ELSEVIER

Computers & Security Vol. 16, No. 8, pp. 687-708, 1997

0167-4048/97 $17.00

Protecting databases from inference attacks*

Thomas H. Hinke, Harry S. Delugach and Randall P. Wolf

Computer Science Department, The University of Alabama in Huntsville, Huntsville, AL 35899, Phone: (205) 895-6455 FAX: (205) 895-6239, E-mail: thinke@cs.uah.edu, delugach@cs.uah.edu

This paper presents a model of database inference and a taxonomy of inference detection approaches. The Merlin inference detection system is presented as an example of an automated inference analysis tool that can assess inference vulnerabilities using the schema of a relational database. A manual inference penetration approach is then offered as a means of detecting inferences that involve instances of data or characteristics of groups of instances. These two approaches are offered as practical approaches that can be applied today to address the database inference problem. The final section discusses future directions in database inference research.

Keywords: computer security, database inference, database security, inference detection tools, inference detection analysis

1. Introduction

A database holds a great amount of data that is critical for the operation of enterprises, be they commercial or government. This data, while providing crucial support for the mission of the enterprise, can also provide a source of sensitive information that is useful for those who are competitors or adversaries of the enterprise. Using available secure database management systems, an enterprise has the ability to provide various degrees of protection for the data. This protection can range from access lists to label-based protection, where security labels are assigned to the data based on its sensitivity. Access to this data is mediated based on the privileges of those who attempt to access it.

Unfortunately, properly protecting individual portions of the database may not provide complete protection. A competitor or adversary may be able to use data that in isolation appears to be properly protected to infer data that is highly sensitive. The problem for the enterprise is to discover these inferences, and then to take necessary countermeasures to close them.

The general solution to the inference problem is difficult, since an adversary can apply a deep body of knowledge in performing an inference attack. Any adversary must be assumed to possess an extensive educational background, as well as familiarity with the specific domain of knowledge of his intended attack. All of this knowledge can be applied in performing the inference attack. The implication of this

* This work was supported under Maryland Procurement Office Contract No. MDA904-94-C-6120.

is that the protectors of the database must also apply this deep knowledge to their inference analysis in order to discover the vulnerabilities of their database before they are discovered by their adversary. While work is proceeding to address database inference detection in light of the deep knowledge required [1], the results of this work are still in the research phase. However, the fact that deep knowledge is required to address the general inference problem fully does not mean that there are no practical techniques available today to apply to important segments of the problem. It is the objective of this paper to describe these practical techniques.

In order to provide both a context for this discussion and a means to help those with responsibility for database protection to understand some of the subtle implications of the inference problem, Section 2 of this paper will present a model of the inference problem that has been developed as part of the AERIE inference project at the University of Alabama in Huntsville, USA. This model provides a useful means of visualizing the various data vulnerabilities associated with the inference problem, as well as terminology for addressing methods for countering the problem. Section 3 will then consider a framework for categorizing the techniques that can be applied to the design of databases in order to reduce their vulnerability to inference attack. Section 4 will discuss current state of the art automated inference techniques, and Section 5 will present manual techniques that can be applied. Finally, Section 6 will contain a brief discussion of future techniques that are still in the research stage.

2. Characterization of Inference Vulnerabilities in Databases

The AERIE inference research project at the University of Alabama in Huntsville has developed a model of the inference problem that is called AERIE (Activities, Entity, Relationships Inference Effects). This model assumes that an adversary desires certain data that is the target of his or her inference attack. This target, which is referred to as the *sensitive target*, can be expressed in terms of the constructs of the AERIE model. This model augments the entity-relationship (ER-model) developed by Chen that is commonly used for database modelling [2].

The AERIE model characterizes possible inference targets in terms of entities, activities and various relationships[3]. An entity, as in the ER-model, is some *thing* that has existence and can be distinguished from other things. Entities are the nouns in the AERIE model. Activities are the verbs and they indicate actions. Relationships are used to represent various associations between entities and activities. Using the model to represent various possible inference targets, we have the following types of targets:

Entity Materialization: this represents an inference that detects the existence of an entity or some characteristics of an entity. An example of this type of inference would be to infer that the entity growing season is underway within a farming community based on the nature of items that show up in a point-of-sale database, such as fertilizer, seed or pesticides.

Activity Materialization: this represents an inference that detects the existence of an activity. For example, one can infer that a winter mountain climbing expedition is about ready to occur based on the ordering of relevant equipment, such as an ice axe, cross country skis or low-temperature sleeping bags.

Entity-Entity Relationship: this represents an inference of a relationship between two entities. An example is the ability to infer the companies that are supporting a very sensitive project, based, for example, on an employee for the company attending a meeting for the project.

Activity-Activity Relationship: this represents a sensitive relationship between activities. For example, the fact that the activity of cotton picking has occurred can be deduced by the fact that a cotton gin's database shows daily ginning activity.

Entity-Activity Relationship: this represents an inference that detects a sensitive relationship between an entity and an activity. An example of this could be inferring that a company was adopting a new process

for the manufacture of computer chips based on its ordering of particular types of equipment.

Relationship-Relationship Relationship: in this type of inference, it is the relationship between relationships that is being inferred. For example, in a classroom setting, the posting of student grades along with a student number would be a relationship. A sorted list of student names would also be a relationship. A sensitive relationship between these two lists could be knowledge that the same sorting algorithm was used in both cases. With this information one could easily deduce the grade associated with each student's name, which is highly sensitive. Another type of relationship-relationship target is the ability to infer some rule (a type of relationship) that has been applied to the data. For example, by scanning the list of ages for members of a retirement community one could infer a rule that each member must be at least 55 years old.

These various types of inference targets represent entities, activities and relationships that occur in the real world. Any database, however, constitutes a microworld. This microworld selectively represents a portion of the real world that is relevant to the enterprise that maintains the database. Within the microworld of the database, sensitive targets that are to be protected by an enterprise are represented by what we call *signatures*. The signature represents the manifestation of the real world inference target within the microworld of the database. For example, the orders for the mountain climbing expedition represent the signatures of the expedition in the database of the equipment retailer.

The relationship between the inference signature and target can be understood in terms of a model developed by Morgenstern [4, 5]. This model uses an inference function — INFER — which is defined in terms of the uncertainty, $H(Y)$, about the value of some information Y, and the relative uncertainty, $H_X(Y)$, about Y, given knowledge of X. $H_X(Y)$ is equal to 0 if X fully discloses the exact value of Y. This means that the uncertainty of Y, given X, is 0; thus there is no uncertainty. If X discloses no information about Y, then $H_X(Y)$ is equal to $H(Y)$. The infer function is defined as:

$$INFER(X \rightarrow Y) = \left|\frac{H(Y) - H_X(Y)}{H(Y)}\right| \text{ if } \left|\frac{H(Y) - H_X(Y)}{H(Y)}\right| > \varepsilon$$
$$= \quad 0 \quad \text{otherwise}$$

In this case, ε is some minimum threshold, below which X supplies what can be considered an insignificant amount of information about Y.

In this model, the inference function INFER $(X \rightarrow Y)$ has a value related to how much information X discloses about Y. INFER has a value of 0 if X discloses no information about Y. It has the value of 1 if X discloses the exact value of Y. In terms of the signatures within the database and the sensitive targets that are to be protected, it would be desirable if INFER (signature $\rightarrow$ target) = 0 for all signatures that are contained within the database.

Each of the various types of sensitive targets expressed in the AERIE model has a signature. The entity signature (E-sig) is a signature in the database that reflects the existence and characteristics of an entity that exists within the real world. For example, as noted for the entity materialization example, the growing season represents the real world target. The database itself may not contain any explicit information about the growing season. However, it may contain an E-sig of the growing season, which consists of records of the sale of items that are normally purchased at the beginning of the growing season. Examples might be seed and fertilizer. Of course, to be able to perform this inference, an adversary would have to have a knowledge base that included the signatures of all of the sensitive targets that were of interest.

The activity signature (A-sig) represents the database manifestation of an activity that occurs in the real world. For example, preparation for an attack could be indicated by database entries for material requisitions and troop movements. When it is not important to differentiate between an activity or entity signature, we can refer to a Q-sig, which represents either an A-sig or an E-sig.

