# VALID: A Value-based Inference Detection Scheme[1]

**Raymond Yip, James Hoagland, Calvin Ko, Karl Levitt**

**Department of Computer Science**
**University of California, Davis**
**Davis, CA 95616**

{yip, hoagland, ko, levitt}@cs.ucdavis.edu

***Abstract***

*We describe a VALue-based Inference Detection Scheme (VALID) that detects logical inferences about sensitive associations in a database system. Previous approaches to this problem involved detecting possible inference paths at schema level and eliminating the paths by upgrading the sensitivity levels of some attributes. Our approach is different in that we attempt to precisely keep track of user knowledge on the protected associations at run-time using audit trails generated by the database system. The audit trails contains user queries and the return tables. Our approach avoids the overclassification problem and the false inference problem. In addition, it does not rely on functional dependencies among attributes that may not be readily available. We describe an inference detection algorithm, which uses an inference graph to model user knowledge about the protected associations. We have developed a prototype in Perl as a proof-of-concept, demonstrating that our scheme works efficiently in a database system of a moderate size. In the future, we plan to implement the scheme in a large database system to fully evaluate this approach.*

## 1.0 Introduction

In a multi-level database system, an association between two entities may be classified at a higher sensitivity level than that of the two entities in isolation. For example, in a personnel database, "*Name*", "*Address*", and "*Salary*" are *unclassified*, but "*Name*" and "*Salary*" taken together are *confidential*. To protect an association between entities, the database system should not only prohibit unauthorized users from retrieving the association directly, but also prevent them from inferring the association. An inference occurs when an adversary deduces information about an associations that has a higher sensitivity level than that of the adversary. The problem is resulted from the relationship among data in the database.

An adversary launches an inference attack by constructing an *inference path* that leads to the discovery of the protected association. An inference path is a sequence of attributes (possibly with composed attributes) that reveals the relationship between the first and the last attributes in the

1. This work is funded by the National Security Agency University Research Program under Contract No. DOD-MDA904-93-C4083.

sequence. An adversary can access an inference path when the least upper bound of the sensitivity levels of the attributes on the path is lower than that of the adversary.

Previous works on inference detection are mainly concerned with finding the inference paths using functional dependencies among the attributes at the database schema level [2, 4, 6, 8, 11, 13, 15, 16].[1] However, two attributes that are determined to be functionally independent at the schema level may be related to each other in an extension of the schema, and the dependencies found at the schema level may not truly reflect the relationship among data in the extension. For example, the relationship between "*Destination*" and "*Trip_Duration*" is in general a *n-m* mapping; that is, each destination has several possible trip durations (due to different routes), and each trip duration can be related to different destinations. It is possible to have only one type of warships going to a single destination, making the relationship between "Destination" and "*Trip_Duration*" *1-1* for this particular type of warship. An adversary can use this information to launch an inference attack.

An inference path can be eliminated by raising the sensitivity levels of some attributes on the inference path. However, upgrading an attribute may introduce new inference paths in the database [16]. Also, upgrading introduces the *overclassification problem* wherein an object is classified at a sensitivity level higher than its intended level [3]. Overclassification reduces the database utilization. Besides, upgrading is unnecessary when the inference path detected in the database schema simply does not appear in the extension of the database schema. This is termed as the *false inference* problem [8].

We developed a VALue-based Inference Detection scheme (VALID) that detects inferences about sensitive associations in a relational database system. Our scheme differs from other approaches in the following ways:

- Audit trail that consists user queries and their return tables are analyzed at run-time to detect inferences on protected associations.
- It is a purely value-based approach, which does not rely on functional dependencies among attributes.
- Our scheme does not require redesign of the database schema and upgrading of the sensitivity levels of attributes, thus avoiding the overclassification problem and the false inference problem.

Our scheme can detect three types of inferences (described in Section 2): direct, complementary, and supplementary inference. The scheme maintains an inference graph to represent user knowledge about the associations in the database using audit trails. The inference detection problem is then transformed into a graph traversal problem. An inference path is detected if there is a set of user queries that uniquely reveals a protected association. Prior knowledge on the database can be formulated into a user query and is used to initialize the inference graph. Our scheme can be used as a prevention scheme if the system rejects queries that lead to occurrences of inferences.

1. An attribute functionally determines another attribute if the relationship between them is either one-to-one (*1-1*) or one-to-many (*1-n*).

The remainding of the paper is organized as follows. In Section 2, we introduce the notations used in the paper, and describe the three types of inferences that our scheme can detect. We present the inference detection scheme in Section 3, and discuss the prototype we developed in Section 4, which is followed by a review of some related works, and our future work on the VALID scheme.

