 3 will fail because the query $[(1,1),(1,3)]$ is an odd range query and will be suppressed. Under this restriction it is less straightforward to compromise any tuple in the data core. Nevertheless, we show in Example 5 that inferences are still possible in this case.

Example 5 *For the data core shown in Figure 3, we can compromise the tuple* $(1,1)$ *in a more sophisticated way using only even range queries. First consider* $\mathcal{S} = \{[(1,1),(1,2)],[(1,2),(2,2)],\{(1,1),(2,2)\}\}$. *Obviously we have that* $\mathcal{M}((1,1)) = [\frac{1}{2},-\frac{1}{2},\frac{1}{2}] \cdot \mathcal{M}(\mathcal{S})$. *Now the question is how do we get the answer to* $\{(1,1),(2,2)\}$ *since it is not even a range query. We need more queries for this purpose. Let* $\mathcal{S}' = \{[(1,1),(2,4)],[(1,2),(1,3)],[(2,3),(2,4)]\}$. *Clearly all queries in* $\mathcal{S}'$ *are even range queries. Moreover we have that* $\mathcal{M}(\{(1,1),(2,2)\}) = [1,-1,-1] \cdot \mathcal{M}(\mathcal{S}')$.

Definition 3 (Even Range Query Attack) *Even range query attack happens when a group of range queries* $\mathcal{S}$ *compromises the targeted tuple and all range queries in* $\mathcal{S}$ *contain even number of tuples.*

The key observation in Example 5 is that the three queries in $\mathcal{S}$ form an *odd cycle*. Here we consider each tuple as a vertex in a graph and each set of two tuples as an edge (notice that the set system we used to model the data cube is naturally a hypergraph). With such an odd cycle we can always compromise any tuple in the cycle. Basically we begin from the targeted tuple and traverse the cycle. At each step we could *remove* a tuple shared by the two adjacent edges. Then when we reach the last edge we complete the compromise with the targeted tuple being added twice and all other tuples in the cycle being removed.

However, the attack works only if all the edges in the cycle are even range queries or can be derived from even range queries as in Example 5. For this purpose we need the *QDT Graph* introduced in [26]. The *QDT graph* has the data core as its vertex set and a special group of sets of two tuples as its edge set. A brief algorithm is given in Figure 5.1 to explain how QDT graph is constructed from the data core for the given even range query $[t_1,t_2]$. Basically we *pair* adjacent vertexes (tuples) by adding edges (sets of two tuples) to the QDT graph. The Proposition 2 states that the algorithm guarantees every edge added to the QDT graph to be derivable from some set of even range queries over $[t_1,t_2]$.

Proposition 2 *Any edge* $\{t_a,t_b\}$ *in the output of the Algorithm* QDT_Graph *can be derived from a set of even range queries over the input* $[t_1,t_2]$.

Proof (Sketch): The Algorithm *QDT_Graph* pairs vertexes in totally k rounds, with the vertexes differing in the last i dimensions being paired in the i^{th} round. We prove by induction on i. The initial case with $i=1$ is trivial because the added edges are even range queries.

Suppose in the edges added in up to the $i-1$ round satisfy the hypothesis. In the i^{th} round an edge $\{t_a,t_b\}$ is added into E. Let S_a and S_b be the sets of vertexes that have the same first $k-i$ dimensions as t_a and t_b, respectively. Then we know that all vertexes in S_a and S_b are already paired up in the first $i-1$ rounds and the edges between them therefore satisfy the hypothesis. Moreover, let S_c be the set of vertexes

Algorithm *QDT_Graph*
Input: a data core C and an even range query $[t_1, t_2]$.
Output: an edge set E.
Method:

(1) **Let** $\mathcal{S} = [t_1, t_2]$ and $E = \phi$.
(2) **For** $i = 1$ to k
  Let t_a, t_b be two tuples in $\mathcal{S}$ satisfying:
    (2.a) t_a and t_b differ in the last i dimensions,
    (2.b) there does not exists $t_c \in \mathcal{S}$ such that
      $t_a[j] = t_c[j]$ for all $j < i$ and $t_a[i] < t_c[i] < t_b[i]$.
  Let $E = E \cup \{\{t_a, t_b\}\}$.
  Let $\mathcal{S} = \mathcal{S} - \{t_a, t_b\}$.
(3) **Return** E.