The relationship signature (R-sig) represents the signatures in which the various types of relationships that

may exist in the real world are reflected within the database. A particular type of R-sig is the second path inference [6, 7]. An example of second path inference is shown in *Figure 1*. This represents the real-world target that the identity of companies that are supporting certain sensitive projects must not be disclosed. This is an example of an entity-entity sensitive target. In this example, the relationship between a project and the companies that support the project is considered to be classified at level HIGH, as indicated by the dashed line in the figure. This sensitive target can be inferred at a lower classification level (LOW) by finding a second path that makes the association between company and project.[1]

One such second path shown in the figure is [project, meeting, visitor, company]. This path recognizes the possibility of using a meeting attendee list to associate all of the companies for which the attendees work with the classified project. As can be noted, while such a path exists at the HIGH level (where it can be of no use to a LOW cleared adversary), this path is not visible at the LOW level, as indicated by its dashed line. Thus, this does not provide an exploitable second path at the LOW level.

However, as can be noted, an exploitable LOW second path consisting of [project, escort, visitor, company] does exist. This example assumes a working environment in which all visitors are escorted by some employee who has been designated as an escort for the particular visitor. This path uses the project to which the escort charges his or her time as the basis for associating the visitor's company with the project, and thus forming the classified association using only LOW classified data.[2]

Using this second path, the value of the INFER function would not equal one, since there could be some false associations. For example, assume that a visitor from the Alpha Company was to be escorted by an employee who works only on Project Gamma. Using the second path inference signature, this would be viewed as a signature for the association between the Alpha Company and Project Gamma. However, if the visitor was actually attending a meeting for Project Omega, this sensitive association would not be indicated by this second path. In terms of the INFER function, this means that the value of INFER of this second-path signature for this company-project association is less than one. It is also less than one due to the fact that in many companies, people work on multiple projects. However, the fact that the value of INFER is less than one does not mean that it is not valuable.

The AERIE project has identified the following three tiers of data within the database that can potentially support a sensitive target: schema, group and instance. In a relational database, the schema consists of the definition of the relation tables and associated attributes that are contained in each relation. Second-path inferences (such as the company-project inference that was previously discussed) are an example of a type of inference signature whose potential can be detected with schema-level analysis.

The group tier consists of inference signatures that involve properties about groups of data values. For example, knowledge that certain types of parts are unique to certain types of aircraft could be used to infer that an airbase supports a particular type of aircraft, based on the nature of parts that are shipped to the base. This tier can also be used for inferences involving signatures that involve statistical properties of the data or correlations with various types of data.

The instance tier represents those inference signatures that involve individual tuples of a relation in a relational database. For example, if the chairman of the Beta Company is known to be performing secret negotiations with a foreign government, and the chairman's aircraft (identified by its tail number) is reported to have landed in Iceland, then one can infer that Beta is negotiating with the Government of Iceland. This would be an example of an instance-tier inference.

[1] Note that while HIGH and LOW are used for the example, these can be generalized to various types of hierarchically ordered security levels (e.g. unclassified, confidential, secret or top secret) or various non-hierarchical categories (e.g. competition-sensitive, company Delta proprietary) that have been used to label data within the US Department of Defense.

[2] One solution to this problem would be to polyinstantiate [8] the project number, such that the escort would have a classified project number and an unclassified project number. He would use the unclassified project number for escorting visitors. If the classified project were not tied to this unclassified number, then the inference path would be broken.

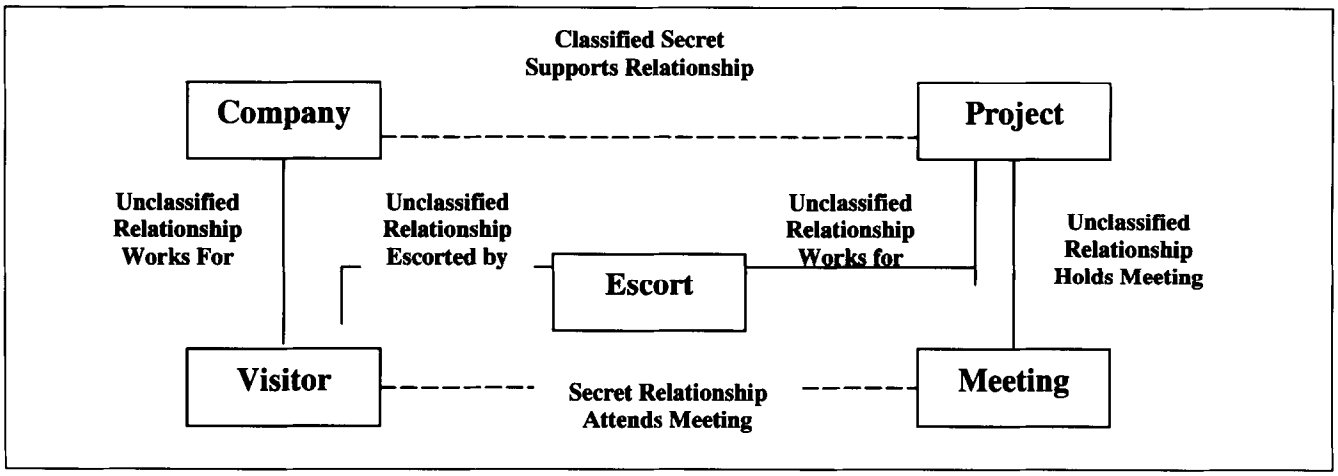

Figure 1. Company-Project Inference Using Escort

3. Characterization of Database Design Techniques

The inference vulnerabilities presented in the previous section can be countered through the use of various inference-orientated design guidelines and techniques. To provide a context to understand how the various techniques relate to each other and where they fit into the continuum of techniques that may be available in the future, this section presents a number of ways to characterize the techniques.

The first means of characterizing the techniques is based on when in the database life-cycle the techniques can be applied. Those techniques that can be applied to the database during database design are called proactive techniques. These techniques do not require that the database data be available; only the schema is required. Those techniques that can be applied to existing database are called reactive techniques. These techniques can use the data instances for their analysis.

The second means of characterizing design techniques is based on the data tier to which the technique is applied. The data tiers were introduced in the previous section.

The final means of characterizing design techniques is based on the sophistication of the approach. A useful model of increasingly more sophisticated security guidelines is illustrated by the historical development of guidelines and methods for removing security flaws from operating systems. The following ordered list, provided by Marvin Schaefer [9] traces this historical development of trusted operating system design guidelines and associated methodologies from the earliest stages to the latest, most technologically advanced stages:

1. Testing

2. Penetration and patch

3. Code review

4. Automated analysis

5. Application of fundamental principles

6. Formal analysis

The initial design guideline for the development of trusted operating systems was limited to just the normal testing of security features that is used for testing the correct functioning of any system. When it was realized that this was not sufficient, security-orientated penetration testing was applied [10]. These penetration techniques were then supplemented with manual reviews of the code in an attempt to discover software anomalies that would lead to potential pene-

tration vulnerabilities [10]. There was also some research in automating the search for such flaws through automated code analysis [11]. The attempts to discover flaws and patch existing operating systems were replaced with the development of security principles that could be used to design systems from inception with security as a guiding design target [12]. For those systems that were to have the highest level of assurance, all of the security relevant software was formally specified in a language that could then be formally verified to prove that — at least at the specification level — the system satisfied a desired security policy [12]. Even today, the actual verification of the code is viewed as beyond the current state of the art for real (as opposed to toy) systems [12].

All of these operating system techniques have the objective of developing software that has a minimal number of software flaws. Using these evolutionary stages as guidance in developing a list of increasingly more sophisticated inference removal techniques for databases, we see some useful analogies and distinct differences. The following are suggested as a reasonable sequence of increasingly more sophisticated stages in what will be an evolving technology base for countering database inference vulnerabilities:

1. Penetration and fix
2. Automated inference analysis
3. Application of fundamental principles to database design
4. Formal analysis of database design

Since there is no functional capability to test for inference vulnerabilities, the initial stage in inference analysis and guidance is provided by an inference-directed penetration analysis of the data. This is a manual, thought-intensive process to find inference vulnerabilities in a particular database. This is analogous to both the second and third stages of the operating system sequence. The second stage of inference analysis and guidance is automating the process of analyzing the data for inference vulnerabilities. This involves some type of automated analysis of the actual data instances. The third stage concerns the application of fundamental inference-prevention principles when the database is designed. Examples of this taken from a database area other than inference, are the various types of integrity constraints that are applied to relational databases (e.g. referential integrity). The fourth and final stage of inference analysis and guidance is formal analysis. An example of formal analysis, taken from a non-inference domain, is the use of functional dependencies in database normalization to reduce data redundancy. Formal analysis derived from an inference domain could involve the formalization of the semantics of the data and an automated analysis to remove potential inference vulnerabilities as that data is placed in a database.