## 2.0 The Model and Types of Inferences

We assume the Database Management System (DBMS) employs the relational data model, though the detection scheme can be extended to database systems that employ other data models, such as hierarchical, network, and object-oriented data models. A relational database is a collection of relations. Each relation has a schema and an extension of the schema with tuples. The schema of a relation $R_i$ with $r$ attributes is denoted as $R_i(A_1, ..., A_r)$, where $A_j$ is an attribute of $R_i$, for $i \leq j \leq r$. An extension of a relation schema (or a table) is a set of tuples. Each tuple $t$ of an $r$-ary relation has *r attribute values*, $\{t(A_1), ..., t(A_r)\}$, where $t(A_j)$, $i \leq j \leq r$, is taken from the domain of $A_j$, $dom(A_j)$. In this paper, we address the issue of protecting the association between two attribute values, $t_p(A_1)$ and $t_q(A_n)$, where $t_p$ (or $t_q$) is a tuple in a table that has the attribute $A_1$ (or $A_n$). When $A_1$ and $A_n$ are in the same table, $t_p$ is equal to $t_q$.

A ship database is employed to illustrate the inference detection scheme. The database has two tables: the *Ship* table and the *Destination* table. Table 1 shows the ship table which has three *unclassified* attributes, "*Ship_ID*", "*Sailing_Date*", and "*Duration*", and an attribute *"Destination"* that has different classification for different attribute values. Table 2 shows the *Dest_Info* table which has three *unclassified* attributes, "*Destination*", "*Sailing_Date*" and "*Duration*". The association between the ship "*WS5678*" and its destination of "*Country X*" is *classified*.

| Ship_ID [U] | Sailing_Date [U] | Duration [U] | Destination |
|---|---|---|---|
| WS1234 | 5/15/95 | 20 | UK [U] |
| WS5678 | 5/15/95 | 25 | Country X [C] |
| WS9012 | 5/20/95 | 20 | Amsterdam [U] |

**TABLE 1. The Ship Table in the Sample Database**

| Destination[U] | Sailing_Date [U] | Duration[U] |
|---|---|---|
| UK | 5/15/95 | 20 |
| Country X | 5/15/95 | 25 |
| Amsterdam | 5/20/95 | 20 |

**TABLE 2. The Dest_Info Table in the Sample Database**

Suppose semantically "*Ship_ID*" functionally determines "*Sailing_Date*", "*Duration*", and "*Destination*". Figure 1 shows the conceptual graph of the sample database schema as used in conventional inference analysis methods. It shows that there is no secondary inference path from "*Ship_ID*" to "*Destination*" besides the primary one in the *Ship* table; hence, conventional methods will conclude that there is no inference path in the database schema. We will show that inferences can indeed occur in the sample database.

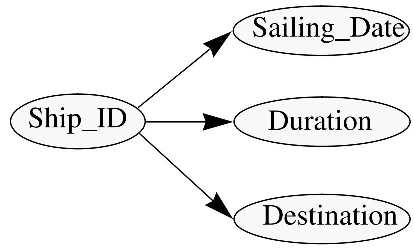


**FIGURE 1. The Conceptual Graph of the Sample Database**

To simplify the discussion, we consider user queries of the following form:

***Select*** $A_i, ..., A_j$
***From*** $T_m, ..., T_n$
***Where*** $C(A_k, ..., A_l)$,

where $A_i, ..., A_j$ in the *select*-clause are the attributes projected by the query from the tables, $T_m, ..., T_n$. $C(A_k, ..., A_l)$ in the *where*-clause is a conditional expression that specifies the condition that each of the return tuples of the query must satisfy. Notice that $\{A_i, ..., A_j\}$ and $\{A_k, ..., A_l\}$ need not have any set-subset relationship.

We observe three types of inferences that can happen in a database system: *direct inference, complementary inference, and supplementary inference*. A direct inference results from accessing a set of attribute values that uniquely determines the protected association. For example, Query 1 and 2 constitute a direct inference.

*Query 1.1:* ***Select*** *Ship_ID, Duration*
***From*** *Ship*,
*Query 1.2:* ***Select*** *Destination*
***From*** *Dest_Info*
***Where*** *Duration = 25.*

By knowing that only the ship "*WS5678*" has a trip duration of "*25*", and that "*Country X*" is the only destination with trip duration of "*25*", the protected association is inferred.

*Complementary inference* is undertaken by collecting a set of possible associations, and eliminating those impossible associations to infer the protected association. For example, the adversary first finds out that ship "*WS5678*" goes to either "*UK*" or "*Country X*" using Query 2.1 and Query 2.2.

*Query 2.1:* ***Select*** *Sailing_Date*
***From*** *Ship*
***Where*** *Ship_ID = "WS5678",*
*Query 2.2:* ***Select*** *Destination*
***From*** *Dest_Info*
***Where*** *Sailing_Date = "5/15/95".*

The adversary can use Query 2.3 to discover that ship "*WS1234*" goes to "*UK*" (an unprotected association), and the protected association can be inferred.

