

Controlling Inference and Information Flows in Secure Databases

Shiuh-Pyng Shieh Chern-Tan Lin Yi-Shian Juang
Department of Computer Science and Information Engineering
National Chiao Tung University
Taiwan, R.O.C. 30050

Abstract

In a secure database system, controlling users' accesses is the main method to prevent the sensitive information from being compromised. The database management system must deny any user's access that will cause the exposure of unauthorized information. In this paper, we propose two models for the protection of information in a secure database management system, namely, inference control model and information flow control model. The inference control model analyzes the static protective restrictions (access permissions and integrity constraints) placed on views and relations. If unauthorized information can be inferred from a set of views which violate the protective restrictions on the views, the views cannot be granted. The information flow control model analyzes the dynamic information flows caused by a sequence of queries. If a query can cause unauthorized indirect read or write to a view, the query should be denied. The use of the two models can improve the security of a database.

Index terms: database security, inference control, information flow control, access control.

1. Introduction

Database security problems arise from complex interactions between users and data in a database management system. In principle, conventional access control and authentication mechanisms can provide the penetration resistance necessary to *prevent* illegitimate access by unauthorized users [1]. However, inference and information flow problems cannot be prevented by neither conventional authentication nor access control since unauthorized access need not be attempted [2, 3]. Most access control mechanisms for databases state access permissions granted to users. When a query is submitted to the database systems, the permissions table is consulted to determine whether the query should be allowed. The restrictions for protecting a database system are generally stated in terms of database views, which can be used by a user to access a relation which he is not authorized to directly access. However, with information flow and inference, a user may acquire additional, unauthorized information, which may be the exact values of attribute or the relationship of attributes. Two inference techniques can be used to derive additional information: (1) analyzing functional dependencies between attributes within a relation or across relations, (2) merging views with the same constraints. On the other hand, information flows are caused by invoking a sequence of queries to indirectly read/write the sensitive data of a relation. The potential information flow paths in computer systems can be discovered by analyzing the static settings of system variables [4]. To discover the dynamic information flows, the actual operations invoked need be analyzed [5]. A sequence of seemingly innocuous operations may lead to indirect unauthorized information flows, which can be detected by after-the-fact audit analysis [3, 6]. However, to prevent the indirect unauthorized information flows, a new method for controlling information flow is desirable.

To demonstrate the inference of unauthorized information by merging views with the same constraints, we use the following database as an example. Assume a database with relation *Employee* = (*Name*, *Rank*, *Salary*, *Department*, *Manager*). An instance of the relation is shown in Figure 2-1.

Employee:

-
1. This work was supported in part by National Science Council, Taiwan, under contract NSC85-2622-E-009-006R
 2. The authors are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 30010, Tel: +886-3-5731876, Fax: +886-3-5724176, Email: ssp@csie.nctu.edu.tw

Name	Rank	Salary	Department	Manager
Andy	senior	43,000	strip	Cathy
Calvin	junior	35,000	strip	Cathy
Cathy	junior	48,000	strip	Cathy
Dennis	junior	38,000	panel	Herman
Herman	senior	55,000	panel	Herman
Ziggy	senior	67,000	panel	Herman

Figure 2–1. Database Instance

Assume there are two views, V_1 and V_2 , and the relationship of *Name* and *Salary* must be protected from unauthorized access.

```
create view  $V_1$  as
select Name, Department
from Employee
where Department = "strip"
```

```
create view  $V_2$  as
select Rank, Salary
from Employee
where Department = "strip"
```

Users can use V_1 and V_2 to access data from the *Employee* table. The retrieved data of the two views is in Figure 2–2.

Table 2: Retrieved Data by V_1

Name	Department
Andy	strip
Calvin	strip
Cathy	strip

Table 3: Retrieved Data by V_2

Rank	Salary
senior	43,000
junior	35,000
junior	48,000

Figure 2–2. Retrieved Data by V_1 and V_2

These two views are secure individually because users cannot infer the relation between *Name* and *Salary* by functional dependencies. However, if we consider the constraints of the two views, the relationship of attribute *Name* and *Salary* may not be secure. Since both views, V_1 and V_2 , have the same access constraints, it is possible to merge the two tables to derive the table in Figure 2–3.

Name	Rank	Salary	Department
Andy	senior	43,000	strip
Calvin	junior	35,000	strip
Cathy	junior	48,000	strip

Figure 2–3. Combine two tables with the same constraints

Consequently, users can infer the relationship of attribute *Name* and *Salary*. This violates the access control policy that the relationship of attributes *Name* and *Salary* must not be disclosed.

There are two types of information flows: indirect read and indirect write. Users may indirectly read an unauthorized relation through a sequence of access operations. For example, authorized users may read data from sensitive tables and then write to a table shared with unauthorized users. In this way, unauthorized users can access the sensitive data by reading data from the shared table. That is, unauthorized users can indirectly read sensitive data from authorized users. Users may also indirectly write to an unauthorized relation through a sequence of access operations. Unauthorized users may write data to shared tables and then authorized users read the data from

the shared tables and write into the sensitive tables. In this way, users can indirectly write data to unauthorized tables. Database security problems get even more complex when both inference and information flow techniques are used.