In terms of the reactive and proactive approach categorization, the penetration and fix approach is a reactive technique that can be applied to an existing database, since it involves the analysis of data. As will be shown in Section 5, this represents a group- and instance-tier approach.

The automated-inference-analysis approach is a combination of both reactive and proactive approaches. The reactive approaches involve automated analysis of both the schema- and instance-tier data in an existing database to ferret out potential inference approaches. The proactive approach involves the analysis of the database schema while the database is still under development.

The various types of database inference approaches discussed in this section can be combined with the various data tiers in which inference signatures can be located. This combination results in the matrix of *Figure 2*. The checked cells in *Figure 2* indicate the current state of practical database design techniques that are available for countering inference vulnerabilities for each of the data tiers. Practical approaches for addressing each of these checked areas will be considered in the remainder of this paper.

In the next section, an automated technique for schema-tier analysis will be described. This will be followed by a section that describes a manual technique that can be used for group- and instance-tier

Inference Technology	Applicable Data Tier		
	Schema	Group	Instance
Penetration & Fix		✓	✓
Automated Inference Analysis	✓		
Application of Fundamental Principles			
Forma Analysis of Database Design			

Figure 2: Inference Detection Techniques and Applicable Data Tiers

analysis while automated techniques are still in the research stage.

4. Automated Techniques: Automated Inference Analysis

Existing automated inference analysis tools have focused on schema-tier analysis. These approaches can be used proactively to perform an analysis of an initial database design, or reactively to analyze existing database schemas. Automated support for group and instance-tier data is still in the research stage and will be briefly described in Section 6, which looks to the future.

The initial work in schema-tier inference detection was performed by Hinke for relational database schemas as part of the Advanced Secure DBMS research project at TRW in Redondo Beach, California, USA [7, 13]. This project developed the concept of second-path inference detection. Additional research on second-path detection has been conducted by the US Government [14-17] and SRI International [18]. These approaches all use some form of *path finding*. Work on exploring inference issues associated with functional and multivalued dependencies was conducted by Su and Ozsoyoglu [19, 20]. In its Merlin schema analysis system, the AERIE project has addressed the mechanization of inference detection using second-path analysis based on functional dependencies. The Merlin tool uses an algorithm that is faster than path finding to perform the initial analysis [21]. In Merlin, path finding is used only to elaborate those paths that are identified as problems. As will be described, Merlin also provides some additional assistance in schema-tier analysis.

For relational databases, the value of second-path inference detection is that it does not require a deep knowledge level, as does the detection of some other types of inference signatures; it requires only analysis of the schema. The schema analysis will indicate whether a second path could be constructed with a series of joins. However, actual data instances will have to be consulted to determine if the potential vulnerability is realized in practice. In the company-project example, the inference vulnerability is of concern only for classified projects. If all visitors for classified meetings refused to divulge their company affiliations, then the database modelled in *Figure 1* would not contain instances to support the derivation of the classified relationship between company and project. This means that a second path leading to a sensitive association could not be generated using the database data. Of course, it is possible that an adversary with access to this database could use information external to the database (e.g. company parking stickers on the visitor's car or company affiliation information from professional society or conference attendance lists) to make the link from visitor to company.

The Merlin system automates this second-path detection. In the next section, we will describe the path detection approach used in Merlin. Following that we will describe a path grouping approach that can be used if the schema under analysis contains security labels. We will conclude this discussion of automated techniques with a description of how Merlin provides assistance in breaking second paths.

Project Relation (r1)	
Job_Number[u,u]	Project_Number[u,u]

Figure 3: Project Relation

Job_Cost Relation (r2)			
Employee_Number[u,u]	Job_Number [u,u]	Week_Date[u,u]	Hours[u]

Figure 4: Job_Cost Relation

Visitor_Log Relation (r3)			
Visitor_Name[u,u]	Date[u,u]	Company[u,u]	Escort[u,u]

Figure 5: Visitor Log Relation

Project Company Relation (r4)		
Job_Number[s,s]	Project_Name[s,s]	Company[s,s]

Figure 6: Company Project Relation

4.1 Merlin Path Detection

The basis for Merlin's path detection approach is an algorithm for testing a relational database decomposition for the lossless join property [22, 23]. We have adapted this algorithm for use in the detection of second-path inferences within schema-tier data. We have also extended this algorithm to include subtype relationships.[3] The complete algorithm is described in [1] and a comparison between the performance of this algorithm and conventional path finding is presented in [21]. However, a brief overview of this algorithm is presented here.

To assist in the description of this algorithm, an example that embodies the company-project second path of *Figure 1* will be used. The database for this example consists of the four relations: *project* shown in *Figure 3*; *job_cost* shown in *Figure 4*; *visitor_log* shown in *Figure 5* and *project_company* shown in *Figure 6*. The classifications associated with each attribute are shown. The letters (e.g. [s,s]) associated with each attribute indicates its minimum and maximum classification, with the left *s* indicating a minimum classification of secret and the right *s* indicating a maximum classification of secret. All of the attributes shown indicate a single classification level for all of the attribute values, either all are unclassified, *u*, or all are secret, *s*. However, Merlin will allow for the case in which an attribute has a range of values, such as [*u*-*s*].

The Merlin algorithm represents the schema as a matrix consisting of rows (representing relations) and columns (representing attributes). For each row, the initial state of the algorithm contains an 'a_i' under each attribute column '*i*' if the relation represented by that row contains the attribute represented by the column. Also, for each row, a unique 'b_j' is placed under each column for which the relation represented by row '*j*' does not contain such an attribute. Because of its use of *a*'s and *b*'s, the matrix is called the AB matrix. The AB matrix for our example is shown in *Figure 7*. It will be noted that the range of security levels associated with each attribute is also indicated in the matrix. These will be used for ranking the second paths that are found. It should be noticed that the last row contains only *b*'s. The reason is that in order to address multilevel security, the Merlin algorithm is run several times, once at each classification level. In this current example, the run level is unclassified, which means that the information contained in the relation

[3] A subtype has all of the characteristics of its supertype, but adds some additional constraint. For example, a company escort may be an employee who has been given the responsibility of escorting visitors. This would make the escort a subtype of employee. Any mention of escort also constitutes a mention of employee.

Initial Setting of Matrix S									
Relation	job_ number	project_ name	Employee_ number	week_date	hours	visitor_ name	company	escort	date
R1	a0[u,u]	a1[u,u]	b0	b0	b0	b0	b0	b0	b0
R2	a0[u,u]	b1	a2[u,u]	a3[u,u]	a4[u,u]	b1	b1	b1	b1
R3	b2	b2	b2	b2	b2	a5[u,u]	a6[u,u]	a7[u,u]	a8[u,u]
R4	b3[s,s]	b3[s,s]	b3	b3	b3	b3	b3[s,s]	b3	b3

Figure 7: Initial Matrix

represented by the last row in the AB matrix, *project_company* is more highly classified than the run level. However, the classification level of these classified attributes are still indicated in the AB matrix and will be used for path ranking. More will be said about this later in the paper.

In addition to this AB matrix, the algorithm is also presented with a list of the functional dependencies that apply to the schema. A function dependency between some set of attributes X and some set of attributes Y is denoted by $X \rightarrow Y$, and is read as "X functionally determines Y." The meaning of this functional dependency is that if for any two tuples t_i and t_j in any relation contained in the database to which the functional dependency applies, if the values of the X attributes of tuple t_i, represented as $t_i[X]$, is equal to the value of the X attributes of tuple t_j, represented as $t_j[X]$, then $t_i[Y] = t_j[Y]$. In other words, if the values for the set of attributes represented by X are the same in both tuples, then the values for the set of attributes represented by Y will also be the same in both tuples. The functional dependencies that apply to our example are as follows:

1. employee_number → job_number, which means that an employee number uniquely determines the job number on which the employee is currently working. This assumes that an employee works only on a single project.

2. visitor_name → company, which means that a visitor works for only a single company.

3. visitor_name, date → escort, which means that each visitor is escorted by a single escort.

4. job_number → project_name, which means that a job number uniquely determines a project name.

5. employee_number, job_number, week_date → hours, which means that the combination of employee number, job number and week date, such as the Friday date for each week can be used to identify the number of hours worked.

6. project_name → company, which means that a project name uniquely determines a single company that is supporting the project.