*Query 2.3:* ***Select*** *Ship_ID*
***From*** *Ship*
***Where*** *Destination = "UK".*

The collection and the elimination step need not be performed in any particular order.

*Supplementary inference* is used to incrementally obtain pieces of information instead of retrieving them all at once. For example, query 1.1 can be broken into two separate queries as follows,

*Query 3.1:* ***Select*** *Ship_ID, Duration*
***From*** *Ship*
***Where*** *Duration < 22,*
*Query 3.2:* ***Select*** *Ship_ID, Duration*
***From*** *Ship*
***Where*** *Duration >= 22.*

Supplementary inference can be used with complementary attack to hide user activity. Our detection scheme is designed to properly handle these three types of inferences

## 3.0 The Inference Detection Scheme

Figure 2 shows the system architecture of our inference detection system. Users issue queries to the DBMS, which processes them and returns the query results. The DBMS enforces the security policy by rejecting queries that directly retrieve the protected association. In addition, the DBMS audits all user queries and return tables into an audit log. The detection system maintains an inference graph representing user knowledge about the associations in the database. The inference graph is constructed from the audit log and the database schema. The Inference Detection module is invoked whenever the graph is updated to determine whether an inference has occurred. The scheme consists of two parts: the construction of the inference graph, and the detection of inference paths in the graph. We discuss these two parts in the following subsections. Details of the algorithm are shown in the Appendix.

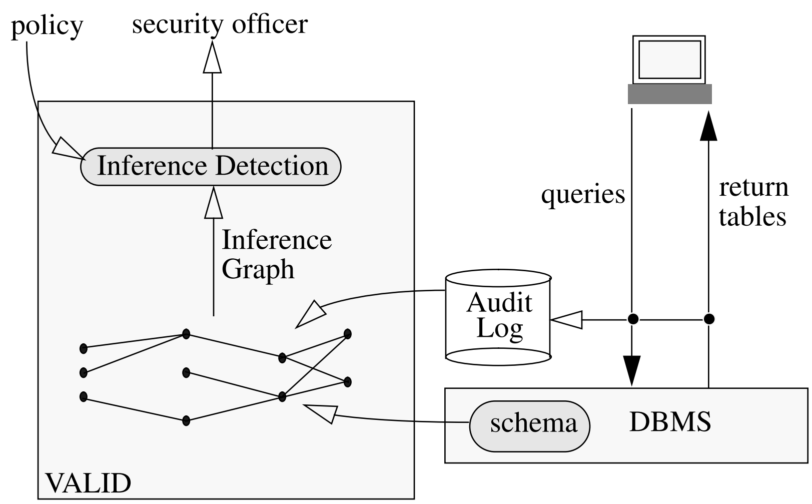


**FIGURE 2. The Overall System Architecture**

### 3.1 Inference Graph Construction

We define the inference graph as follows:

*Definition 3.1.1: An Inference Graph is an undirected graph G = (V, E), where*

1. $V = \{<A_i, a_i>: A_i$ *is an attribute in the database, and* $a_i$ *is an attribute value of* $A_i\}$, *and*
2. $E = \{(V_1, ..., V_m)$: $2 \leq m$, *and* $V_1, ..., V_m$ *are in V}.*

Each vertex in *V* corresponds to a distinct attribute value of an attribute in the database. An edge along a set of vertices describes an entity in the database. For example, an edge along the four attribute values of the "*WS5678*" tuple in the *Ship* table is *(<Ship_ID, WS5678>, <Sailing_Date, 5/15/95>, <Duration, 25>, <Destination, Country X>)*. We say that the edge *E* is a *simple edge* when $m = 2$; otherwise, we say that the edge *E crosses over* the vertices $V_1, ..., V_m$. Inference through composed attributes is represented by a cross over edge along the involved vertices.

Two attributes appearing in two different tables may be semantically equivalent. For example, the primary key in one table may appear as a foreign key in another table. These semantically equivalent attributes having the same domain are combined as a single attribute in *G*. Also, two attributes whose domains have a set-subset relationship may be combined into a single attribute. For example, when the domain of "*Manager_ID*" is a subset of the domain of "*Employee_ID*", an adversary may learn about the information about a manager by forming an inference path through either one of these two attributes. The combination of attributes are done regardless of the sensitivity levels of the attributes. For example, the "*Destination*" in both of the *Ship* and *Dest_Info* tables are combined as a single attribute even though the two attributes are classified at different sensitivity levels.

Our scheme maintains an inference graph for each user. Initially, the inference graph of a user is empty, indicating that the user has no prior knowledge about the database. As the user makes queries to the DBMS, vertices and edges are added to the graph based on the queries and the return tables. To detect collaborating inference attacks, the inference graphs of the collaborating users are unioned to obtain the combined user knowledge. We maintain a function *Query(E)* that maps from an edge *E* to the set of queries that generate the edge. This is useful in carrying out alternative elimination, and reconstructing the set of queries that constitute an inference.[1]

We denote *S* as the set of the attributes appearing in the *select*-clause of the user query, and *W* as those appearing in the *where*-clause, and *T* as the return table. Three cases are considered.

