COMMUTATIVE FILTERS FOR REDUCING INFERENCE THREATS IN MULTILEVEL DATABASE SYSTEMS

Dorothy E. Denning

SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025.

Disclosure of classified data in multilevel database systems is threatened by direct user access, user inference, Trojan Horse release, and Trojan Horse leaks. Earlier work showed how the problems of direct user access and Trojan Horse release can be solved by using a trusted filter and cryptographic checksums, but left the problems of inference and leaks open. We now show how the problem of user inference can be solved with the concept of a commutative filter that ensures that the result returned to a user is equivalent to one that would have been obtained had the query been posed against an authorized view of the database. The technique allows query selections, some projections, query optimization, and subquery handling to be performed by the database system. It does not solve the Trojan Horse leakage problem.

INTRODUCTION

Consider a multilevel relational database system, which handles data of different security classifications and provides shared access to users with different clearances. The *disclosure* requirement of multilevel security states that if cd is the classification of data d (security level plus compartment), and cu the clearance of user u, then disclosure of d to u is authorized only if $cu \geq cd$, where $\geq$ is the usual partial ordering relation on security levels and compartments. We assume that all data is labelled with its security classification, and that labelling is either at the relation, tuple, attribute, or element level.

Ignoring for the moment the problem of enforcing multilevel security, there are four categories of threats to the disclosure of unauthorized data. The first two of these are initiated by a user without assistance from the database system; the other two normally require assistance from a Trojan Horse in the database system, although they could also arise through faults in the system. The first and third represent direct disclosures; the second and fourth indirect disclosures.

1. **Direct Access.** Here the user queries the database system for unauthorized data d, and the system responds with d, correctly labelled with its classification.

2. **Indirect Access by Inference.** Here the query is for authorized data, but the data returned is a function of unauthorized data d in such a way that the user can infer d from the response.

3. **Trojan Horse Direct Release.** Here a Trojan Horse in the database system causes unauthorized data d to be released with an incorrect classification label that effectively makes the data appear to be authorized to the user. This may be achieved either by changing the label on d, by placing d in a record (or field) with the lower classification, or by failing to update the label on d if its classification increases.

4. **Trojan Horse Leakage.** Here the Trojan Horse indirectly leaks unauthorized data d by encoding it in authorized data returned to the user. Whereas the result returned by the database system appears to answer the user's query, it actually answers a different query, namely one for d.

All of the above forms of disclosure could arise from accidental as well as malicious actions on the part of a user. For example, a user could request a set of records, not realizing that some of the records in the set are classified higher than his clearance permits. Although we are primarily concerned with malicious attacks, we must also protect against accidental disclosures.

Preprint of paper to appear in the Proceedings of 1985 Symposium on Security and Privacy, Oakland, California, April 1985.

Turning now to the problem of protecting against unauthorized disclosure, we observe that disclosure by direct access (type 1 threat) can be prevented with database access controls that compare the classification of the data with the clearance of the user before releasing the data. Although most commercial database systems do not provide this capability, including this capability in a relational system would be relatively straightforward.

Unfortunately, adding simple access controls to a database system does nothing to prevent the other three types of disclosure. Of particular concern are Trojan Horses in the database system that release classified data improperly labelled (type 3 attacks), thereby circumventing the access checks. If the database system is "untrusted" (i.e., not verified with respect to its security properties), installing such a Trojan Horse in the code would seem to be relatively straightforward.

The seriousness of this Trojan Horse threat combined with the difficulties of designing large verifiable systems led researchers to look for solutions that do not require verification of a complete database system. The solution that emerged uses a trusted (verified) and isolated filter (guard), which interfaces with the untrusted database system as illustrated in Figure 1. The filter is a reference monitor responsible for enforcing the requirements for multilevel security. The filter labels the data at the relation, record, attribute, or field level (or some combination), and binds the data to its classification by computing a cryptographic checksum over the data and its label using a secret key. The cryptographic checksums make it practically impossible for the untrusted database system to change the data or the classification labels. The checksum is recomputed when the data is retrieved in order to authenticate the data and its classification. If the response to a query includes data that the user is not cleared to see, the filter simply removes that data from the response returned to the user.