Most inference problems, based on the degree to which sensitive data may be inferred, fall into one of three distinct classes: (1) deductive inference, (2) abductive inference, and (3) probabilistic inference [7, 8]. deductive inference methods [9–15] makes use of functional dependencies property to find the inference problem in databases. Deductive inference can be used when a formal deductive proof of the unauthorized information can be derived from the authorized information that is directly available in the databases. Since deductive inference does not involve assumptions of facts, it must be based entirely on data found within the database. Graph based methods and theorem proof methods have been used to detect the deductive inference [8, 10, 12–17]. Abductive inference may be used when a formal deductive proof of sensitive data can be derived from the authorized information that is directly available in the databases or available by making assumptions of facts [8,12]. Since abductive inference can involve assumptions of facts, it can infer more unauthorized information within the database than deductive inference. The third type of inference is the probabilistic inference. It can be used when it is possible to estimate the likelihood of the truth of a sensitive data element based on likelihood known or computable nonsensitive data [8, 18–21].

The authorization scheme of System R [22] allows permissions to be granted to both relations and views. It is a flexible scheme that incorporates useful features, such as allowing users to grant other users permissions that were granted to them. The authorization scheme of INGRES is different. Queries are handled by a "query modification" algorithm [23 – 26]. Essentially, the algorithm searches for permitted views whose attributes contain the attributes addressed by the query, and the qualifications placed on these attributes in the views are then conjoined with the qualification specified in the query. The algorithm is attractive because when a query exceeds its permissions, it delivers the data that are within the permissions.

Authorization models using MAC (Mandatory Access Control), DAC (Discretionary Access Control) [2, 27–29] and Query Modification cannot completely prevent individuals from inferring unauthorized information, because unauthorized accesses need not be attempted. On the other hand, conventional inference control methods described above also cannot prevent (1) the inference problem arising from merging views with the same constraints to expose sensitive information, (2) the information flow problem caused by a sequence of authorized queries.

In order to prevent individuals from inferring unauthorized information, we will present two models that can determine the information that a user can infer from a given set of views, and the information that a user can acquire by indirectly reading and writing, respectively. The inference control model analyzes the protective restrictions (access permissions and integrity constraints). If unauthorized views can be inferred from a set of requested views, the request will be denied. The information flow control model analyzes the information flow caused by a sequence of queries. If a query can cause unauthorized indirect read or write to a relation, the query is rejected. The paper is organized as follows. Section 2 gives definitions of the symbols we will use in the paper. In sections 3 and 4, the inference control model and the information flow control model are presented, respectively.

2. Background

We assume the following definition of a relational database. A relation scheme R is a finite set of attributes A_1, \dots, A_m . With each attribute A_i , a set of values D_i called the domain of A_i is associated. Domains are non-empty, finite or countable infinite sets. A relation based on the relation scheme R is a subset of the product of the domains associated with the attributes of R . A database scheme R is a set of relation schemes R_1, \dots, R_n . A database instance D of the database scheme R is a set of relations $R_1(D), \dots, R_n(D)$, where each $R_i(D)$ is a relation based on the relation scheme R_i . A view V is an expression in the relation schemes of R that defines a new relation scheme, and for each database instance D defines a unique relation on the scheme denoted $V(D)$.

Database views may possess properties. A property is a proposition which is either true or false for any given view. A property p is inherited, if all views derived from views with property p , also have property p . An example of an inherited property is permission to access. If user U has

permission to access V_1, \dots, V_n , then U also has permission to access any views that can be derived from V_1, \dots, V_n .

Users can access databases by means of queries submitted by users. Each query specifies a view that is to be either retrieved or modified. Database security is enforced by granting users permission to retrieve or to modify specific views, and checking the granted permissions to prevent users from accessing unauthorized information. Only queries that are within the permissions granted to the users are allowed to be executed. Thus, there are views which users are granted to retrieve or modify, or which users are never granted to access. These protective restrictions are stated by associating users with the permissions of views:

- Retrieve permission (denoted by R): User U can retrieve view V , if U was granted permission to retrieve view V .
- Modify permission (denoted M): User U can modify view V , if U was granted permission to modify view V .

This information is stated in a table $VP = (View, Property)$ for each user, where view identifies a view definition, and property is either R or M or both. It is clear that R and M are inherited properties. Therefore, a user who was granted permission to retrieve or modify a set of views, also has permission to retrieve or modify any views derived from the granted views.

A view can be defined as:

$$V(Y_1, \dots, Y_k) \leftarrow R(X_1, \dots, X_n) \wedge C_{R.X_1} \wedge \dots \wedge C_{R.X_n}$$

- V : the view name
- $R(X_1, \dots, X_n)$: R is the relation name which the view accesses. X_1, \dots, X_n are the attributes of relation R .
- $R.X_i$: the i -th attribute of relation R .
- attribute Y_j : the j -th retrieved attribute, and $Y_j = R.X_i$.
- constraint $C_{R.X_i}$: a conditional search expression that identifies the attribute X_i of relation R to be retrieved. Its form is $R.X_i \text{ op } R.X_j$ or $R.X_i \text{ op } C$ or $C \text{ op } R.X_i$, where C is a constant and $op \in \{=, \neq, <, >, \geq, \leq\}$.