$(\boldsymbol{W \subseteq S}) \wedge$ ***T is not empty*** . In this case, all the information about the query can be found in the return table. Suppose the query projects the set of attributes $\{A_k, ..., A_l\}$. Vertices corresponding to attribute values in the return table are added to *G* if they have not already been in *G*. A vertex in *G* indicates that the user knows the existence of the corresponding attribute value in the database. For each tuple *t* in the return table, $t(A_k)$, ..., $t(A_l)$ describe the same entity; hence, an edge crosses over $\langle A_k, t(A_k)\rangle$, ..., $\langle A_l, t(A_l)\rangle$ is added to *G* if it does not exist. *Query()* is updated when the new edge(s) is being added.

$(\boldsymbol{W - S \neq \emptyset}) \wedge$ ***T is not empty*** . In this case, there are some attributes that appear in the *where*-clause but not in the *select*-clause. Such attributes may provide information about the associations. For example, suppose the following query,

| | |
|---|---|
| ***Select*** | $A_2, A_3$ |
| ***From*** | $T$ |
| ***Where*** | $A_1 > 10,$ |

returns the tuple *(x, y)*. From the conditional expression of the query, the user is able to know that *(x, y)* is associated with only those attribute values of $A_1$ that is greater than 10. Such knowledge may lead to the identification of an inference path (to be discussed in Section 3.2). This information is maintained in the *Query()* function. A special case of this category of queries is when the

1. We assume that the return table is not displayed in the order of the record numbers, and hence the adversary cannot use the order of the tuples to deduce the association between tuples from different return tables.

attribute, say $A_i$, that appears only in the *where*-clause is of the form, "$A_i = a_i$ ***AND*** $C_1$", where $a_i$ is an attribute value of $A_i$, and $C_1$ is the rest of the conditional expression. For example, consider this query,

> ***Select*** $A_2, A_3$
> ***From*** $T$
> ***Where*** $A_1 = 5$ ***AND*** $A_2 > 30.$

Even though $A_1$ does not appear in the return table, the user knows that each of the return tuples is associated with the value *"5"* of $A_1$. Hence, an edge is added from $<A_1, 5>$ to each of the edges generated by this query.

***T is empty*** . Empty return table also allows an adversary to gain knowledge about the associations among the attribute values as no tuple satisfies the conditional expression. For example, an empty return table to this query,

> ***Select*** $A_1, A_2$
> ***From*** $T$
> ***Where*** $A_1 = a_i$ ***AND*** $A_2 = b_j,$

implies that there is no association between $a_i$ of $A_1$ and $b_j$ of $A_2$. This information is maintained in a list of queries that return an empty table. The list is indexed by the attributes appearing in the *where*-clause of the queries, and is looked up to determine an association between attributes is possible.

Notice that an edge crossing over $m$ vertices is not equivalent to $m$ edges that connects the $m$ vertices. For example, suppose there are three attribute values, $x$, $y$, and $z$ from three different attributes of a relation. In Figure 3 (a), the three vertices are connected by three simple edges in an inference graph. The edge between $x$ and $y$ can be generated from the tuple $(x, y, z')$, the edge between $y$ and $z$ from the tuple $(x', y, z)$, and the edge between $x$ and $z$ from the tuple $(x, y', z)$, for some $x' <> x$, $y' <> y$, *and* $z' <> z$. Hence, the tuple $(x, y, z)$ may not necessary appear in the database. In Figure 3 (b), there is a cross over edge along $x$, $y$, and $z$, and we can conclude from the graph that $x$, $y$, and $z$ must describe the same entity.

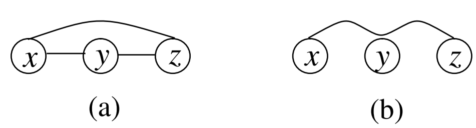


**FIGURE 3. An Example of a Cross Over Edge.**

### 3.2 Inference Detection

An inference about the association between $t_p(A_1)$ and $t_p(A_n)$ occurs if there is an inference path between vertices corresponding to $t_p(A_1)$ and $t_p(A_n)$ in the inference graph; that is the adversary can use legitimate queries to reveal the protected association. The inference detection problem becomes a path discovering problem in the inference graph. A path between two vertices is a set of edges that connect the two vertices, possibly with overlapping vertices.