7. project_name → job_number, which means that a project name uniquely determines the job number to which the employee charges his or her time.

The other two types of information that this algorithm uses are the subtype relationship and the foreign keys. For this example, an *escort* is a subtype of *employee_number.* This means that any time a row contains a reference to *escort,* we can also fill in a similar reference to *employee_number.* A foreign key is an attribute in one relation that references an attribute that is a key for another relation. The referenced key attribute may have a different name than the non-key attribute that references it. Merlin treats foreign keys as synonyms. Thus, any time that a row contains a reference to a foreign key, we can also fill in a similar reference to the key to which it refers, and vice versa.

The algorithm works in cycles. In a single cycle of the algorithm, all of the functional dependencies are matched with the AB matrix. Each functional dependency $X \rightarrow Y$ is considered in turn and is matched against the AB matrix to see if there are two or more

rows that have identical values for the X attributes, where these identical values can be 'a' values, 'b' values or 'c' values (the meaning of c will be described shortly). If there are two or more rows that match, these rows are considered to form a set R. The Y values for the set R are then set to equal values using the following method: if none of the rows in R contains an 'a' or 'c' value for Y, then all of the Y values are set to the lowest 'b' value in the set R. If one row in the set contains an 'a' or 'c' value for Y, then all of the other rows in R are changed from 'b_j' to 'c_j' for the respective attribute in column 'j'. If one of the 'b' values changed to a 'c' value is identical to some 'b' value not in R, then all of those 'b' values not in R are also changed to 'c'. The 'c' value is used to represent places in the AB matrix where a 'b' (which means that it was not visible in the relation) is changed to an attribute that is visible, based on the application of a functional dependency.

For example, using the first functional dependency *job_number* → *project_name*, the Merlin algorithm will scan the X column, *job_number*, and find that both the R_1 and R_2 rows have identical values, $a0$ in this case. These two rows constitute the set R for this stage. Now the Merlin algorithm will scan the Y column, *project_name* looking only at the R rows (in this case R_1 and R_2) and find that row R_1 has an $a1$ and R_2 has a $b1$. The $b1$ value in R_2 will be replaced with a $c1$ value. The replacement is $c1$ rather than $a1$ so that newly replaced attributes can be identified. These provide the basis for a second path. What this replacement means is that since a functional dependency exists and since a mapping in R_1 exists for this functional dependency, this mapping can be used in R_2 to fill in the values for all *project_names*, based on the *R1* mapping. In essence, R_2 is being expanded via a mock join operation that is based on functional dependencies.

A single cycle is completed once all of the functional dependencies have been checked against the AB matrix. The algorithm terminates following a cycle in which no changes are made to the AB matrix.

In this algorithm, a second path exists between two attributes if there exists some row such that the two attributes are represented by two 'c's' or a 'c' and an 'a'. If a row represents these two attributes as both 'a's', this is not a second path, since this indicates that the relation initially contained both attributes. We call this a 'first path'.

The meaning of second path as used for the Merlin algorithm is that an association exists between all of the 'a' and 'c' attributes that share a row. Since all of the non-key attributes within a relation are functionally dependent upon the primary key of that relation, all non-key attributes in a relation have a functional dependency relationship with the key of that relationship. Also, since the Merlin algorithm 'grows' each row by adding new attributes (based on a match on the left-hand side of the functional dependency) that have a functional dependency relationship with an attribute that is already in the row, this new attribute will also have a functional dependency relationship with the primary key of that row. It can be observed, however, that some of the second-path associations will show a second path between two non-key attributes. These may or may not provide useful information. As will be described shortly, Merlin has an approach to rank the paths discovered so that the most promising second paths can be readily identified.

As mentioned earlier, to address multilevel security, Merlin can be executed at different security classifications, called *run levels*. At a particular run level, the AB matrix is constructed from only those attributes that are visible at that run level. This means that the security level of the attributes used in the AB matrix must be dominated by the run level.[4]

The final AB matrix is shown in *Figure 8*. This algorithm has detected that a second path exists between *project_name* and *company*. Since there exists a row that contains an a and a c for *company* and *project_name*, this means that there exists the ability to build an association between a *project_name* and the external *company* that is supporting that project, based on an employee of that company being escorted by an employee who

[4] In terms of security policies used by the US Department of Defense [12, 24], a security classification L_i dominates another security classification L_j if the hierarchical component of L_i (e.g. top secret > secret > confidential > unclassified) is greater than or equal to the hierarchical component of L_j and the set of non-hierarchical categories associated with L_j is a subset of the set of categories associated with L_i.

After Processing S									
Relation	job_ number	project_ name	Employee_ number	week_date	hours	visitor_ name	company	escort	date
R1	a0[u,u]	a1[u,u]	b0	b0	b0	b0	c6	b0	b0
R2	a0[u,u]	c1	a2[u,u]	a3[u,u]	a4[u,u]	b1	c6	b1	b1
R3	c0	c1	c2	b2	b2	a5[u,u]	a6[u,u]	a7[u,u]	a8[u,u]
R4	b3[s,s]	b3[s,s]	b3	b3	b3	b3	b3[s,s]	b3	b3

Figure 8: Final Matrix

is working on the project. Of course, we cannot be certain this association does exist, because the person who performed the escort function may not necessarily be associated with the project for which the person is visiting the facility. Still, assuming that escorts normally escort visitors for their projects, this method will provide a reasonable means of detecting real inference vulnerabilities.

It will also be noted that the row for relation R_4 still shows all *b*'s. However, the classification range information available to the Merlin path ranking code indicates the initial classification for each of the three attributes for the R_4 relation. This information will be used for the path ranking, which will be described in the next section.

This algorithm is fast and will identify that a second path exists by indicating those attribute pairs that are joined by a second path. However, the algorithm does not indicate the various attributes that provide the realization of this path beyond the two endpoints. This detection of a path's existence without the actual identification of the components of the path results in a considerable speed-up in the inference detection process. Comparisons between the Merlin second-path detection algorithm and a conventional path-finding algorithm showed that for a test database, called the AERIE database schema (consisting of 33 relations and 104 distinct attributes) the Merlin second-path detection algorithm took 7.23 seconds, which was 10.9 times faster than the time it took a conventional path-finding algorithm to find the first path for all possible combinations of attributes. On another test schema developed by SRI International from a ER-diagram supplied by the US Air Force Rome Laboratory [25], consisting of 48 relations and 253 distinct attributes, the Merlin algorithm took 72.2 seconds, which was 20.1 times faster than the path finding algorithm. [21] As can be noted, the value of the Merlin path detection increases as the size of the database increases.

The computational complexity of this algorithm is based on the fact that for each cycle of the algorithm, each attribute A of each relation R (row of the matrix) must be considered with respect to each of the functional dependencies D. For C cycles, this results in processing complexity of $O(C * R * A * D)$. Assuming under worst case conditions that one attribute of one relation is changed for each cycle, the number of cycles is bounded by the number of attributes times the number of rows, which is $R * A$. Substituting this for C, we have processing complexity of $O(R^2 * A^2)$. In actual practice this algorithm is much better than this since our experience has been that much fewer than $R * A$ cycles are required. In tests using both the AERIE and the SRI database schemas, we have found that only four cycles were actually required.

The next section presents Merlin's path ranking approach.

4.2 Merlin Path Analysis

To use the Merlin tool, an analyst encodes the attributes and associated attribute security levels for each relation in the database. The analyst will then select a run level (which is lower than the most classified data in the database) at which to perform the analysis. The purpose of this run level is to determine the second paths that exist at this particular security

level. As was noted in the previous section, Merlin operates in cycles. In each cycle, all of the functional dependencies are applied to the AB matrix to see if additional changes result. When one complete cycle results in no additional changes in the AB matrix, Merlin reports its results. These results are reported in terms of attribute pairs. Each row of the AB matrix is scanned to determine which attributes are associated with what other attributes. Those attribute pairs that are new indicate a second path. The Merlin algorithm does not, however, identify the components of this path, only the end-point attributes of the path. Standard path-finding techniques must be used if the set of attributes that constitutes this path is to be identified.

The results of a Merlin analysis on a database of even moderate size (e.g. 50 relations) will be a considerable number of second paths (attribute pairs). In general, the database designer must analyze each of the attribute pairs that have second paths to determine which may be sensitive and which may be ignored.