Figure 1 shows two users, with low and high clearances respectively, with shared access to the database. In practice, any number of users with different clearances could have shared access. Each user (or group of users with identical clearances) is connected to the filter through a private interface process (untrusted front end) that performs pre- and post-processing of queries. These interface processes must be isolated from one another, e.g., on separate machines.

The basic idea, due to Roger Schell, has been applied by Anderson to the RECON bibliographic system, which is essentially a record storage and retrieval system with record level classification[1]. The idea was suggested by Weissman's group at the 1982 Woods Hole Summer Study on Multilevel Data Management Security[2] as a

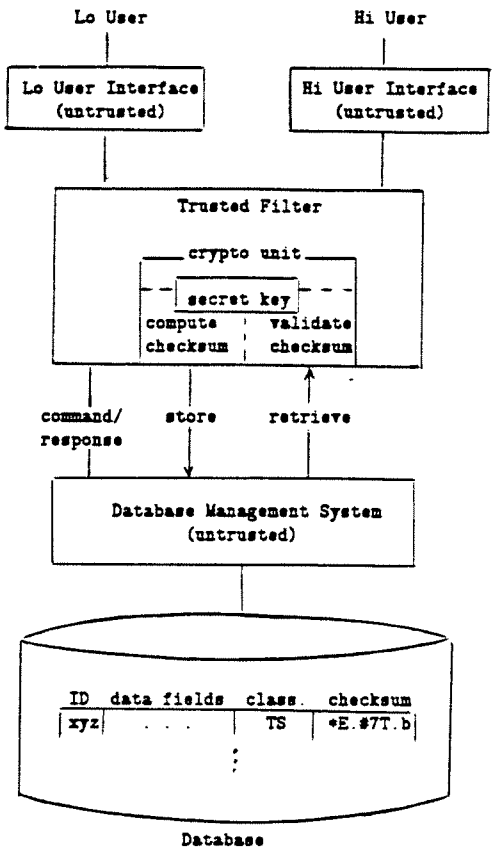


Figure 1. Multilevel Database Protected by Filter and Cryptographic Checksums.

possible method of providing multilevel database security for more general database systems. Since then, we have shown how labels and cryptographic checksums could be securely applied down to the element level[3] and have examined the implementation and security of the approach[4]. The approach has also been studied by Graubart[5].

These studies showed that whereas the approach could effectively eliminate direct disclosure (type 1 and 3 threats), including the return of falsely labelled data by Trojan Horses, it did not eliminate either user inference threats (type 2) or Trojan Horse leaks (type 4). The inference threat was found to be particularly serious because it is often simple to perform and requires no assistance from a Trojan Horse. It appeared that the only secure way of dealing with the threat is for the filter to see all data used to answer a query so that it can remove any element not authorized to the user from the user's view of the database.

Based on this observation, we concluded that the database system cannot perform selections and projections, handle subqueries, do query optimization, or perform statistical computations; these functions must be placed either in the filter or the user interfaces, effectively reducing the database system to a record storage and retrieval system[4]. It thus appeared that the general approach of using a trusted filter with cryptographic checksums is well suited for record storage and retrieval applications such as RECON, but not for general purpose databases.

The objective of this paper is to show how the user inference threat (type 2) can be practically eliminated with a minimal amount of support in the filter. The approach allows the untrusted database system to perform all selections, some projections, subquery handling, and query optimization. Section 2 reviews the nature of the inference threat, showing that attacks can be simple yet devastating. Section 3 presents our solution, which is based on the concepts of authorized views and commutative filters. Section 4 discusses the remaining threat, Trojan Horse leaks. Section 5 concludes.

THE INFERENCE PROBLEM

The inference problem of concern here arises when the classification of the response returned by a query permits the data to be released to the user even though the data used to form the response is not authorized to the user. Under many circumstances, the user can infer the unauthorized data from the authorized response. To make this concrete, we give two examples. We assume that the data is SECRET and TOP-SECRET collateral (not compartmented), and that the user making the queries is cleared only to SECRET.