Figure 6: An Algorithm for Constructing a QDT Graph

satisfying that for any $t_c \in S_c$, $t_a[j] = t_c[j]$ for all $j < i$ and $t_a[i] < t_c[i] < t_b[i]$ holds. The condition (2.a) of the Algorithm *QDT_Graph* implies that S_c must be an even range query. Let $S_{ab} = S_a \cup S_b \cup S_c$, then S_{ab} must also be an even range query. We conclude the proof by deriving $\{t_a, t_b\}$ as the follows:

$$\mathcal{M}(\{t_a, t_b\}) = [1, -1, -1, -1] \cdot \mathcal{M}(\{S_{ab}, S_c, S_a, S_b\}) \quad (1)$$

□

Now that all edges in a QDT graph are guaranteed to be derivable from even range queries, we can compromise the data core by searching for an odd cycle in the QDT graph. Algorithm *Even_Range_Query_Attack* given in Figure 5.1 additively builds an QDT graph for the even range queries enclosing the targeted tuple t. It then utilize a breadth-first-search (BFS) in the QDT graph starting from t, in the attempt to find an odd cycle containing t. If it finds such an cycle, it returns the set of even range queries from which all the edges in the cycle can be derived. This set of even range queries can be found using equation (1) in the proof of proposition 2. If a cycle cannot be found, then the algorithm begins to build the QDT graph for another even range query. The process is repeated with the cardinality of the even range query (for which the QDT graph is built) increasing until either t is compromised or no even range query is left unprocessed.

The algorithm *Even_Range_Query_Attack* attempts to compromise t by asking as few range queries as possible. The algorithm achieves the goal with two heuristics. Firstly, it begins to build QDT graph for smaller even range queries, and moves to larger queries only if an odd cycle can not be found. The second heuristic is that the BFS in QDT graphs finds the shortest odd cycle, which implies that less number of even range queries are required to derive the targeted tuple. Despite those efforts, the number of queries required for the compromise largely depends on the actual data core and could be large. The running time of the algorithm is dominated by the construction of QDT graphs. Constructing a QDT graph for one range query can be fulfilled in linear time in the cardinality of the query. However, in the worst case the algorithm has to build QDT graphs for all possible range queries containing the targeted tuple before a failure is reported. In comparison to the tracker attack discussed in Section 4, the potentially high complexity of even range query

Algorithm *Even_Range_Query_Attack*
Input: a data core C and a targeted tuple t.
Output: a set of even range queries $\mathcal{S}$ compromising t if successful, or ϕ if failed.
Method:

(1) **Let** $\mathcal{S} = \phi$ and $E = \phi$.
(2) **For** $i = 1$ to $\lfloor \frac{|C|}{2} \rfloor$
  While there exists an un-processed range query $[t_a, t_b]$ satisfying $| [t_a, t_b] | = 2i$, and $t \in [t_a, t_b]$.
    Let $E = E \cup$ *QDT_Graph*(t_a, t_b, C)
    Do BFS in $G(C, E)$ to find an odd cycle $\mathcal{E}$ containing t.
    If $\mathcal{E}$ exists
      Let $\mathcal{S}$ be a set of even range queries satisfying $\mathcal{E} \preceq \mathcal{S}$.
      Return $\mathcal{S}$.
    Else
      Let $[t_a, t_b]$ be marked as processed.
(3) **Return** ϕ.

Figure 7: An Algorithm for Even Range Query Attack

attack reflects the effectiveness of the control of even range query.