A view is a virtual table to users. User can specify queries on a view:

$$V'(Y_1, \dots, Y_k) \leftarrow V(Y_1, \dots, Y_k) \wedge C'_{Y_1} \wedge \dots \wedge C'_{Y_k}$$

In order to unify view representations, the V' is transformed to be the following:

$$V'(Y_1, \dots, Y_k) \leftarrow R(X_1, \dots, X_n) \wedge C''_{R.X_1} \wedge \dots \wedge C''_{R.X_n}$$

where for each i , $i = 1, \dots, n$

if $\exists j$ and $1 \leq j \leq k$, such that $Y_j = R.X_i$ then $C''_{R.X_i} = C_{R.X_i} \wedge C'_{Y_j}$

otherwise, $C''_{R.X_i} = C_{R.X_i}$

3. Inference Control Model

Users use views to access data. If a user U has permission to access V_1, \dots, V_n , then U also has permission to access any views that can be derived from V_1, \dots, V_n . We can find that all data that users can access are from views. In order to discover unauthorized information which users can infer, we must find all information which can be inferred from views and check what unauthorized information is inferred by users. There are two methods which can be used to infer information from views: (1) functional dependency, (2) merging views with the same constraints.

(A) Functional Dependency

A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation instance r of R . The constraint states that, for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; or alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. Let $XY \rightarrow Z$ denote $\{X, Y\} \rightarrow Z$. The following five inference rules are well known for functional dependencies:

1. (Reflexive rule) $\{X \supseteq Y\} \Rightarrow X \rightarrow Y$.

2. (Augmentation rule) $\{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ$.
3. (Transitive rule) $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z$.
4. (Decomposition rule) $\{X \rightarrow YZ\} \Rightarrow X \rightarrow Y$.
5. (Union rule) $\{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ$.

The reflexive rule says that a set of attributes always determines itself, which is obvious. The augmentation rule says that adding the same set of attributes to both the left- and right-hand sides of a dependency results in another valid dependency. According to the transitive rule, functional dependencies are transitive. The decomposition rule says that we can remove attributes from the right-hand side of a dependency; applying this rule repeatedly can decompose the functional dependency $X \rightarrow \{A_1, A_2, \dots, A_n\}$ into the set of dependencies $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$. The union rule allows us to do the opposite; we can combine a set of dependencies $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$ into the single functional dependency $X \rightarrow \{A_1, A_2, \dots, A_n\}$.

The reflexive rule, decomposition rule cannot be used to derive additional information from a granted set of views. However, with the augmentation rule, a augmented view V'_1 may be derived from a given pair of views, V_1 and V_2 . That is, if

$$\begin{aligned} &\{X_1, X_2, \dots, X_a\} \rightarrow \{Y_1, Y_2, \dots, Y_b\}; \\ &V_1(X_1, \dots, X_a, Z_1, \dots, Z_c) \leftarrow R(A_1, \dots, A_n) \wedge V_1 C_{R.A_1} \wedge \dots \wedge V_1 C_{R.A_n}; \\ &V_2(X_1, \dots, X_a, Y_1, \dots, Y_b) \leftarrow R(A_1, \dots, A_n) \wedge V_2 C_{R.A_1} \wedge \dots \wedge V_2 C_{R.A_n} \end{aligned}$$

then in the best case that there exists many-to-one mappings from X to Y (that is, in addition to functional dependency constraints there exists $t_1[X] \neq t_2[X]$ such that $t_1[Y] = t_2[Y]$), we can get

$$\begin{aligned} &V'_1(X_1, \dots, X_a, Y_1, \dots, Y_b, Z_1, \dots, Z_c) \leftarrow R(A_1, \dots, A_n) \wedge V'_1 C_{R.A_1} \wedge \dots \wedge V'_1 C_{R.A_n} \\ &\text{where } V'_1 C_{R.A_i} = V_1 C_{R.A_i} \wedge V_2 C_{R.A_i}, \text{ if } R.A_i = X_i \text{ or } R.A_i = Y_i; \\ &V'_1 C_{R.A_i} = V_1 C_{R.A_i}, \text{ otherwise.} \end{aligned}$$

The view-augmentation algorithm:

Initially, there are a given set of views VS and a set of functional dependencies FDS in the database.

1. Repeat the following until no view is inserted into VS
 - i. For each functional dependency $X \rightarrow Y$ in FDS that apply to the database do the following:
 - a. For each view V in VS :

If the set of attributes of V contains all attributes of X and Y then insert V into the set A

else if the set of attributes of V contains all attributes of X then insert V into the set B .
 - b. For each view V_i in B , do the following:

For each view V_j in A , with the augmentation rule an augmented view V'_i can be derived from the given pair of views, V_i and V_j ; insert V'_i into VS

For example: there are two views and $X_1 \rightarrow X_4, X_4 \rightarrow X_5$.

$$\begin{aligned} &V_1(X_1, X_4) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_1 \geq "a") \wedge (X_2 = "junior") \\ &V_2(X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_1 \leq "f") \wedge (X_4 = "strip") \end{aligned}$$

Considering $X_4 \rightarrow X_5$, we can derive a augmented view V' :

$$\begin{aligned} &V'(X_1, X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_1 \geq "a") \wedge (X_2 = "junior") \\ &\quad \wedge (X_4 = "strip") \end{aligned}$$

The transitive rule give the same results as the augmentation rule.