Example 1. Consider a relation EMPLOYEE with attributes NAME and ID, which are classified SECRET, and attribute PROJ, which is classified TOP-SECRET (thus classification is by attribute). The following query returns SECRET data, but reveals TOP-SECRET data, namely whether there is a project called GEMINI and, if so, which employees are working on the project.

```
RETRIEVE EMPLOYEE.NAME
WHERE EMPLOYEE.PROJ = "GEMINI"
```

One can argue that a SECRET user should not even be aware of the existence of the TOP-SECRET attribute PROJ, whence such a query would never be posed. Indeed, the filter should hide the existence of unauthorized data. The point of the example is that knowing the existence of unauthorized data should not allow a user to obtain the data.

The example shows the problems that can arise when the database system performs selections and projections, and the filter sees only the results of these operations. If the database system instead returns the complete EMPLOYEE records to the filter, without doing the selection on PROJ code or projection onto NAME, the filter could delete the PROJ field, and securely pass the result back to the user interface (where the selection would no longer even be feasible). Alternatively, the filter could observe that the query named an attribute (PROJ) that was unauthorized to the user and abort the query (without passing it to the database system). This latter approach will be an important part of our proposed solution.

It is interesting that if classification is by record rather than by attribute, where records with certain PROJ codes including GEMINI are TOP-SECRET, then the query does not pose an inference problem. This is because the filter will remove all TOP-SECRET records regardless of whether PROJ is GEMINI. Hence, no information in TOP-SECRET records can be inferred from the response. Note, however, that some information may be inferred still since a null response from the filter reveals either that GEMINI is TOP-SECRET, that no employee belongs to the project, or that GEMINI is not in fact a valid code. There is little that can be done to protect against this type of inference short of classifying the entire relation as TOP-SECRET.

Classification by record or relation does not by itself solve the inference problem, however, as illustrated by the next example.

Example 2. Consider a database with two relations: EMPLOYEE, with attributes NAME and ID; and TRIPS, with attributes ID and DEST (destination), where the employee ID field in the two relations can be joined. Suppose classification is by record, where all records in EMPLOYEE are SECRET, and all records in TRIPS are SECRET unless DEST is "RHODESIA", in which case they are TOP-SECRET. The following query returns records from EMPLOYEE, which are SECRET, although it reveals TOP-SECRET information, namely the employees that went to Rhodesia:

```
RETRIEVE EMPLOYEE.NAME
WHERE EMPLOYEE.ID = TRIPS.ID
   AND TRIPS.DEST = "RHODESIA"
```

This query is a standard "select-project-join", which would normally be optimized and performed in one step by a database system. If for security the database system must return all the data needed to answer the query to the filter, then the database system could not be allowed to perform the standard optimization trick of selecting on DEST before doing the join to reduce the volume of data

to be joined. Our proposed solution will allow the database system to perform the optimized join, but require that it return the record classifications from the joined records in both EMPLOYEE and TRIPS that satisfy the selection formula over DEST.

SOLUTION TO INFERENCE PROBLEM

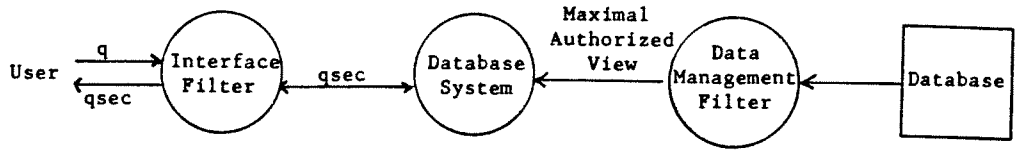
Observe that if each query is made against a view of the database consisting only of that data authorized to the user (i.e., all unauthorized data is effectively deleted from the database), then unauthorized data cannot be inferred from the response. Observe further that partitioning the data into authorized vs. unauthorized sets for a given user is straightforward since classification is applied to data objects (relations, records, attributes, or elements), and it is a simple test to determine whether an object's classification is less than or equal to a user's clearance.