5.2 Indirect Even Range Query Attack

The completeness of the Algorithm *Even_Range_Query_Attack* is left open in the previous section. That is, when the algorithm fails, is there any other way to compromise the targeted tuple? Unfortunately, the answer is *yes*. We shall show that the tuple can be compromised in a more indirect manner when an cycle cannot be found after QDT graphs have been constructed for all the range queries containing the targeted tuple.

Definition 4 *Indirect even range query attack happens when the targeted tuple is compromised through the compromises of other tuples.*

Example 6 *In Figure 3, after we compromise the tuple* $(1, 1)$ *basically we could successively compromise all other tuples using the even range queries connecting each pair of them.*

First we need the following result proved in [26], which states that for any range query $[t_1, t_2]$, if we build QDT graphs for all the even range queries contained in $[t_1, t_2]$ and additively union the edge sets of all those QDT graphs, then the final outcome must be a connected graph.

Lemma 1 *Given any even range query* $[t_1, t_2]$ *over the data core* C*, let* $\mathcal{S}$ *be the group of all even range queries contained by* $[t_1, t_2]$ *and let* $E = \bigcup_{\forall [t_a, t_b] \in \mathcal{S}}$ QDT_Graph(t_a, t_b, C)*. Then the graph* $G(C, E)$ *must be connected.*

□

Because of Lemma 1, we could compromise the targeted tuple t by first compromising any other tuple t_1 using the Algorithm *Even_Range_Query_Attack*. We then find the shortest path $\mathcal{P}$ from t to t_1 in the graph $G(C, E)$ described in Lemma 1. All tuples from t_1 to t can then be successively compromised. The set of even range queries for indirect even range query attack is stated in Algorithm *Indirect_Attack* shown in Figure 5.2.

Algorithm *Indirect_Attack*
Input: a data core C and a targeted tuple t.
Output: a set of even range queries $\mathcal{S}$ if successful, ϕ otherwise.
Method:
(1) **Let** $\mathcal{S}$ =*Even_Range_Query_Attack*(C, t).
(2) **If** $\mathcal{S} = \phi$
For any unprocessed tuple $t_1 \in C - \{t\}$
Let $\mathcal{S}_1$ =*Even_Range_Query_Attack*(C, t_1)
If $\mathcal{S}_1 \neq \phi$
Let$\mathcal{P}$ be the shortest path between t and t_1
Let$\mathcal{S}_2$ be the set of even range queries satisfying $\mathcal{P} \preceq \mathcal{S}_2$.
Let$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$.
Return $\mathcal{S}$.
Else
Lett_1 be marked as processed.
(3) **Return** ϕ.

Figure 8: An Algorithm for Indirect Even Range Query Attack

Several heuristics may help the Algorithm *Indirect_Attack* to minimize the number of even range queries required by the actual attack, although the number could be huge in some cases. Firstly, the intermediate tuple t_1 may be chosen in such a way that compromising t_1 requires as few queries as possible. Secondly the tuple t_1 should be close to t such that the path $\mathcal{P}$ is short. Those two goals must be balanced to reduce the overall number of required queries. Finally redundant construction of QDT graphs should be avoided. While processing a new tuple t_{new}, the QDT graph of any even range query that contains both t_{new} and at least one processed tuple must have already been constructed and should not be considered again.

The completeness of the algorithm *Indirect_Attack* is guaranteed by the following result proved in [26]. Given a data core, if all possible QDT graphs has been constructed and their edge sets are unioned but still no odd cycle can be found, then the data core is safe from even range query attack. Hence if the algorithm *Indirect_Attack* returns an empty set for any targeted tuple t, then we know that there is no other way to compromise t. Moreover, no other tuple can be compromised by even range query attack.

5.3 Skeleton Query Attack

When users are restricted to only skeleton queries with trivial attacks suppressed, compromises is not always possible.. For example, the data core in Figure 3 is safe under such restrictions. In [25] we give other sufficient conditions for the data core to be safe from such compromises. However, in some cases such an attack is still attainable.