The database designer can be assisted in this task of path analysis if the schema contains security labels. The AERIE project has developed a technique of automatic path grouping based on a schema that has security labels assigned to the attributes of the schema. The AERIE path grouping is based on schemas described in the MSQL (multilevel SQL) language, a multilevel extension to the SQL database language developed by SRI International.[26] However, the technique is not specific to MSQL; any schema classification language would support this path-grouping technique. MSQL provides the capability for the database designer to specify the classification of each attribute in a relation. If all of the values associated with a particular attribute of a relation are uniformly classified at the same security level, then a single classification is associated with the attribute. If the values associated with an attribute are not uniformly classified, then a classification range is associated with the attribute. This range indicates the lowest and highest classifications that apply to all of the values that are associated with the attribute within the relation. The security classifications of a given attribute may differ in different relations. An attribute could, for example, be unclassified in one relation but be secret to top secret in another relation. This can be seen in *Figure 3*, where the *project_name* attribute in the *Project* relation is unclassified, while it is secret in the *Project_Company* relation, *Figure 6*, because of its association with the *company*.

Under the AERIE path-ranking technique, the schema is used as a classification guide for paths that are detected by Merlin. Each relation can be viewed as a set of attribute pairs consisting of each attribute in the relation associated with each of the other attributes in the relation. Looked at in the most general sense, each attribute within any relation has two classifications, representing the minimum and maximum security levels for the attribute's values. For the case of uniformly classified attributes, the minimum and maximum security levels are the same. Each attribute pair association would have four classifications, representing the minimum and maximum of both of the associated attributes. For example, the association between *job_number* and *company* in relation R_4 has the classifications minimum *s* and maximum *s* for *job_number* and minimum *s* and maximum *s* for *company*.

Identical attribute pairs may exist in different relations, and it is possible that they will have different classifications. This difference may reflect the fact that the relations from which these attributes were taken represent different entities, but have some of the same attribute names. For example, in an aircraft relation, the attributes of speed and range may be classified secret; in a motor vehicle relation, both of these attributes may be unclassified. Since there may be a number of identical attribute pairs with differing classifications, the set of attribute pairs and their associated classifications does not constitute a real classification guide; however, it does provide some indication of the possible classification of paths and a useful basis for ranking the paths.

As has been mentioned, Merlin is executed at a particular run level. This run level represents a potential access level of an adversary. At the run level, Merlin can view only those attributes whose classification is dominated by the run level, but, as shown in *Figure 8* the AB matrix contains classification level information

for all levels. The second paths that are generated use only those attributes that are dominated by the run level. To detect inferences, Merlin's run level will be below that of the most highly classified attributes in the database. Thus there will be attributes to which the Merlin second-path detector does not have access, since they exist at a security level that strictly dominates the run level, where a security level strictly dominates another level if it dominates it but is not equal to it.

A path discovered by Merlin is described in terms of the pair of attributes that delineate the end-points of that path. To perform the path rankings, each new path discovered by Merlin is compared with the set of attribute pairs that are generated from the schema. This set of schema-based attribute pairs reflects the entire schema; they are not limited to the attributes that are viewable at the run level. The groupings will be made based on this comparison.

Using a refinement of the approach presented in [21], the various paths are ranked into grades, which differ in their potential security vulnerability. These grades are defined in terms of four predicates. The values of the predicates 1V, 1N, AV and AN are based on an analysis of the schema with respect to the newly detected path that has been found (and is to be ranked) and the run level. The schema can be viewed as a set of path templates, each template being designated by a pair of attributes that exist within at least one relation as specified in the original schema. For example, the *Company_Project* relation in *Figure 6* contains three path templates *(job_number, project_name), (job_number, company)* and *(project_name, company)*. Multiple instances of a path template can exist, one for each relation that contains the two attributes that define the path template. Each path template instance has a lower and upper security level associated with each of the attributes of the path template instance. Thus the path template instance for the *(job_number, project_name)* template from the *Company_Project* relation has a lower and upper security level of secret for both attributes. The instance of this path template from the *Project* relation has a lower and upper bound of unclassified for both attributes.

The predicates used to group the paths are defined in terms of the relationship between the run level, the set of path template instances that are viewable at the run level and the new path, (i,j) that is to be ranked. The security level of each path template instance, consisting of attributes i and j, can be defined in terms of the lowest security level of i, L_i, the lowest security level of j, L_j, the upper security level of i , U_i, the upper security level of j, U_j and the security level of the run L_R. In the predicate definitions we will use the notation $X \leq Y$ to mean that Y dominates X.

The following define each of the four predicates:

- The predicate 1V (at least one path potentially visible) is true for path (i,j) if at least one path template instance (i,j) in the schema is potentially visible. This is true if there exists some path template instance such that $L_i \leq L_R$ AND $L_j \leq L_R$. What this says is that for a path to be visible in the schema, both of its end-points must be visible. Note that this does not say that the actual data will contain a realization of this path. Since each attribute can be represented by a range of security levels, it is possible that the schema indicates that a path could be visible, since the lower bound of both of the end-points of the path is at a security level dominated by the run level. However, the upper bound on the attribute classification for one or both end-points could strictly dominate the run level. In this case, all of the actual database data that would permit a realization of this path could be above the run level. This is why it is said that a path is *potentially* visible.

- The predicate 1N (at least one path is potentially not visible) is true for path (i,j) if at least one path template instance (i,j) in the schema is potentially invisible. This is true if there exists some path template instance such that $Ui > L_R$ OR $U_j > L_R$. What this says is that a path is invisible if at least one of its end-points is invisible.

- The predicate AV (all paths visible) is true for path (i,j) if all path template instances (i,j) in the schema are visible. This is true if for all path template instances, $U_i \leq L_R$ AND $U_j \leq L_R$. This says that all paths are visible if the upper bound on the end-

points of the path has a classification that is dominated by the run level.

- The predicate *AN* (all paths are not visible) is true for path (i,j) if all path instances (i,j) in the schema are invisible. This is true if for all path template instances $L_i > L_R$ AND $L_j > L_R$. This says that all paths are invisible if the lower bound on the security levels of the end-points of the path is classified at a security level that strictly dominates the run level.

Using the predicates, newly discovered paths are analyzed and partitioned into vulnerability grades. In all cases, this grading process is triggered by the discovery of a new path in the AB matrix (e.g. at least one attribute in an attribute pair is a 'c'). The following are the criteria for each grade:

Red: *AN* = true. All of the templates for the newly discovered path are invisible at the run level. Thus, red paths represent a very high potential inference vulnerability, since all examples of these paths taken from the schema are classified at a level that strictly dominates the run level.

Yellow: $1V$ = True AND $1N$ = True. At least two path templates matching the newly found path exist in the relation schema, and at least one path template instance is visible and at least one is invisible at the run level. Thus, yellow paths may or may not be a problem, since we have examples of paths taken from the schema that are both visible and invisible at the run level. In general, while there is a high probability that red paths are a problem, there is a smaller probability that yellow paths are a problem.

White: No path template matching this newly found path exists in the relation schema. Thus, the schema classification can provide no guidance as to the potential classification of this newly found path. White paths must be carefully reviewed by a knowledgeable analyst.

Green: AV = true. The only instances of these paths exist in the relation at levels that are dominated by the run level. In terms of what can be inferred about this association from the original schema, green paths do not appear to represent a problem.

In addition to being ranked into colour grades, each path can be further ranked into three subgrades, depending upon whether the attributes that designate the second paths are primary keys, including near keys[5] or non-keys. The following are the three subgrades:

Subgrade 1: both attributes of the second path are primary keys, near keys or part of a primary key;

Subgrade 2: only one of the attributes of the second path is a primary key, near key or a part of a primary key, and

Subgrade 3: neither of the attributes of the second path are a primary key, near key or a part of a primary key.

While not currently supported by the implementation, one of the reviewers of this paper suggested that it would be useful to order the white relationships according to transitivity — the number of hops involved in the transitive association. This would be a useful means of partitioning the paths, since longer paths would normally be viewed as less problematic. This would require path instantiation, which is discussed in the next section.

An example of the path grades from the company-project example are shown in *Figure 9*. Green paths are not shown since they are not considered to be a problem.

Having found the paths and ranked them by potential vulnerability severity, Merlin can then be used to provide automated assistance in determining where a path can be broken. It should be noted that at this point in the analysis, Merlin knows that a second path exists, but it does not know the series of attributes that comprise the path; it knows only the attributes that serve as the end-points of the path. In order to break a path, the attributes that comprise the path must be known. A path will then be broken by classifying one or more of the attributes that contributes to it.

[5] An attribute that, while not technically a key, is sufficiently close to a key that in most cases it will designate a specific tuple in a relation.