With the union rule, a combined view V'_1 may be derived from a given pair of views, V_1 and V_2 . That is, if

$$\begin{aligned} &\{X_1, X_2, \dots, X_a\} \rightarrow \{Y_1, Y_2, \dots, Y_b\}, \{X_1, X_2, \dots, X_a\} \rightarrow \{Z_1, Z_2, \dots, Z_c\}; \\ &V_1(X_1, \dots, X_a, Y_1, \dots, Y_b) \leftarrow R(A_1, \dots, A_n) \wedge V_1 C_{R.A_1} \wedge \dots \wedge V_1 C_{R.A_n}; \\ &V_2(X_1, \dots, X_a, Z_1, \dots, Z_c) \leftarrow R(A_1, \dots, A_n) \wedge V_2 C_{R.A_1} \wedge \dots \wedge V_2 C_{R.A_n} \end{aligned}$$

then in the best case that there exists many-to-one mappings from X to both Y and Z , we can get

$$\begin{aligned} &V'_1(X_1, \dots, X_a, Y_1, \dots, Y_b, Z_1, \dots, Z_c) \leftarrow R(A_1, \dots, A_n) \wedge V'_1 C_{R.A_1} \wedge \dots \wedge V'_1 C_{R.A_n} \\ &\text{where } V'_1 C_{R.A_i} = V_1 C_{R.A_i} \wedge V_2 C_{R.A_i}, \text{ if } R.A_i = X_i, R.A_i = Y_i \text{ or } R.A_i = Z_i; \\ &V'_1 C_{R.A_i} = V_1 C_{R.A_i}, \text{ otherwise.} \end{aligned}$$

A view–union rule algorithm:

There are a given set of views VS and a set of functional dependencies FDS .

1. Repeat the following until no view is inserted into VS
 - i. For each functional dependency $X \rightarrow Y$ in FDS that apply to the database do the following:
 - a. For each view V in VS :
If V has the same attributes for all of the attributes of X and Y then insert V into the set A else if there is a functional dependence $X \rightarrow Z$ and V has the same attributes for all of the attributes of X and Z then insert V into the set B .
 - b. For each view V_i in B , do the following:
For each view V_j in A , a unified view V' can be derived from a given pair of views, V_i and V_j , with the union rule and insert V' into VS .

For example: there are two views and $X_1 \rightarrow X_4, X_1 \rightarrow X_5$.

$$V_1(X_1, X_4) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_1 \geq "a") \wedge (X_2 = "junior")$$

$$V_2(X_1, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_1 \leq "f") \wedge (X_4 = "strip")$$

Considering $X_1 \rightarrow X_4$ and $X_1 \rightarrow X_5$, we can derive a augmented view V'_1 :

$$V'_1(X_1, X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge ("a" \leq X_1 \leq "f") \wedge (X_2 = "junior") \\ \wedge (X_4 = "strip")$$

(B) Merging Views with the Same Constraints

The functional dependencies can help users to find more relationships. Without functional dependencies, it is still possible to infer new relationships. Consider two views with the same constraints to retrieve data. Because their constraints are identical, the two views can retrieve the data of difference attributes of the same tuples in a relation. Therefore, we can directly combine the two retrieved data to derive a new view. A formal representation is given below.

If user U can access two views:

$$V_1(Y_{11}, \dots, Y_{1a}) \leftarrow R(X_1, \dots, X_n) \wedge V_1 C_{R.X_1} \wedge \dots \wedge V_1 C_{R.X_n}$$

$$V_2(Y_{21}, \dots, Y_{2b}) \leftarrow R(X_1, \dots, X_n) \wedge V_2 C_{R.X_1} \wedge \dots \wedge V_2 C_{R.X_n}$$

U can find two queries accessing the two views V'_1, V'_2 and:

$$V'_1(Y_{11}, \dots, Y_{1a}) \leftarrow V_1(Y_{11}, \dots, Y_{1a}) \wedge C'_{Y_{11}} \wedge \dots \wedge C'_{Y_{1a}}$$

$$\Rightarrow V'_1(Y_{11}, \dots, Y_{1a}) \leftarrow R(X_1, \dots, X_n) \wedge V''_1 C_{R.X_1} \wedge \dots \wedge V''_1 C_{R.X_n}$$

$$V'_2(Y_{21}, \dots, Y_{2b}) \leftarrow V_2(Y_{21}, \dots, Y_{2b}) \wedge C'_{Y_{21}} \wedge \dots \wedge C'_{Y_{2b}}$$

$$\Rightarrow V'_2(Y_{21}, \dots, Y_{2b}) \leftarrow R(X_1, \dots, X_n) \wedge V''_2 C_{R.X_1} \wedge \dots \wedge V''_2 C_{R.X_n}$$

and make their constraints equivalent, that is $V''_1 C_{R.X_k} = V''_2 C_{R.X_k}$ for $k = 1 \sim n$.

U can combine V'_1 and V'_2 to derive a new view :

$$V(Y_{11}, \dots, Y_{1a}, Y_{21}, \dots, Y_{2b}) \leftarrow R(X_1, \dots, X_n) \wedge C_{R.X_1} \wedge \dots \wedge C_{R.X_n}$$

$$\text{where } C_{R.X_k} = V''_1 C_{R.X_k} = V''_2 C_{R.X_k} \text{ for } k = 1 \sim n.$$

That is U can know relationships among attributes of V_1 and V_2 .