We can imagine implementing authorized views by inserting a data management filter between the database management system and the stored data on disk. This filter would ensure that all data returned from the disk were authorized to the user making the query; unauthorized records and attributes would be deleted, and unauthorized elements replaced by nils. The data management filter would know the secret key used by the user interface filter to compute checksums so that it could authenticate retrieved records. The approach resembles the two kernel approach proposed by Downs and Popek[6].

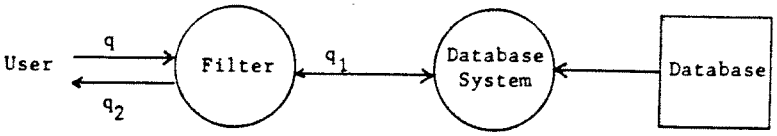
Figure 2a illustrates. For a given query q, the result returned to the user is equal to the result of a query *qsec*, which is q applied to the user's *maximal authorized view*, i.e., that subset of the database consisting of all data authorized to the user, but no unauthorized data.

The use of a data management filter was suggested by Tad Taylor and Bret Hartman, who observed that it could solve the Trojan Horse problem as well as the inference problem since the database system would see the maximal authorized view with respect to the user. Unfortunately, the approach assumes that the database system does not mix user views in its internal workspace; if it can mix the data in a TOP-SECRET view with that in a SECRET view, for example, then leaks and inferences may be possible. Such mixing might arise when the database system is simultaneously accessed by users with different clearances, but could also arise with sequential accesses. To ensure that such mixing does not occur would require that some trust be placed in the database system, which is what the filter approach is designed to eliminate.

We propose instead an indirect implementation of the principle, which we call the *authorized view equivalence scheme*, whereby the filter serves to make the result returned from a user's query q equal to what would be obtained had the query been applied to the maximal authorized view, that is, equal to *qsec*. Figure 2b illustrates how the filter obtains a result equal to *qsec* in two steps: Step 1 poses a query q_1 to the database system; q_1 is the original query q, augmented as necessary



a) Maximal Authorized View Approach.



b) Commutative Filter Approach (Authorized View Equivalence).

Figure 2. Preventing User Inferences with Authorized Views and Commutative Filters.

to include classification and checksum fields. Step 2 filters the response from q_1 to obtain a result for a query q_2 that is equivalent to *qsec*; the result of q_2 is returned to the user.

The approach effectively creates a filter that is commutative with the database management system; that is, the operations of removing unauthorized data and computing the response to a query are commutative. We emphasize, however, that this is true only to the extent that the database system does not have a Trojan Horse that leaks data.

We will now show how the scheme can be implemented when data is classified by record, attribute, or data element (or some combination). For clarity, we develop the approach for three types of increasingly complex queries: select only, select-project, and select-project-join. Generalizing the approach to more complex queries over more than two relations, including nested queries, is straightforward. The approach is not described in terms of the elementary relational operators since our objective is for the filter to pass complete queries to the database system for processing. For each type of classification and each type of query, we show the secure version *qsec* for an arbitrary query q, the query q_1 presented to the database system, and the query q_2 corresponding to the result returned to the user.

All queries are expressed in the relational algebra using the relational operators σ (selection), Π (projection), and $\times$ (join). Properties of the relational algebra are used to prove that q_2 is equivalent to *qsec* (e.g., see Ullman[7] for a description of the relational algebra operators and their properties).

Classification by Record

Let R be a relation, and let C denote an attribute that gives the classification of a record (tuple) of R. Let *cu* be the clearance of the user making the query. The maximal authorized view of R is thus given by

$$AUTH\text{-}VIEW\text{-}R = \sigma_{C \leq cu}(R) .$$

We now consider the three classes of queries.