Grade	Subgrade	Attribute 1	Attribute 2
Red	1	job_number	company
		project_name	company
Yellow	1	job_number	project_name
White	1	employee_number	company
		employee_number	date
		employee_number	escort
		employee_number	visitor_name
		job_number	date
		job_number	escort
		job_number	visitor_name
		project_name	date
		project_name	employee_number
		project_name	escort
		project_name	visitor_name
		project_name	week_date
		week_date	company
	2	company	hours
		project_name	hours

Figure 9: Path Grading Example

4.3 Merlin Path Breaking

Path breaking in Merlin is based on path finding followed by a manual analysis of the components of the path to determine the most attractive places to break the path.

As has been described, the path-detection capability of Merlin determines the existence of all paths that connect two attributes. One of the ways that these two end-points can be related is though a series of functional dependency relationships. Because of the transitivity property of functional dependencies, this provides the most meaningful association from an inference perspective, since the end-points will themselves have a functional dependency relationship. As was previously noted, not all of the paths found by Merlin fall into this category. Because of this, Merlin supports two different path finding approaches, based on whether the path is a functional-dependency-based path or a path based on attribute associations.

A functional-dependency-based path consists of a sequence of links that are drawn from the set of functional dependencies applicable to the database. An attribute-association-based path is comprised of links drawn from the set of attribute pairs that the Merlin tool has indicated have second paths associating them, or attribute pairs that are members of the same relation and thus are joined by what we call first paths.

Merlin can compute all paths connecting two attributes using either functional-dependency-based paths or attribute-association-based paths. As has been noted, the functional-dependency-based paths are the most meaningful. However, it some cases, there may be a second path that is not based on a functional-dependency-based path. In this case, the attribute-association-based path must be computed. In either case, once these paths are computed, they must be analyzed to determine which attribute should be classified to break the path. Since each pair of attributes may have a number of second paths connecting them, a number of different attributes may have to have their classification raised in order to break all of these paths. This identification of which attribute selected to have

their classification increased is a manual process. However, since the Merlin tool is fast, one approach is to analyze the path and then reclassify particular attributes and re-run the Merlin tool to determine if the path has been broked. In this way, Merlin can be interactively used to ask "What if questions" much as one uses spread-sheet programs in financial analysis.

5. Manual Techniques: Penetration and Fix

For group- and instance-tier data, the initial stage of inference-orientated database analysis and design is inference penetration testing of the database. Analogous to operating system penetrations, this is a manual process whose objective is to locate inference vulnerabilities in the database. Useful guidance in performing inference penetration testing of a database can be gained by reviewing the steps used for operating system penetrations. A widely used operating system penetration methodology is Weissman's Flaw Hypothesis Methodology (FHM) [10]. This methodology consists of the following four steps: generation of a set of flaw hypotheses; confirmation of each flaw through desk checking or testing; generalization of each flaw into a generic system vulnerability and elimination of each flaw through a change in the code or design of the system, as appropriate.

These four steps are also applicable to inference penetration of databases, although the precise methods used for database inference detection will differ from those used for operating system penetrations due to the very different nature of operating system code and database data. In operating system penetrations, one is looking for flaws in the code that would permit one to gain access to unauthorized data or take control of the operating system. In database inference analysis, in contrast, one is looking at ways that one can exploit the data that can be accessed to gain insight into data to which access is prohibited. Nevertheless, these four penetration steps provide a useful means for organizing our thinking about the task of inference penetrations. Each of these steps will be considered in the following discussion.

5.1 Inference Hypothesis Generation

The initial stage of inference penetration analysis is the development of a list of sensitive targets. As has been noted, these targets represent the information that is to be protected from an inference attack that uses the database. In terms of the AERIE model, these sensitive targets can be expressed in terms of entities, activities and the various relationships between entities and activities.

For each of these sensitive targets, the next step is the development of plausible hypotheses as to how these sensitive targets could be inferred from the particular database that is being analyzed. These hypotheses represent the potential inference signatures that could potentially exist in the database for each of the sensitive targets that have been identified. A useful question to ask in the development of plausible inference hypotheses is, "Which data currently in the database would change if the inference target did not exist?" In answering this question, we are concerned with changes in the data at the group or instance tier. At the group tier, we are concerned not only about changes in the data, but changes in the statistical or multivariate properties of the data as well.

Taking some guidance from the Flaw Hypothesis Methodology, the inference hypotheses should be rated according to two factors. The first is an estimate of the probability that the inference hypothesis will actually be confirmed as a valid signature for its specific target. The second is an estimate of the damage that such an inference signature would cause if it were confirmed as a viable signature. These estimates represent judgments of those conducting the inference penetration analysis and should be specified in terms of high (H), medium (M) and low (L). The combination of these two ratings leads to nine different groupings, with HH indicating those hypotheses that have the highest potential for damage and LL being those with the lowest. Those flaws with a high or medium rank for either factor are the first ones that should be investigated in the inference hypothesis validation phase.

5.2 Inference Hypothesis Validation

During this stage of the inference penetration analysis, each of the inference hypotheses must be assessed to determine if it represents a valid inference signature for the specified sensitive target. Again, using the FHM

as a guide, this assessment is made in terms of a 'gedanken' (thought) experiment. These gedanken experiments can be organized in terms of various types of analysis techniques that can be applied to data. In general, these methods are tied to various types of signatures that exist within the various data tiers of the database. Group-tier inference hypotheses can be assessed in terms of various types of standard statistical or multivariate analysis [27] techniques. These are widely used in data analysis and are beyond the scope of this paper.

One means of validating the instance-tier inference hypothesis that has resulted from the AERIE project is an inference-driven sensitivity analysis. This sensitivity experiment is based on asking questions that are related to the particular nature of the sensitive target, as exhibited by the AERIE model. For entities and activities, the question is: "How would the database change if the existence state of the entity or activity changed?" For example, consider an F-17 fighter aircraft as being a sensitive target. In a logistics database, the inference analyst would ask how the database would change if there was suddenly no longer an F-17 fighter. Presumably this would mean that parts associated exclusively with an F-17 would disappear from the inventory list, and parts that were used on the F-17 — but not exclusively — would have their part-count changed. These then represent the sensitive signatures for entities and activities that must be addressed if inference signatures for an F-17 are to be eliminated.

For the various types of relationships (R-sigs), the question one would ask is how the database would change if the truth of the relationship changed. In a company-project relationship, where the sensitive target consists of the companies supporting the AERIE project, the question to be asked is, "What changes would exist if there were not a sensitive AERIE project that was related to a number of performing companies?"

Another example of an instance-tier inference signature comes from what we have termed the "California example". This example inference involves a sensitive target that asserts that a significant number of people are moving out of California. The database of interest is from a truck rental company. However, it is not the database that shows the actual rentals of specific trucks, but one that indicates — on a location basis — the one-way rental rate between that particular location and every other location that is served by the company. Performing a sensitivity analysis on this database during the summer of 1992 one would have found that the rental rates out of California were significantly higher than the rates into California. This type of inference, while very difficult to model, could easily be identified by the sensitivity analysis conducted as a gedanken experiment. This problem has been presented to a number of different audiences, and without exception, within a matter of seconds someone can identify the sensitive field that would provide this inference. Therefore, we consider this sensitivity analysis as a reasonable means of validating potential inference vulnerabilities until more automated approaches are available.

To infer a sensitive rule, we would look at states of the database before and after transactions that involve that rule. The question to be asked is how the database state after the transaction would change if the rule were not applied. The answer to this question will provide guidance on what must be protected if the rule signature is to be eliminated.

5.3 Inference Flaw Generalization

During this step, any inference signatures that are identified during the previous step are analyzed to see if they can be generalized to more generic inference problems. The flaw generalization may lead to additional inference hypotheses that were not identified during the inference hypothesis generation stage.

5.4 Inference Elimination

The primary tool for protecting a database from inference vulnerabilities is to classify the inference signature data that was discovered in the two previous steps. A sufficient portion of the signature data must be classified so as either to eliminate the signature totally or reduce the signature to a residue — one that when operated on by an INFER function provides little or no additional information about the sensitive target. At that point the residue of the signature would be of little value to an adversary.

If the relational database management system supports element level classification, then each individual value within a tuple of a relation can have a distinct classification. With element-level classification, the inference channel can be closed by classifying the particular field value that provides the signature. If only tuple-level classification is available, then the entire tuple will have to be classified. If this is not acceptable, then the relation will have to be split into one relation that contains the portion of the signature attributes that need to be classified, and another relation that contains the attributes that one does not desire to classify.