We define the notion of *alternative* on a vertex and *unique path* as follows:

*Definition 3.2.1: A vertex* $\langle A_i, a_i \rangle$ *has an alternative if there exists an attribute* $A_j$*, and at least two attribute values of* $A_j$*, say* $a_{j1}$ *and* $a_{j2}$*, such that there are edges connecting vertex* $\langle A_i, a_i \rangle$ *to both vertices* $\langle A_j, a_{j1} \rangle$ *and* $\langle A_j, a_{j2} \rangle$*.*

*Definition 3.2.2: A path is unique if it has no alternative on every vertex on the path.*

A vertex has an alternative when there are two or more possible paths going through the vertex. The existence of a unique path implies that an inference has occurred.

There are three ways for an adversary to eliminate alternatives on a path to obtain the unique path: unique path propagation, conditional expression, and empty return table. We discuss the alternative elimination methods as follows.

**Elimination by Unique Path Propagation**. A unique discovered path may lead to the elimination of an alternative at a vertex that is connected to the unique path. Consider an inference graph in Figure 4,

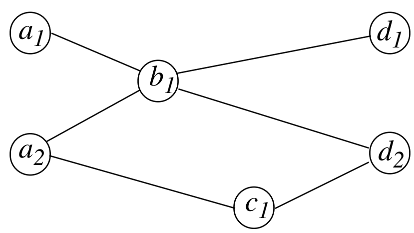


**FIGURE 4. An Example on Alternative Elimination**

where $a_1$ and $a_2$ are vertices corresponding to attribute values in attribute $A_1$, $b_1$ is a vertex for an attribute value in $A_2$, and so on. Suppose the association between $a_1$ and $d_1$ is protected. There is an alternative at $b_1$, and both $a_1$ and $a_2$ can be associated with either $d_1$ or $d_2$. Now, with the fact that $a_2$ and $d_2$ is associated through vertex $c_1$, the user can eliminate the possibility that vertex $a_1$ is associated with $d_2$, and obtain a unique path from $a_1$ to $d_1$.

**Elimination by Conditional Expression**. Conditional expression elimination employs the conditional expression as specified in the queries to determine if the vertices connected by two edges are associated. Consider the inference graph shown in Figure 4. Suppose the edge between $b_1$ and $d_2$ is added in the graph by the following query,

| | |
|---|---|
| ***Select*** | $A_2, A_4$ |
| ***From*** | $T$ |
| ***Where*** | $A_1 > a_1,$ |

where $a_1$ and $a_2$ are numerical values and $a_2 > a_1$. By substituting the values of $a_1$ and $a_2$ into "$A_1 > a_1$", the adversary can eliminate the possibility that $a_1$ is associated with $d_2$.

**Elimination by Empty Return Table**. This technique uses the fact that no tuple satisfies the conditional expression of the query that produces an empty return table. Again, consider the inference graph in Figure 4. Suppose the adversary has issued this query,

***Select*** $A_1$, $A_2$, $A_4$
***From*** $T$
***Where*** $A_1 = a_2$ ***AND*** $A_2 = b_1$ ***AND*** $A_4 = d_1$,

which returns an empty table. Then the user can infer that it is impossible to have $a_2$ associated with $d_1$, and hence obtains the unique path.

The three elimination techniques described can be used to detect the complementary inference. Direct inference is detected simply by looking for a unique path in the graph. Supplementary inference is handled by accumulating user knowledge in a single graph.

### 3.3 Course of Action After An Inference is Detected

We introduce two notions: *weak inference* and *strong inference*. A "conclusion" about the unique path obtained from a weak inference may change as the adversary gains more information about the database. It is possible for an adversay to use inadequate information to "infer" the protected association.

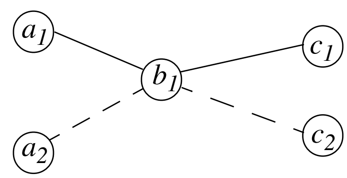


**FIGURE 5. An Example of a Weak Inference**

In the inference graph as shown in Figure 5, the solid lines indicate the information known to the adversary, and the dotted lines indicate the associations existing in the database but not yet known to the adversary. Without knowing the association among $a_2$, $b_1$, and $c_2$, the adversary infers that $a_1$ is associated to $c_1$. Such a conclusion may change when the adversary finds out more about the associations of $b_1$. A weak inference occurs if there exists at least a vertex on the inference path such that the adversary does not know about all associations connected to the vertex.

Strong inference occurs when the user can ensure that any additional information will not change the inferred conclusion. This happens when each vertex on the inference path has edges connected to each instance of the attribute value. The security officer determines whether the occurrence of a weak inference has revealed the protected information to the users. Also, it should be noticed that users may not be aware of the existence of inference paths. The security office may investigate the query history to gain some clues about whether a user is fishing protected information or simply doing his/her job.