In the views–merging algorithm below, $V_1 \subseteq V_2$ denotes that the retrieved data of V_1 is the subset of the retrieved data of V_2 .

Views–Merging Algorithm:

There is a given set of views VS .

1. For each V_i in VS , insert V_i into NVS (New Views Set).
2. Repeat the following steps until the NVS is empty:
 - i. Take a view V_i from NVS . For each V_j saved in NVS , do the following:
If $V_j \subseteq V_i$ then delete V_j from NVS ; else if $V_i \subseteq V_j$ then delete V_i and goto step 2.
/* $V_j \subseteq V_i$ indicates the information in V_j is also in V_i , and therefore V_j need not be analyzed. The same reason is applied to $V_i \subseteq V_j$ */
 - ii. For each V_k saved in DVS , if the set of retrieved attributes of V_k is not the subset of retrieved attributes of V_i , do the following:
Derive a new view V by merging V_i and V_k with the same constraints and insert V into NVS .
 - iii. Delete V_i from NVS and insert DV_i into DVS .

As an example, consider that Smith are authorized to access views V_6 and V_7 of the database instance in Figure 2–1. V_6 gives the names of employees and their departments, and V_7 gives the ranks and salaries of employees who work at "strip". That is,

$$V_6(X_1, X_4) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5)$$

$$V_7(X_2, X_3) \leftarrow Employee(X_1, X_4, X_3, X_4, X_5) \wedge (X_4 = \text{"strip"})$$

Because $Employee.Rank$ does not functionally determine $Employee.Salary$, he cannot use functional dependency to derive a new view. Without functional dependency, he can modify V_6 and V_7 to the following:

$$V'_6(X_1, X_4) \leftarrow V_6(X_1, X_4) \wedge (X_4 = \text{"strip"})$$

$$\Rightarrow V'_6(X_1, X_4) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_4 = \text{"strip"})$$

$$V_7(X_2, X_3) \leftarrow Employee(X_1, X_4, X_3, X_4, X_5) \wedge (X_4 = \text{"strip"})$$

Because the constraints of V'_6 and V_7 are the same, he can derive a new view V such that:

$$V(X_1, X_2, X_3, X_4) \leftarrow Employee(X_1, X_4, X_3, X_4, X_5) \wedge (X_4 = \text{"strip"})$$

(C) The Inference Algorithm

Users can infer from a set of granted views. When a new view is granted to user U , the model uses inference control algorithm to infer relationships of the relation R which user U can access by the new view. The model uses the new view and granted views which user U has read permission to infer information. If there are unauthorized relationships inferred, U 's request for a new view must be rejected. Otherwise, U 's request for the new view is granted. After executing the algorithm, the model can find all relationships which user U can infer from views. In the inference algorithm below, let VS represent the set of views to be granted to a user, and DVS represent the set of inferred views.

The Inference Algorithm:

1. For each V_i in VS , insert V_i into NVS (New Views Set).
2. Repeat the following steps until the NVS is empty:
 - i. Take a view V_i from NVS . For each V_j saved in NVS , do the following:
 - If $V_j \subseteq V_i$ then delete V_j from NVS ; else if $V_i \subseteq V_j$ then delete V_i and goto step 2.
 - /* $V_j \subseteq V_i$ indicates the information in V_j is also in V_i , and therefore V_j need not be analyzed. The same reason is applied to $V_i \subseteq V_j$ */
 - ii. For each V_k saved in DVS :
 - If $V_i \subseteq V_k$ then delete V_i from NVS and goto step 2, else do the following:
 - A. Derive a new view V by merging V_i and V_k with the same constraints using view–merging algorithm, and then insert V into NVS .
 - B. If we cannot derive a new view at step a, do the following:
 - a. Find a functional dependency $X \rightarrow Y$ such that both V_i and V_k contains all attributes of X .
 - b. If both V_i and V_k contains all attributes of Y , then use the view–union rule to derive a new view V .
 - else if there is either V_i or V_k whose attributes contains all attributes of Y then use the view–augmentation rule to derive a new view V .
 - c. If $V \subseteq V_i$ or $V \subseteq V_k$ then don't insert V into NVS . Otherwise, insert V into NVS .
 - iii. Delete V_i from NVS and insert V_i into DVS .

As an example, consider the three views to be granted to user U :

$$V_1(X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_4 = \text{"panel"})$$

$$V_2(X_2, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_3 > 40,000)$$

$$V_3(X_2, X_3) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_5 = \text{"Herman"})$$

When these views are granted to user U , the view–inferring algorithm is used to find new relationships from the views of the relation $Employee$.