Select Only Query

Consider the following query

RETRIEVE $R.All$ WHERE F,

where "*All*" denotes all attributes in R, and F is a selection formula over the values of the attributes. This query is expressed in the relational algebra as

$$q = \sigma_F(R) .$$

The secure version of q is thus

$$qsec = \sigma_F(AUTH\text{-}VIEW\text{-}R) = \sigma_F(\sigma_{C \leq cu}(R)) .$$

To obtain a result equivalent to *qsec*, the filter first poses the user's query to the database system, and gets back the response:

$$q_1 = \sigma_F(R) .$$

The filter then selects out those records unauthorized to the user, using the checksums to check the integrity of the data and the classification labels, and returns:

$$q_2 = \sigma_{C \leq cu}(q_1) = \sigma_{C \leq cu}(\sigma_F(R)) ,$$

which is equivalent to *qsec* because selections commute. This means that as far as protecting against inference is concerned, the order in which selections are performed does not matter; records can be selected first by their classification and then by other criterion (formula F), or vice-versa. Thus, the database system can use index structures on the database to efficiently perform selections and reduce the volume of data that must be returned to the filter.

Select only queries correspond to the queries handled by the RECON guard; indeed, the RECON guard processes these queries in the same manner described here, returning q_2. Because q_2 is equivalent to *qsec*, the RECON guard is secure against user directed inference attacks. Of course, this does not mean that it is free from Trojan Horse leaks (see Section 4).

Select-Project Query

Consider now a select-project query on R:

RETRIEVE $R.A$ WHERE F,

where A is a subset of the attributes in R. Expressed in the relational algebra, this query becomes:

$$q = \Pi_A(\sigma_F(R)) ,$$

and its secure version is given by:

$$qsec = \Pi_A(\sigma_F(\sigma_{C \leq cu}(R))) .$$

To obtain a result equivalent to *qsec*, the filter can again let the database system perform the selection using F, and then take the result and select by the classification C. If

the checksums are at the record level, the filter cannot, however, let the database system perform the projection onto A since complete records are needed for authentication; the projection must be done in either the filter or user interface. If the checksums are at the element level, the database system can project out data attributes, but must not project out the classification label C*. Because we are interested in knowing the minimal amount of data that must be returned to the filter for processing, we will pursue the case of element level checksums. Then the query sent to the database is**

$$q_1 = \Pi_{A,\, C}(\sigma_F(R)) .$$

The filter then selects out the records authorized to the user and projects onto A, thereby returning

$$\begin{aligned} q_2 &= \Pi_A(\sigma_{C \le cu}(q_1)) \\ &= \Pi_A(\sigma_{C \le cu}(\Pi_{A,\, C}(\sigma_F(R)))) \\ &= \Pi_A(\sigma_{C \le cu}(\sigma_F(R))) \\ &= qsec . \end{aligned}$$

This shows that the database system can perform projections securely, thereby reducing the volume of data returned to the filter. This would be important in applications where the communication link between the filter and the database system is a potential bottleneck.

Select-Project-Join Query

Finally, we consider a query which involves the joining of two relations:

RETRIEVE $R.A$, $S.B$ WHERE F ,

where A is a subset of the attributes in relation R, B is a subset of the attributes in S, and F is an arbitrary selection formula over the attributes of the two relations, which would include clauses for joining pairs of attributes in R and S by equality (for equijoin, including natural join) or by some other relational operator. Either of the projected subsets, A or B, may be null. In the relational algebra, the query is given by

$$q = \Pi_{R.A,\, S.B}(\sigma_F(R \times S)) .$$

Since the secure view for this query is obtained by selecting the authorized records from R and S independently before performing the join, we have

$$qsec = \Pi_{R.A,\, S.B}(\sigma_F(\sigma_{R.C \le cu}(R) \times \sigma_{S.C \le cu}(S))) .$$

To obtain a result equivalent to $qsec$, we follow the same strategy as for select-project queries, namely, of obtaining the classification attributes from both relations along with the other requested attributes. The queries q_1 and q_2 are thus:

$$q_1 = \Pi_{R.A,\, R.C,\, S.B,\, S.C}(\sigma_F(R \times S))$$

$$\begin{aligned} q_2 &= \Pi_{R.A,\, R.B}(\sigma_{R.C \le cu \,\wedge\, S.C \le cu}(q_1)) \\ &= \Pi_{R.A,\, R.B}(\sigma_{R.C \le cu \,\wedge\, S.C \le cu} \\ &\qquad (\Pi_{R.A,\, R.C,\, S.B,\, S.C}(\sigma_F(R \times S)))) \\ &= \Pi_{R.A,\, R.B}(\sigma_{R.C \le cu \,\wedge\, S.C \le cu}(\sigma_F(R \times S))) \\ &= \Pi_{R.A,\, R.B}(\sigma_F(\sigma_{R.C \le cu}(R) \times \sigma_{S.C \le cu}(S))) \\ &= qsec . \end{aligned}$$