Definition 5 *A skeleton query attack happens when the targeted tuple is compromised by skeleton queries that contain more than one tuples.*

Example 7 *The data core shown in Figure 9 is subject to skeleton query attack. First observe that all skeleton queries contain more than one tuple, and hence trivial attack is impossible. The skeleton query can be obtained through the following equation:*

$$\mathcal{M}(\{(1,1)\}) = [1,1,-1,-1] \cdot \mathcal{M}(\{[(1,1),(1,4)],[(2,1),(2,4)],[(1,2),(4,2)],[(1,3),(4,3)]\}) \quad (2)$$

	1	2	3	4
1	(1,1)	(1,2)	(1,3)	
2		(2,2)	(2,3)	
3	(3,1)			(3,4)
4	(4,1)			(4,4)

Figure 9: A Data Core Subject to Skeleton Query Attack.

The general skeleton query attack can be achieved using the similar techniques used in linear system attack [6]. That is, we first transform the incidence matrix of a group of queries into its reduced row echelon form (RREF). Then the queries compromise the targeted tuple iff a row vector in the RREF contains only one non-zero element corresponding to the targeted tuple. However, compared to the linear time of building a QDT graph in previous sections, the elementary row transformation used to obtain RREF for m queries runs in $O(m^2)$ time. Hence when the number of range queries required for compromises increases, the attack may become infeasible. Nevertheless, compromising is still easier than controlling the compromises because attackers can focus on the queries essential for the attack but the system has to check all answered queries.

The Algorithm *Skeleton_Query_Attack* shown in Figure 5.3 exploits a simple heuristic to reduce the size of matrices to be transformed. It starts from the targeted tuple and check all the skeleton queries containing this tuple. If no compromise is possible, it checks more tuples that overlap the checked queries with at least one tuple. This is based on the simple fact that if a group of queries compromise a tuple, then any query in the group must share at least one tuple with some other query in the group, and at least one query must contain the compromised tuple.

Algorithm *Skeleton_Query_Attack*
Input: a data core C and a targeted tuple t.
Output: a set of skeleton queries $\mathcal{S}$ if sucessful, ϕ otherwise.
Method:

(1) **Let** $\mathcal{S} = \phi$ and $T = \{t\}$.
(2) **While** T is not empty
Let $\mathcal{T}$ be the set of unprocessed skeleton queries
containing at least one tuple in T.
Let $\mathcal{S} = \mathcal{S} \cup \mathcal{T}$.
Let $M = \mathcal{M}(\mathcal{S})$.
If the RREF of M contains a unit row vector corresponding to t
Return $\mathcal{S}$.
Else
Let all tuples in T be marked as processed.
Let T be the unprocessed tuples contained in at least one query in $\mathcal{S}$.
(3) **Return** ϕ.

Figure 10: An Algorithm for Skeleton Query Attack

6 A Demo System

We are in the progress of developing a demo system to demonstrate the privacy breaches in practical OLAP systems. The system is implemented in Java and composes of three basic components. An attack module implements the algorithms presented in this paper to compromise targeted tuples with specified classes of range queries. An evaluation module determines if the attacks are theoretically possible by checking the RREF of incidence matrices. A backend processing module answers the range queries posed by the other two modules. The current implementation uses files as data sources but future work may provide interfaces to actual data warehouses. Ongoing work also includes experiments using the system with both synthetic data and real data in order to exploit the effectiveness of various control mechanisms.

7 Conclusion

In this paper we have shown the privacy breaches caused by the multi-dimensional range queries to be a serious threat to OLAP systems. We showed that it is easy to compromise a targeted tuple or restricted query with only a few legitimate range queries, when the type of range queries is not restricted. We then showed that restricting users to even range queries makes compromises more difficult. However, the compromise is still possible with the algorithms we presented. Finally we showed that even when users are restricted to skeleton queries only, compromise can still be achieved. Those findings invoke more research on this issue and also provide a better understanding of the problem.