1. Initial status: insert views into NVS .
 - $DVS = \{\}$
 - $NVS = \{V_1, V_2, V_3\}$

2. Take V_1 from NVS . Because DVS is empty, we insert V_1 into DVS .
 $DVS = \{V_1\}$
 $NVS = \{V_2, V_3\}$
3. Take V_2 from NVS . We can use view–augmentation rule to derive a new view V_{21} using V_2 and V_1 .
 $DVS = \{V_1, V_2\}$
 $NVS = \{V_3, V_{21}\}$

$$V_{21}(X_2, X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_3 > 40,000) \wedge (X_4 = \text{"panel"})$$

4. Take V_3 from NVS . it is possible to merge views V_2 and V_3 with the same constraint to derive a new view V_{32} .
 $DVS = \{V_1, V_2, V_3\}$
 $NVS = \{V_{21}, V_{32}\}$

$$V_{32}(X_2, X_3, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_3 > 40,000) \wedge (X_5 = \text{"Cathy"})$$

5. Take V_{21} from NVS . We cannot derive any new view with V_{21} . We insert V_{21} into DVS .
 $DVS = \{V_1, V_2, V_3, V_{21}\}$
 $NVS = \{V_{32}\}$
6. Take V_{32} from NVS . We can use view–augmentation rule to derive V_{321} using V_1 and V_{23} , V_{3221} using V_{32} and V_{21} . We just insert into V_{32} DVS and V_{321} , V_{3221} into NVS .
 $DVS = \{V_1, V_2, V_3, V_{21}, V_{32}\}$
 $NVS = \{V_{321}, V_{3221}\}$
 $V_{321}(X_2, X_3, X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_3 > 40,000) \wedge (X_4 = \text{"panel"}) \wedge (X_5 = \text{"Cathy"})$
 $V_{3221}(X_2, X_3, X_4, X_5) \leftarrow Employee(X_1, X_2, X_3, X_4, X_5) \wedge (X_3 > 40,000) \wedge (X_4 = \text{"panel"}) \wedge (X_5 = \text{"Cathy"})$
7. Take V_{321} from NVS . We can find V_{321} is equal to V_{3221} in NVS . We can delete V_{321} from NVS . Take V_{3221} from NVS . We cannot derive any new view with V_{3221} . We insert V_{3221} into DVS .
 $DVS = \{V_1, V_2, V_3, V_{21}, V_{32}, V_{3221}\}$
 $NVS = \{\}$

The views saved in DVS are the views which U can infer from granted views.

4. Information Flow Control Model

It is possible to indirectly read/write an unauthorized relation through a sequence of access operations. This type of access problems can be solved by information flow control method proposed herein. Indirect read/write problems arise from interactions among users. These interactions include read (select) operation and write (insert) operation. The operation of inserting data retrieved from relation R_j into relation R_i can be denoted by:

$$R_i(Y_1, \dots, Y_a) \leftarrow R_j(X_1, \dots, X_n) \wedge C_{R_j.X_1} \wedge \dots \wedge C_{R_j.X_n}$$

There are two possible cases:

- (1) $\{R_i \cdot Y_1, \dots, R_i \cdot Y_a\} - \{R_j \cdot X_1, \dots, R_j \cdot X_n\} = \phi$ or
- (2) $\{R_i \cdot Y_1, \dots, R_i \cdot Y_a\} - \{R_j \cdot X_1, \dots, R_j \cdot X_n\} \neq \phi$.

In the latter case, the additional attributes of R_i may be filled with $NULL$ or some given value.

It is possible to disclose the sensitive information of a relation to an unauthorized user through a sequence of queries described in section 1. In this context, these seemingly normal queries represent unauthorized accesses and cannot be prevented by access control mechanisms.

Consider a relation $R_j(X_{j1}, \dots, X_{jm})$ that user U_1 is authorized to access, but user U_2 is unauthorized to access. Suppose U_1 submitted a query

$$R_i(X_{i1}, \dots, X_{in}) \leftarrow R_j(X_{j1}, \dots, X_{jm}) \wedge C_{R_j.X_{j1}} \wedge \dots \wedge C_{R_j.X_{jm}},$$

and then U_2 submits a query

$$V_k(Y_1, \dots, Y_a) \leftarrow R_i(X_{i1}, \dots, X_{in}) \wedge C_{R_i X_{i1}} \wedge \dots \wedge C_{R_i X_{in}}.$$

If $\{R_i \cdot X_{i1}, \dots, R_i \cdot X_{in}\} - \{R_j \cdot Y_{j1}, \dots, R_j \cdot X_{jm}\} = \phi$, then

$$V_k(Y_1, \dots, Y_a) \leftarrow R_j(X_{j1}, \dots, X_{jm}) \wedge C_{X_1} \wedge \dots \wedge C_{X_m}$$

where for *each* l , $l = 1, \dots, m$

if $\exists k$, $1 \leq k \leq n$, such that $R_j \cdot X_{jl} = R_i \cdot X_{ik}$ then $C_{X_l} = C_{R_i X_{ik}} \wedge C_{R_j X_{jl}}$

otherwise $C_{X_l} = C_{R_j X_{jl}}$

Else if $\{R_i \cdot X_{i1}, \dots, R_i \cdot X_{in}\} - \{R_j \cdot Y_{j1}, \dots, R_j \cdot X_{jm}\} \neq \phi$, then

$$V_k(Y_1, \dots, Y_a) \leftarrow R_j(X_{j1}, \dots, X_{jm}) \wedge C_{X_1} \wedge \dots \wedge C_{X_m}$$

where for *each* l , $l = 1, \dots, m$

if $\exists k$, $1 \leq k \leq n$, such that $R_j \cdot X_{jl} = R_i \cdot X_{ik}$ then $C_{X_l} = C_{R_i X_{ik}} \wedge C_{R_j X_{jl}}$

otherwise $C_{X_l} = C_{R_j X_{jl}}$

and for $R_i \cdot X_{ik} \in \{R_i \cdot X_{i1}, \dots, R_i \cdot X_{in}\} - \{R_j \cdot Y_{j1}, \dots, R_j \cdot X_{jm}\}$ then $C_{X_k} = C_{R_i X_{ik}}$

In the sequence of operations, U_2 can indirectly read the contents of R_j , which he is unauthorized to access. Thus, we have a indirect read violation.