Note that the filter must examine the classification fields from both relations, even if all attributes are projected out of one of the relations. This is necessary to prevent the type of inference illustrated by Example 2 in Section 2, where the records in the returned relation were classified lower than the records in the relation joined to it.

Classification by Attribute

Let $c(A_i)$ denote the classification of attribute A_i in relation R. The set of attributes authorized to a user u with clearance cu is defined by

$$R.AUTH = \{A_i \in R \mid c(A_i) \le cu\} .$$

Thus, the maximal authorized view of R is

$$AUTH\text{-}VIEW\text{-}R = \Pi_{R.AUTH}(R) .$$

Again, we examine the three classes of increasingly complex queries:

Select Only Query

Consider again the query

$$\begin{aligned} q &= \text{RETRIEVE } R.All \text{ WHERE } F \\ &= \sigma_F(R) . \end{aligned}$$

With attribute level classification, the secure version of q becomes

---

* In addition, the checksum fields are needed to authenticate the data, and the record identifier is needed to compute the cryptographic keys (using the field authentication technique we proposed in[3]); these fields are not shown in the augmented queries since they are not relevant to the inference problem.

** Strictly speaking, we should write "$A \cup \{C\}$" instead of "A, C" since C could be in the set A; we will use "," since it is notationally simpler.

$$qsec = \sigma_F(\Pi_{R.AUTH}(R)) .$$

If we adopt a strategy similar to the one taken for record level classification, the filter would obtain a result equivalent to *qsec* by first presenting the user's query to the database system:

$$q_1 = \sigma_F(R) ,$$

and then filtering out the unauthorized attributes by computing:

$$q_2 = \Pi_{R.AUTH}(q_1) = \Pi_{R.AUTH}(\sigma_F(R)) .$$

But in this case q_2 is equivalent to *qsec* if and only if the formula F names only attributes in *R.AUTH*; otherwise, the projection onto *R.AUTH* does not commute with the selection on F. Thus, to achieve equivalence, the filter must check the attributes named in F to determine whether they are in *R.AUTH*, disallowing the query if they are not. This requires more processing in the filter than for record level classification since the filter must obtain the attributes named in a query explicitly or implicitly (through *R.All*) and their classifications. The attribute classifications could be stored in the database system (e.g., in a relation ATTRIBUTES that gives the type and classification of each attribute in the database), and checksums used to guarantee their integrity.

Classification by attribute may also require additional processing in the user interface if the attributes returned do not include all of the ones expected (i.e., because they are not authorized to the user). The interface must be able to receive shorter records with fewer fields and know which fields have been deleted.

Select-Project Query

For a query

$$\begin{aligned} q &= \text{RETRIEVE } R.A \text{ WHERE } F \\ &= \Pi_A(\sigma_F(R)) , \end{aligned}$$

the secure version is

$$qsec = \Pi_A(\sigma_F(\Pi_{R.AUTH}(R))) .$$

The query from the filter to the database is again $q_1 = q$, and the response from the filter to the user

$$q_2 = \Pi_{R.AUTH}(q_1) = \Pi_{R.AUTH}(\Pi_A(\sigma_F(R))) .$$

As before, q_2 is equivalent to *qsec* if and only if the formula F names only attributes in *R.AUTH*; the filter must check this property.