If a target has multiple signatures, each of which is reduced to a non-sensitive residue, care must be taken to ensure that these various signature residues cannot be intersected to result in a new signature that can still be used to perform the inference. For example, the initial signature might permit one to deduce a particular entity. By classifying a portion of the signature, the number of entities to which the signature could refer would be increased. However, the intersection might permit the initial entity to be identified. As an example, consider a person's automobile that is identified by the signature, green Ford truck with license number California WIE188. This could be reduced, through classification, to a residue consisting of green truck. Assume that another signature for the person who owns the truck is a dentist, license number K611744, in Twin Lakes, California. Assume that this signature was reduced to just Twin Lakes, California. However, by combining the two residues, we would have green truck in Twin Lakes, California. Since Twin Lakes has less than 100 homes, this might be sufficient to identify the particular person. Breaking this inference is difficult since the set of property owners in Twin Lakes is so small. If the dentist is associated only with the county of residence (e.g. Mono County) rather than the area of Twin Lakes, this inference could be broken.

This type of inference detection is usually associated with statistical databases. To preserve the security of these databases, responses which involve only a very small number of the subjects of the database, or which involve most of the subjects are not answerable, since they could be used to infer sensitive data about a single subject. A considerable amount of research has been performed on how to determine if sensitive data can be extracted from statistical databases. The work by Denning [28] considers how characteristic formulas called trackers can be used to extract sensitive data from statistical databases. This aspect of database inference differs from the previously described Merlin work, in that it is concerned with inferring data about individual instances within an entity, while Merlin is concerned with inferring relationships between different entities in the database.

6. Future Directions in Inference Detection Approaches

The checked cells in *Figure 2* indicate the practical database design techniques we have provided for countering inference vulnerabilities for each of the three data tiers. Approaches for each of these checked cells have been presented in previous sections of this paper. This section discusses research that is currently underway to provide practical approaches for the other cells of *Figure 2*. In this paper, we will consider future research directions for the schema tier and for the lower tiers.

6.1 Schema Tier

At the schema tier, we have presented an example of a very fast schema analysis tool and indicated where other research centres have developed schema-tier tools. Research at SRI International focused on the ability to follow paths based on foreign keys. [18] For Merlin, this analysis is based on functional dependencies, foreign keys and subtypes. While this is a sound inference detection approach in that all paths that are detected have the potential to lead to an inference problem in the database, the approach is not complete, since there are legitimate inferences that will not be detected. As an illustration, in the company-project second path example, the critical functional dependency that permits the second path to be detected is employee_number → project. This implies that an employee (and an escort, since an escort is a subtype of employee_number) can work for only one project. If an employee could work on multiple projects, then a second path inference would not have been detected, since we would no longer have the critical functional dependency. Likewise, if an escort could work

for two or more projects, then there would no longer be a functional dependency between escort and project. The removal of this functional dependency would mean that the Merlin tool would no longer be able to detect this second path.

Note that this lack of completeness also occurs if a path-finding approach, such as that used by SRI International, is applied. In their path-finding approach, foreign keys are used to create the second path. If an employee can work on multiple projects, then employee_number would no longer be a key for the Project relation, since it would not uniquely identify a tuple of the relation. In this case, the key for the Project relation would be the multi-attribute key job_number plus project_number. There is no foreign key in the Visitor_log relation to permit a path from Visitor_log to Job_cost in order to connect company with project number.

One approach to addressing this problem is to permit the $\rightarrow$ relationships to include multivalued dependencies [22], which are those relationships in which the right-hand side of the arrow represents a set of values, rather than just a single value. For those cases in which the potential set of right-hand-side values is relatively small, then one can use Merlin to detect potential inference problems. What is given up in this case is soundness, since not all paths that are identified will be legitimate second paths.

6.2 Group and Instance Tier

The major challenge in inference-orientated security design techniques for databases is to push the group and instance-tier analysis technique to levels beyond the manual penetration approach described in this paper. This section will highlight research approaches that have been investigated for this area.

One of the earliest efforts in this area was an approach by Buczkowski in which an expert system is used to encoded inference rules that can be used to detect inference vulnerabilities [29]. In this case, one must first know about the inference vulnerabilities and then encode the necessary rules in the expert system to be able to detect the existence of signatures for sensitive targets.

The AERIE project has worked to extend the Merlin tool into instance-tier data. The challenges here are that the amount of data is significantly greater at the instance level than it is at the schema level and there may be significant inferences which are not built on functional dependency relationships. In [30] we have presented a model-based approach for detecting instance-level inferences that does not rely on functional dependencies. Under this approach two entities X and Y are related by a minimum and maximum association cardinality. A maximum association cardinality $ACmax(X \rightarrow Y)$ is the maximum number of Y instances that can be associated with a single X instance using joins (based on attributes that may or may not be keys). Similarly, the minimum association cardinality $ACmin\ (X \rightarrow Y)$, is the minimum number of Y instances that can be associated with a single X instance, using joins. These minimum and maximum association cardinalities can be used to annotate an object model of a database, such that the model can be used to assist in instance-tier analysis.

Figure 10 shows a portion of a company database that has been modelled in a variation of the Unified

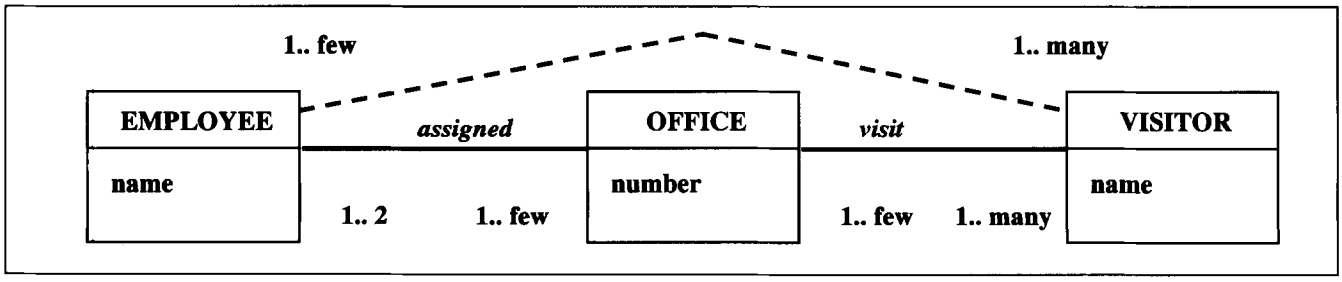

Figure 10: Visitor-Employee Inference

Modelling Language [31]. The notation also shows the minimum and maximum association cardinalities between the various entities. For example, the AC_{max}(Visitor → Office) is *few*, which means that a visitor visits only a maximum of a *few* offices, where few is defined as an application specific number that is sufficiently small to constitute an inference problem. The AC_{min} (Visitor → Office) is one, which means that each visitor visits a minimum of one office. In the opposite direction, the AC_{max} (Office → Visitor) is *many*, indicating that many visitors visit each office. In contrast to *few*, *many* is an application specific number that is sufficiently large that it does not constitute an inference problem. The AC_{min} (Office → Visitor) is one meaning that each office has a minimum of one visitor. These association cardinalities are based on knowledge that a security analyst has of the actual database, or they could result from accesses made to the database to verify or discover the association cardinalities. Finally, the dotted line is used to indicate a sensitive association between visitors and employees, while the solid line indicates that the information is publicly available.

By using this annotated model, the security analyst (or a program) can use the model to determine if the database is vulnerable to an inference problem. An inference problem in this case is defined as the ability to find the association between a visitor and an employee or an employee and a visitor, since this is the sensitive association.

Since the association cardinalities are transitive, the AC_{max} (Visitor → Employee) = AC_{max} *(Visitor → Office)* × AC_{max} *(Office → Employee)*. In this case this is 2 * few. Our view is that any relatively small constant times *few* or even a small number of multiples of *few* constitutes an inference problem. In this case, each visitor can be associated with a small number of employees and this could cause an inference problem, since this association is sensitive.

In the opposite direction, AC_{max} *(Employee → Visitor)* is few * many, which is not considered an inference problem. However, since AC_{min} *(Employee → Visitor)* is one, this means that it is possible that an individual employee could be associated with a single visitor, and this would be an inference problem since this association is sensitive. For large object models, we have applied transitive closure algorithms (with appropriate adjustments for the association cardinalities) to determine those entity-to-entity association cardinalities that could constitute a potential inference problem. As necessary, particular associations can be instantiated using standard database queries.