A user may indirectly write the contents of a relation through a sequence of queries. Consider a relation $R_k(X_{j1}, \dots, X_{jm})$ that user U_2 is authorized to access, but user U_1 is unauthorized to access. If user U_1 submitted a query $R_i(X_{i1}, \dots, X_{in}) \leftarrow R_j(X_{j1}, \dots, X_{jm}) \wedge C_{R_j X_{j1}} \wedge \dots \wedge C_{R_j X_{jm}}$,

user U_2 submits a query $R_k(Y_1, \dots, Y_a) \leftarrow R_i(X_{i1}, \dots, X_{in}) \wedge C_{R_i X_{i1}} \wedge \dots \wedge C_{R_i X_{in}}$.

In the sequence of operations, U_1 can indirectly write the contents of R_k , which he is unauthorized to access. Thus, we have a indirect write violation.

The relations participated in the information flow form a vector which describes the information flow path. Analyzing the protective restrictions is able to find the potential information flow path. With a potential information path, a sequence of queries can be invoked to cause the actual flow of data that violates the flow control policy. Therefore, to ensure better security, the sequences of invoked queries (which represents the actual flow) must be analyzed.

If an individual can infer extra relationships of attributes which are not included in the relationships of attributes that he is authorized to access, the security policy, which is defined in terms of protective restrictions, is violated. Direct implementation of the protective restrictions as a two-dimensional structure is generally impractical due to the sparseness of the matrix. We can specify these relationships of attributes which can be known by authorized users in *Authorized Access Lists* (AAL) = $\{(US_1, A_1, V_1), (US_2, A_2, V_2), \dots\}$, where V_i is a view that must be protected, A_i is the set of access rights, and US_i (Users Set) is the set of users who have the set A_i of access rights to access V . There are two kinds of access rights *read*, *write*, that is, $A_i \subseteq \{read, write\}$. An authorized access list is associated with a view. If user U_i is authorized to read V_i , U_i can know any tuple and any relationship of the attributes in V_i . If a user can infer the data beyond the views he is authorized to access, a security violation occurs.

The query that user U_i reads view V_i causes not only new data flows into the query but also the change of access rights of the query for other views. In this context, the AAL associated with the view can propagate along with the data, and can affect the AALs of the view and the query that accept the data. Let $D[V_i]$, $read[V_i]$ and $write[V_i]$ represent the set of the users that have data flows into view V_i , (that is, indirectly write), the set of users that are authorized to read view V_i , and the set of users that are authorized to write view V_i . Similar definitions are applied to $D[Q_i]$ and $read[Q_i]$. The *read* operation of query Q_i on view V_i invoked by user U_i is represented by the command:

```

command action.read( $U_i, Q_i, V_i$ )
  if  $Q_i$  reads  $V_i$ 
  then  $D[Q_i] = D[Q_i] \cup D[V_i]$ , and

```

$$read[Q_i] = read[Q_i] \cap read[V_i]$$

end

In the command, $D[Q_i] = D[Q_i] \cup D[V_i]$ means that foreign data has flowed into the query, and $read[Q_i] = read[Q_i] \cap read[V_i]$ means that the data flowed into the query should have at least the same protection as the view from which the data was captured.

The access rights may also be changed if a query containing a *write* is invoked which causes direct and indirect flows of data into a view. The invocation of action *write* may cause both new information flows and access rights changed. For example, the action that user U invokes query Q to write view V not only causes a new information flow into the view but also changes the AAL of view V . This is represented by the command:

```
command action.write( $U_i, Q_i, V_i$ )
  if  $Q_i$  writes  $V_i$ 
  then    $D[V_i] = D[Q_i] \cup D[V_i] \cup \{U_i\}$ , and
          $read[V_i] = read[Q_i] \cap read[V_i]$ 
  end
```

In the command, $D[V_i] = D[Q_i] \cup D[V_i] \cup \{U_i\}$ means that foreign data has flowed into the view, and $read[V_i] = read[Q_i] \cap read[V_i]$ means that the data flowed into the view should have at least the same protection as the view from which the data was captured.

A flow control policy specifies the authorized flows of a database system. An access is permitted if the flows caused by the access are authorized. The flow policy is enforced by validating every user access for appropriate access rights. Every view has a monitor that validates all accesses to that view in the following manner.

1. User U_i invokes query Q_i to requests an access that causes α -flow to view V_i .
2. The protection system presents triplet (U_i, Q_i, α, V_i) to the monitor of V_i .
3. The monitor looks into the access rights of Q_i to V_i . There are two possible cases:
 - (1) $\alpha = write$
If $U' \in write[V_i]$ for all $U' \in D[Q_i]$, then the access is permitted, else it is denied.
 - (2) $\alpha = read$
If $U_i \in read[V_i]$, then the access is permitted, else it is denied.