### 3.4 An Example

We illustrate the inference detection scheme by showing how an attack using the complementary inference scheme (as described in Section 2) can be detected. Figure 6(a) shows the inference graph after Query 2.1 returns a table that indicates "*WS5678*" has the sailing date of "*5/15/95*". Query 2.2 tells that there are two destinations with the sailing date of "*5/15/95*", which are "*UK*", and "*Country X*". Hence, two new edges are added as shown in Figure 6(b). We say that there is an alternative at the vertex corresponding to "*5/15/95*", as it connects to two attribute values of "*Destination*". This alternative is eliminated by using Query 2.3 to find out that "*UK"* is the destination of "*WS1234*". The final graph is shown in Figure 6(c). Using the unique path propagation, an inference path is detected (bold lines).

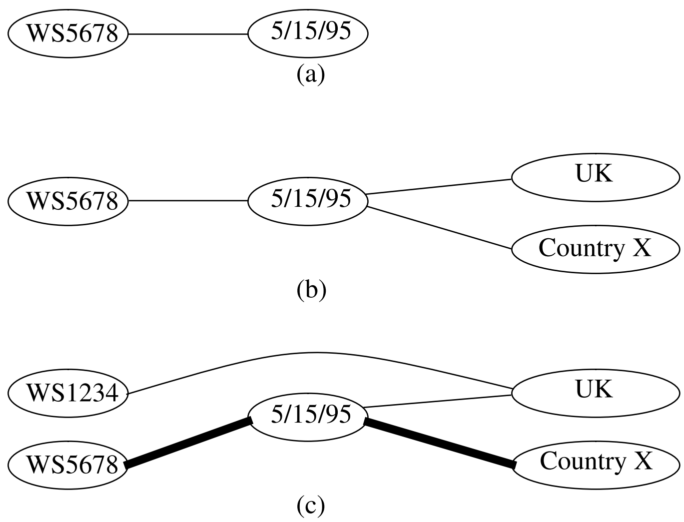


**FIGURE 6. An Example on Detecting Complementary Inference**

## 4.0 The VALID Prototype

We have developed a prototype of the VALID scheme using Perl [19]. We use the *dot* system [12] to graphically display inferences path detected. A sample relational database and user queries are randomly generated to experiment with the scheme. The prototype successfully detects direct inference and complementary inference using the unique path propagation technique. Figure 7 shows a sample screen display of the prototype. The number on an edge identifies the query that generates the edge. The sample database has five attributes: "*Ship_Name*", "*Crew_Size*", "*Maximum_Warp*", "*Commander*", and "*Destination*". The association between "*Hood*" and "*Riesa*" is sensitive, while the individual attribute values are not classified. For clarity of presentation, vertices and edges that are not connected to the inference path are omitted. Before query 40 is processed, there is an alternative at vertex "243". "*Hood*" can be assoicated with either "*Pacifica*" or “*Riesa*”. After query 40 is processed, there is a unique path going from “*Stargazer*” to “*Pacifica*”. This information is used to infer the sensitive association.

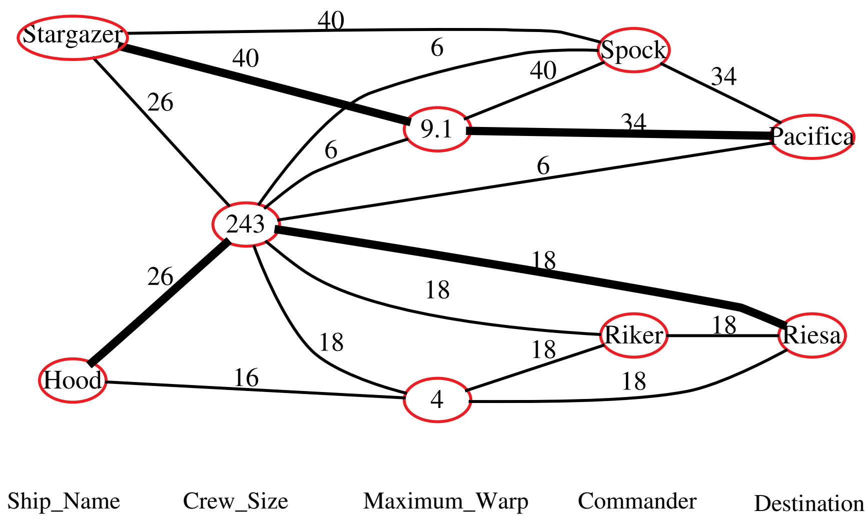


**FIGURE 7. A Sample Screen Display of the Prototype**

## 5.0 Related Work

Lunt [13] points out that some inference problem can be avoided by redesigning the database schema, and classifying the attributes correctly. However, the security officer may not be aware of all inferences that can occur in a database system. Our work attempts to automate the detection process without relying on human expertise.

Hinke [8, 9] uses a semantic relationship graph to represent the dependencies among attributes in the database schema. An inference occurs when there is an alternative path (or a secondary path as termed in [2]) between two attributes. The attribute information may be either stored in the database, or simply known to the users. The false inference problem is observed, which occurs when the inference path does not exist in an extension of the database schema. Run-time queries are periodically issued to determine if inferences exist in the database. Our scheme tackles this problem by finding the information flow between attribute values in an extension of the database schema.