References

[1] N.R. Adam and J.C. Wortmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[2] D. Agrawal and C.C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 439–450, 2000.

[4] L. Brankovic, M. Miller, P. Horak, and G. Wrightson. Usability of compromise-free statistical databases. In *Proceedings of ninth International Conference on Scientific and Statistical Database Management (SSDBM '97)*, pages 144–154, 1997.

[5] F.Y. Chin and G. Özsoyoglu. Security in partitioned dynamic statistical databases. In *Proc. of IEEE COMPSAC*, pages 594–601, 1979.

[6] F.Y. Chin and G. Özsoyoglu. Auditing and inference control in statistical databases. *IEEE Trans. on Software Engineering*, 8(6):574–582, 1982.

[7] L.H. Cox. Suppression methodology and statistical disclosure control. *Journal of American Statistical Association*, 75(370):377–385, 1980.

[8] D. Denning. *Cryptography and data security*. Addison-Wesley, 1982.

[9] D.E. Denning and P.J. Denning. Data security. *ACM computing surveys*, 11(3):227–249, 1979.

[10] D.E. Denning, P.J. Denning, and M.D. Schwartz. The tracker: A threat to statistical database security. *ACM Trans. on Database Systems*, 4(1):76–96, 1979.

[11] D.E. Denning and J. Schlorer. A fast procedure for finding a tracker in a statistical database. *ACM Transactions on Database Systems*, 5(1):88–102, 1980.

[12] D.E. Denning and J. Schlörer. Inference controls for statistical databases. *IEEE Computer*, 16(7):69–82, 1983.

[13] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings 2003 ACM PODS Symposium on Principles of Database Systems*, 2003.

[14] D. Dobkin, A.K. Jones, and R.J. Lipton. Secure databases: protection against user influence. *ACM Trans. on Database Systems*, 4(1):97–106, 1979.

[15] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings 2003 ACM PODS Symposium on Principles of Database Systems*, 2003.

[16] A. Evfimievski, R. Srikant, , R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of the 8th Conference on Knowledge Discovery and Data Mining (KDD'02)*, 2002.

[17] L.P. Fellegi. On the qestion of statistical confidentiality. *Journal of American Statistic Association*, 67(337):7–18, 1972.

[18] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, crosstab and sub-totals. In *Proceedings of the 12th International Conference on Data Engineering*, pages 152–159, 1996.

[19] D.T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *Proceedings 1997 ACM SIGMOD International Conference on Management of Data*, pages 73–88, 1997.

[20] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Auditing boolean attributes. In *Proc. of the 9th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 86–91, 2000.

[21] Y. Li, L. Wang, X.S. Wang, and S. Jajodia. Auditing interval-based inference. In *Proceedings of the 14th Conference on Advanced Information Systems Engineering (CAiSE'02)*, pages 553–568, 2002.

[22] F.M. Malvestuto and M. Mezzini. Auditing sum queries. In *Proceedings of the 9th International Conference on Database Theory (ICDT'03)*, pages 126–146, 2003.

[23] J.M. Mateo-Sanz and J. Domingo-Ferrer. A method for data-oriented multivariate microaggregation. In *Proceedings of the Conference on Statistical Data Protection'98*, pages 89–99, 1998.

[24] S. Rizvi and J.R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th Conference on Very Large Data Base (VLDB'02)*, 2002.

[25] L. Wang, D. Wijesekera, and S. Jajodia. Cardinality-based inference control in sum-only data cubes. In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS'02)*, pages 55–71, 2002.

[26] L. Wang, D. Wijesekera, and S. Jajodia. Precisely answering multi-dimensional range queries without privacy breaches. Technical Report, 2003. Available at http://ise.gmu.edu/techrep/2003/.