5. Conclusions

Conventional authorization models cannot completely prevent individuals from deriving unauthorized information, because unauthorized accesses need not be attempted. In this paper, we propose two models for database security, which is able to protect the data and relationships of attributes in a secure database. The models investigate both the static and dynamic aspects of protection states by analyzing (1) functional dependencies between attributes within a view or across views, (2) views with the same constraints, and (3) information flows within a database. With the proposed models, the security of a database can be improved.

References

- [1] S.P. Shieh, W.H. Yang, "An Authentication and Key Distribution System for Open Network Systems," ACM Operating Systems Review, April 1996, pp 32–41.
- [2] S. Sanhu, "Lattice-Based Access Control Models," IEEE Computer, v26n11, Nov. 1993, pp. 9–19.
- [3] S.P. Shieh, V. D. Gligor, "On a Pattern-Oriented Model for Intrusion Detection," IEEE Transactions on Data and Knowledge Engineering, Dec. 1997, pp 661–668.
- [4] C. R. Tsai, V. D. Gligor and C. S. Chandrasekaran, "A Formal Method for the Identification of Covert Storage Channels in Source Code," IEEE Transactions. on Software Engineering, Vol. 16, No. 6, pp. 581–592, June 1991.
- [5] S.P. Shieh, V.D. Gligor, "Detecting Illicit Information Leakage in Operating Systems," Journal of Computer Security, Apr. 1997, pp 123–148.
- [6] S.P. Shieh, "A Distributed Intrusion Detection Model," Journal of Computers, Vol 6, No 4, December, 1994, pp 24–34.
- [7] T. D. Garvey, T. F. Lunt, and M. E. Stickel, "Abductive and Approximate Reasoning Model for Characterizing Inference Channels," in Proceedings of the Fourth Workshop on The Foundations of Computer Security, June 1991.

- [8] Thomas D. Garvey and Teresa F. Lunt, "Covert Stories for Database Security," Database Security, V: Status and Prospects, 1992, pp.363–381.
- [9] Leonard J. Binns, "Inference through Secondary Path Analysis," Database Security, VI: Status and Prospects, 1993, pp.195–209.
- [10] Leon J. Buczkowski, "Database Inference Controller," Database Security, III: Status and Prospects, 1990, pp.311–322.
- [12] Thomas D. Garvey, Teresa F. Lunt, Xiaolei Qian, and Mark E. Stickel, "Toward a Tool to Detect and Eliminate Inference Problems in the Design of Multilevel Databases," Database Security, VI: Status and prospects, 1993, pp.149–167.
- [13] Thomas H. Hinke, "Inference Aggregation Detection in Database Management Systems," Proceedings of IEEE 1988 Compute Society Symposium of Research in Security and Privacy, pp.96–106.
- [14] Thomas H. Hinke, "Database Inference Engine Design Approach," Database Security , II: Status and Prospects, 1989, pp.247–263.
- [15] Thomas H. Hinke, Harry S. Delugach and Asha Chandrasekhar, "Layered Knowledge Chunks for Database Inference," Database Security, VII:Status and Prospects, 1994, pp.275–295.
- [16] Bhavani Thuraisingham, "The Use of Conceptual Structures for Handling the Inference Problem, and Cover Stories for Database Security," In Proc. 5th IFIP WG 11.3 Working Conference on Database Security, November 1991.
- [17] H. Delugach, T. Hinke, "Wizard: A Database Inference Analysis and Detection System," IEEE Trans. on Knowledge and Data Engineering, v8, no1, February 1996, pp. 56–66.
- [18] Judea Pearl, "Fusion, Propagation, and Structuring in Bayesian Networks," Technical report CSD–850022, Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, June 1985.
- [19] E.H. Ruspini, "Epistemic logic, Probability, and the Calculus of Evidence," In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, Italy, 1987.
- [20] G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, New Jersey, 1976.
- [21] Lotfi A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," Fuzzy Sets and Systems, 1:3–28, 1978.
- [22] Griffiths P. and Wase B. W., "An Authorization Mechanism for Relational Database Systems," ACM Transactions on Databases Systems 1 (3), September 1976.
- [23] Stonebraker M. and Wong E., "Access control in relational database management system by query modification," in Proceedings of ACM Annual Conference, 1974, pp.180–186.
- [24] *SunINGRES Manual Set*, Sun Microsystems, Mountain View, California, Release 5.0. 1987.
- [25] Bob P. Weems, Wen–Gong Shieh, and Muhammad Jaseemuddin, "Complete Containment Sets and their Application to the Inference Problem," In Proceedings of Conference on Computer Assurance, June 1991, pp.187–200.
- [26] Amihai Motro, "A Unified Model for Security and Integrity in Relational Databases," Journal of Computer Security, 1994, pp.189–213.
- [27] E. Denning, *Cryptography and Data Security*, Addison Wesley, Treading, Massachusetts, 1982.
- [28] M. Gasser, "Building a Secure Computer System," Van Nostrand Reinhold, New York, 1988.
- [29] P. Samarati, E. Bertino, S. Jajodia, "An Authorization Model for a Distributed Hypertext System," IEEE Trans. on Knowledge and Data Engineering, v8, no4, August 1996, pp. 555–562.