The DISSECT [15, 6] is a tool developed at SRI which can detect and eliminate inference paths. The database schema is represented by a graph where nodes are the attributes, with possibly composed attributes, and edges are dependencies that involve either primary keys or foreign keys. The attributes are classified in accordance to two security properties [5]: key security property (non-key attribute dominates primary key) and referential security property (foreign key dominates the primary key that they refers). Our scheme does not assume that key attributes must appear in an inference path.

Thuraisingham [18] uses a multilevel semantic net to represent the relationships specified in the database, and two generalization relationships, “ISA” (is a) and “AKO” (a kind of), which are used in semantic data modeling. Rules are developed to classify attributes under various relationships. Inference is detected by applying three deduction rules on the semantic net: instantiation, joining two graphs, and removing duplications.

The AERIE project [11, 10] aims at developing a conceptual graph based inference analysis system. Four layers are discussed: part-of, used-for, functional dependency, and temporal layers. These layers are applied whenever it is deemed necessary.

Marks et. al. [14] developed a hypersemantic data model that generalizes various inference analysis schemes using various knowledge representations. The security officer only needs to learn this unified data model. Transformations are provided to transform the unified model into the specific data models. The detection scheme provided by the specific data model can be used to detect inference. They also identified three types of inferences relating to aggregation and generalization.

An inference path can be blocked by raising the sensitivity level of a portion of attributes on the path. Su et. al. [16, 17] describe an algorithm to block inference using functional dependencies and multivalued functional dependencies. Polyinstantiation [5, 7] can be used to eliminate an inference channel. Binns [1] illustrates that polyinstantiation may introduce inference when the database is not carefully designed. Binns [3] identifies the overclassification problem when upgrading is used to block inference paths and proposes the use of both intended and actual classification to avoid blocking inference path that is not inherently classified.

## 6.0 Conclusion and Future Work

We described an inference detection scheme that detects inferences of associations among attributes in a relational database system. The attributes involved in the protected association can be classified at a sensitivity level that is lower than that of the protected association. This provides a higher utilization of the database, and supports the distinction between protecting an association and protecting the attributes in an association. Most existing inference detection schemes rely on functional dependencies among attributes to detect inference paths, limiting such scheme to detect inferences that occur only at the database schema level. On the other hand, our scheme is based on user knowledge about the associations, and is able to detect inference in the extension of the schema. Conventional inference detection systems eliminate an inference channel by upgrading attribute(s) on the inference path which causes the overclassification problem, while our scheme does not need any upgrading. The scheme is conservative as the user may not be aware of the existence of the inference path detected.

Our work can be extended in several directions. One direction is to incorporate knowledge that is not stored in the database into the inference graph. Prior knowledge can be expressed as edges in the initial inference graph. For example, if it is a publicly known information about the possible destinations given a trip duration attributes, we can add edges between them in the initial graph to indicate that the user does not need to use query to gain such knowledge. Although functional dependencies are not used in our scheme, such knowledge is a useful heuristic for guiding our inference detection process. For example, we can first detect paths along functionally related attributes, which may provide a rich source of inference paths.

Our scheme only considers the associations across attributes, but not the relationship among the tuples. For example, this query,

| | |
|---|---|
| ***Select*** | *salary* |
| ***From*** | *T* |
| ***Where*** | *T.salary > T.salary,* |

reveals the tuple with the highest salary. When it is known that the highest salary is related to, say, the manager, we can add an edge between the manager and the salary. We may also investigate the relationship among attributes. For example, suppose the duration is equal to the difference between the sailing date and the arrival date. By knowing the sailing date and the arrival date of a warship, the user can deduce the duration of the trip. Integrity constraints in the database may provide information on the relationships among attributes.

This is the first attempt to tackle the inference problem by monitoring user queries and return tables. The scheme we proposed is not necessary complete, but it starts out a new approach to the inference problem.

## Acknowledgments

We would like to express our gratitude to Becky Base and Donald Marks of NSA for their valuable comments on our work.

## References

[1] Binns, Leonard J., "Inference and Cover Stories," *Proc. 6th IFIP WG11.3 Workshop on Database Security*, August, 1992, pp. 169-178.

[2] Binns, Leonard J., "Inference Through Secondary Path Analysis," *Proc. 6th IFIP WG11.3 Workshop on Database Security*, August, 1992, pp. 195-209.

[3] Binns, Leonard J., "Implementation Considerations for Inference Detection: Intended vs. Actual Classification," *Proc. 7th IFIP WG11.3 Workshop on Database Security*, September, 1993, pp. 297-306.

[4] Buczkowski, Leon J., "Database Inference Controller," *Proc. 5th IFIP WG11.3 Workshop on Database Security*, November, 1991, pp. 311-322.