Other instance-level inference detection research has also been performed by a research group at the University of Tulsa [32]. They have used joins to instantiate sensitive associations.

Another very different approach that has been suggested uses the database itself to derive potential inference relationships. Marks suggests that material implications derived from the actual data could be used as the basis for inference detection [33]. A material implication is not necessarily based on a causal relationship, but indicates only that the particular values appear together. This could be due to coincidence or because there is some relationship between these values. This requires that data be grouped and rules be available to relate one group to another. To illustrate this approach, Marks uses an example in which the names of the personnel assigned to the classified Manhattan Project are classified. However, if it is the case that all nuclear engineers in the company are assigned to the Manhattan Project, then there is an inference vulnerability between a skill category, nuclear engineering (which is not classified when associated with an employee name) and the personnel assigned to a classified project. These types of relationships could be identified by scanning the database instances for groups, and then assessing whether material implications exist between these groups. This work appears to be in the early stages, but shows some promise of providing a way of using the data itself to assess potential inference vulnerabilities.

An area that has considerable potential for addressing group-tier inferences is research on statistical databases. The US Bureau of the Census is noted for its effort in ensuring that personally identifiable data cannot be derived from the census data that is released. Work on multivariate analysis would also be

applicable to this area [27]. While consideration of statistical and multivariate approaches is beyond the scope of this paper, they are noted as areas that could be applied to group-tier inferences. The work of Denning [28] as previously noted, is one of the seminal works in this area.

As one attempts to automate inference analysis for group and instance-tier data, one of the difficult problems that must be addressed is the deeper levels of knowledge that are now required, that were not required for schema-tier analysis. This knowledge is required since the adversary can be assumed to bring to the inference attack many years of education as well as specialized expertise in the areas of interest. In a previous paper [1] we have suggested a depth-first knowledge acquisition approach based on microanalyzed knowledge chunks[6], which involves an inference-orientated, in-depth encoding of knowledge relevant to each of the relations within a relational database. As research moves from schema-level data to instance-tier data, the need for deep knowledge greatly increases. The general solution of this problem is an open research question. We believe that the general approach lies in capturing inference-relevant semantics of the data to be analyzed using an approach such as the microanalyzed knowledge chunks described in [1].

Acknowledgements

We would like to acknowledge the very helpful comments provided by one of the reviewers of this paper.

References

[1] Hinke, T.H., Delugach, H.S. and Chandrasekhar, A. 1994. Layered Knowledge Chunks for Database Inference, in *Database Security VII: Status and Prospects*, T.F. Keefe and C.E. Landwehr, Editors. 1994, North-Holland.

[2] Chen, P.P.-S., 1976. *The Entity-Relationship Model* — Toward a Unified View of Data. *ACM Transactions of Database Systems*, 1976.

[3] Hinke, T.H. and Delugach, H.S. 1993. AERIE: An Inference Modeling and Detection Approach For Databases, in *Database Security VI: Status and Prospects*, B.M. Thuraisingham and C.E. Landwehr, Editors. 1993, North-Holland.

[6] These were previously called layered knowledge chunks.

[4] Morgenstern, M., 1988. Controlling Logical Inference in Multilevel Database Systems. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, 1988.

[5] Morgenstern, M., *1987*. Security and Inference in Multilevel Database and Knowledge-Base Systems, In *Proceedings of SIGMOD (ACM Special Interest Group on Management of Data)*, 1987.

[6] Hinke, T.H., 1988. Database Inference Engine Design Approach. In *Proceedings IFIP Working Group 11.3 Workshop on Database Security*, 1988.

[7] Hinke, T.H., 1988. Inference Aggregation Detection In Database Management Systems. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, 1988. Oakland, CA: IEEE.

[8] Denning, D.E., et al., 1988, The SeaView Security Model. In *Proceedings 1988 IEEE Computer Society Symposium on Research in Security and Privacy*, 1988.

[9] Schaefer, M., 1995. Operating System Security Course, 1995.

[10] Weissman, C., 1995. *Essay 11: Penetration Testing, in Information Security: An Integrated Collection of Essays*, M.D.A.a.S.J.a.H.J. Podell, Editor, 1995.

[11] Bisby, R. and Hollingworth, D., 1978. *Protection Analysis Project Final Report*, 1978, USC Information Sciences Institute: Marina Del Rey, CA.

[12] Center, N.C.S., 1985. *Department of Defense Standard: Department of Defense Trusted Computer System Evaluation Criteria*, 1985, National Computer Security Center.

[13] Hinke, T.H., 1990. Database Inference Engine Design Approach, in *Data-base Security II: Status and Prospects*, C.E. Landwehr, Editor, 1990.

[14] Binns, L.J., 1993. Inference Through Secondary Path Analysis, In *Database Security, VI: Status and Prospects*, B.M. Thuraisingham and C.E. Landwehr, Editors, 1993, North-Holland.

[15] Binns, L.J., 1992. Inference Through Secondary Path Analysis. In *Proceedings of the Sixth IFIP 11.3 Working Conference on Database Security*, 1992.

[16] Binns, L.J., 1993. Implementation Considerations for Inference Detection: Intended vs. Actual Classification. In *Proceedings of the IFIP WG 11.3 Seventh Annual Working Conference on Database Security*, 1993.

[17] Binns, L.J., 1994. Implementation Considerations for Inference Detection: Intended vs. Actual Classification, *in Database Security, VII: Status and Prospects*, T.F. Keefe and C.E. Landwehr, Editors, 1994, North-Holland.

[18] Qian, X., et al., 1993. Detection and Elimination of Inference Channels in Multilevel Relational Database Systems. In *Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.

[19] Su, T.-A. and Ozsoyoglu, G., 1991. Multivalued Dependency Inferences in Multilevel Relational Database Systems. In *IEEE Transactions on Knowledge and Data Engineering*, 1991.

[20] Su, T.-A. and Ozsoyoglu, G., 1990. Multivalued Dependency Inferences in Multilevel Relational Database Systems, in *Database Security III: Status and Prospects*, D.L.S.a.C.E. Landwehr, Editor, 1990, North-Holland.

[21] Hinke, T.H., Delugach, H.S. and Chandrasekhar, A., 1995. A Fast Algorithm For Detecting Second Paths in Database Inference Analysis. *Journal of Computer Security*, 1995, 3(2,3): pp. 147-168.

[22] Elmasri, R. and Navathe, S.B., 1984. *Fundamentals of Database Systems, Second Edition*, 1994, Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc.

[23] Ullman, J.D., 1988. *Principles of Database and Knowledge-base Systems, Volume 1*. 1988.

[24] Bell, D.E. and LaPadula, L.J., 1976. *Secure Computer Systems: Unified Exposition and Multics Interpretation*, . 1976, MITRE Corporation: Bedford, MA.

[25] Garvey, T.D., 1993. *SRI RADC Database*, . 1993, SRI International.

[26] Lunt, T.F., 1988. Toward a Multilevel Relational Data Language. In *Proceedings of the Fourth IFIP Aerospace Computer Security Applications Conference*, 1988.

[27] Dillon, W.R. and Goldstein, M., 1984. *Multivariate Analysis: Methods and Applications*. 1984: John Wiley & Sons.

[28] Denning, D.E., Denning, P.E. and Schwartz, M.D., 1979. The Tracker: A Threat to Statistical Database Security. *ACM Transactions of Database Systems*, 1979.

[29] Buczkowski, L.J., 1990. Database Inference Controller, in *Database Security III: Status and Prospects*, C.E. Landwehr, Editor. 1990, North-Holland.

[30] Hinke, T.H., Delugach, H.S. and Wolf, R.P., 1997. *A Framework for* Inference-Directed Data Mining, in *Database Security Volume X: Status and Prospects*, P. Samarati and R.S. Sandhu, Editors. 1997, Chapman & Hall. pp. 229-239.

[31] Rumbaugh, J., Jacobsen, I. and Booch, G., 1988. "*Unified Modeling Language Reference Manual*", 1998.

[32] Rath, S., et al., 1996. A Tool for Inference Detection and Knowledge Discovery in Databases, In *Database Security IX: Status and Prospects*, D.L. Spooner, S.A. Demurjian, and J.E. Dobson, Editors. 1996, Chapman & Hall.

[33] Marks, D.G., 1994. An Inference Paradigm. in *Proceedings of the 17th National Computer Security Conference*, 1994.