[5] Denning, Dorothy E., Teresa F. Lunt, Roger R. Schell, Mark Heckman, William Shockley, "A Multilevel Relational Data Model," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April 1987, pp. 220-234.

[6] Garvey, Thomas D., Teresa F. Lunt, Xiaolei Qian, and Mark E. Stickel, "Toward a Tool to Detect and Eliminate Inference Problems in the Design of Multilevel Database," *Proc. 6th IFIP WG11.3 Workshop on Database Security*, August, 1992, pp. 149-167.

[7] Garvey, Thomas D. and Teresa F. Lunt, "Cover Stories for Database Security", *Proc. 5th IFIP WG11.3 Workshop on Database Security*, November, 1991, pp. 363-380.

[8] Hinke, Thomas H., "Inference Aggregation Detection in Database Management Systems", *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988, pp. 96-106.

[9] Hinke, Thomas H., "Database Inference Engine Design Approach", *Proc. 2nd IFIP WG11.3 Workshop on Database Security*, October, 1988, pp. 247-262.

[10] Hinke, Thomas H., and Harry S. Delugach, “AERIE: An Inference Modeling and Detection Approach for Databases,” *Proc. 6th IFIP WG11.3 Workshop on Database Security*, August, 1992, pp. 179-193.

[11] Hinke, Thomas H., Harry S. Delugach and Asha Chandrasekhar, “Layered Knowledge Chunks for Database Inference,” *Proc. 7th IFIP WG11.3 Workshop on Database Security*, September, 1993, pp. 275-295.

[12] Koutsofios, Eleftherios, and Stephen C. North, “Drawing Graphs with dot”, *dot User’s Manual*, AT&T Bell Laboratories, October, 1993.

[13] Lunt, Teresa F., “Aggregation and Inference: Facts and Fallacies,” *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, May 1989, pp. 102-109.

[14] Marks, Donald G., Leonard J. Binns, and Bhavani M. Thuraisingham, “Hypersemantic Data Modeling for Inference Analysis,” *Proceedings of the IFIP WG11.3 Working Conference on Database Security*, Germany, 23-26 August, 1994, pp. 157-180.

[15] Qian, Xiaolei, Mark E. Stickel, Peter D. Karp, Teresa F. Lunt, Thmoas D. Garvey, “Detection and Elimination of Inference Channels in Multilevel Relational Database Systems,” *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy*, May 1993, pp. 196-205.

[16] Su, Tzong-An and Gultekin Ozsoyoglu, “Data Dependencies and Inference Control in Multilevel Relational Database Systems,” *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April 1987, pp. 202-211.

[17] Su, Tzong-An and Gultekin Ozsoyoglu, “Multivalued Dependency Inferences in Multilevel Relational Database System,” *Proc. 3rd IFIP WG11.3 Workshop on Database Security*, September, 1989, pp. 293-300.

[18] Thuraisingham, Bhavani, “the Use of Conceptual Structures for Handling the Inference Problem,” *Proc. 5th IFIP WG11.3 Workshop on Database Security*, November, 1991, pp. 333-362.

[19] Wall, L., and R. Schwartz, *Programming Perl*, O’Reilly & Associates (pub), 1991.

## Appendix

The inference detection algorithm is shown in Figure 8. The *Graph-Update* procedure is invoked when a new query, $Q$, is issued and a return table, $T$, is audited. It is followed by the *Path-Finding* procedure which detects if any inference occurs resulting from the new query and return table.

*Input*: the policy is to protect the association between $t_p(A_1)$ and $t_q(A_n)$, user query $Q$, and return table $T$.
*Output*: return whether an inference is detected, and if it is, whether it is a strong or weak inference.
*Initialization*: $G$ is an empty graph.

**Graph-Update**($G$, $Q$, $T$)
    **for each** tuple $t$ in $T(A_i, ..., A_j)$ **do**
        add vertices $<A_i, t(A_i)>$ to $G$;
        add an edge $E$ through the vertices of $<A_i, t(A_i)>, ..., <A_j, t(A_j)>$ to $G$;
        **if** $A_i$ only appears in the where-clause with the form "$A_i = a_i$ *AND C*" where $C$ is the rest of the conditional expression **then**
            add edges from $<A_i, a_i>$ to $E$;
        add $Q$ to *Query(E)*;
    **enddo**
    **if** $T$ is empty **then**
        add $Q$ to the *NULL-SET*;
    Path-Finding($G$);

**Path-Finding**(G)
    **if** there is a unique path connecting $t_p(A_1)$ and $t_p(A_n)$ **then**
        *detected*:= true;
    **else if** there are alternatives at some set of vertices $S$ **then**
        **for each** $s$ in $S$ **do**
            eliminate alternatives using one of the three elimination techniques;
        **if** unique path is found **then**
            *detected*:= **true**;
        **else**
            *detected*:= **false**;

**FIGURE 8. An Overview of the Inference Detection Algorithm**