Select-Project-Join Query

For a join query over relations R and S:

$$\begin{aligned} q &= \text{RETRIEVE } R.A, R.S \text{ WHERE } F \\ &= \Pi_{R.A,\, S.B}(\sigma_F(R \times S)) , \end{aligned}$$

the secure version is

$$qsec = \Pi_{R.A,\, S.B}(\sigma_F(\Pi_{R.AUTH}(R) \times \Pi_{S.AUTH}(S))) .$$

Again the query from the filter to the database system is $q_1 = q$, and the result returned to the user is

$$\begin{aligned} q_2 &= \Pi_{R.AUTH,\, S.AUTH}(q_1) \\ &= \Pi_{R.AUTH,\, S.AUTH}(\Pi_{R.A,\, S.B}(\sigma_F(R \times S))) , \end{aligned}$$

which is equivalent to *qsec* if and only if the formula F names only attributes in *R.AUTH* and *S.AUTH*. This implies that the join itself, which is expressed by one or more clauses in F, must be over authorized attributes.

Classification by Element

For each data attribute A_i in a relation R, we assume there is a corresponding classification attribute C_i that gives the classification of the data element stored in the field for A_i in the records of the relation.

The maximal authorized view of R consists of all data elements whose classification is less than or equal to the user's clearance *cu*; unauthorized data elements and their classifications are effectively replaced by a "nil" value (meaning "undefined"), which itself must be unclassified. Since this is difficult to express formally in the relational algebra, but simple to grasp intuitively, we will not give a formal definition here.

Of course, the presence of a "nil" value signals the user that the missing data is either undefined or unauthorized. If undefined is considered less likely than unauthorized, the user can thus deduce something about the classification of the suppressed data. For example, if classification is not compartmented, then a SECRET user can deduce that suppressed data is at least TOP-SECRET. Moreover, the domain of values may be such that the data itself can be deduced (e.g., if there is only one TOP-SECRET value possible for a given attribute). In general, the presence of a nil can reveal up to one bit of information about the data suppressed.

Now, if the filter applies this definition of maximal authorized view by replacing unauthorized values with nils, security is achieved only if selections are performed by the filter instead of by the database system. To see why, consider again the EMPLOYEE relation of Example 1, but now suppose that the data is classified by element. Suppose also that all NAMEs are SECRET, but that PROJ is either SECRET or TOP-SECRET, depending on its value; in particular, the PROJ code GEMINI is TOP-SECRET and the code APOLLO is SECRET. Consider the query

```
RETRIEVE EMPLOYEE.NAME,
    EMPLOYEE.PROJ
WHERE EMPLOYEE.PROJ="GEMINI"
    OR EMPLOYEE.PROJ="APOLLO"
```

If the database system performs the selection on PROJ, it will return something such as:

NAME, CLASS	PROJ, CLASS
Baker, SECRET	GEMINI, TOP-SECRET
Jones, SECRET	APOLLO, SECRET
Smith, SECRET	GEMINI,TOP-SECRET

The filter will then delete the TOP-SECRET data, and return

NAME, CLASS	PROJ, CLASS
Baker, SECRET	nil, nil
Jones, SECRET	APOLLO, SECRET
Smith, SECRET	nil, nil

The user, however, can deduce that the nil code is GEMINI, that GEMINI is TOP-SECRET (in case this was not known), and that employees Baker and Smith have these codes. This inference could be prevented by moving all selections from the database system to the filter, which would disallow selections over unauthorized data. However, since we are interested in minimizing the amount of query processing in the filter, we seek an approach that allows the database system to perform selections.

The example illustrates an important principle about classification of data: data is classified by its context, that is, by its association with other data, and not by its inherent value. The name GEMINI by itself is certainly not classified. It becomes classified only by its association with a particular attribute (in this case PROJ), and possibly also by its association with a particular record (employee). Two policies that would lead to element level classification of PROJ codes are :

1. **Single Attribute Association**. The domain of PROJ codes is partitioned into TOP-SECRET values, SECRET values, etc; thus, the value in a particular record is classified according to the block to which it belongs.

2. **Multiple Attribute Association**. The classification of a PROJ code is determined by the employee having that code; note that this is equivalent to partitioning the domain of NAME into those with TOP-SECRET codes, those with SECRET codes, etc. The names themselves, however, may all be at the SECRET level so that users cleared only at the SECRET level can access them. More complex classification policies can also be envisaged - for example, where classification is determined by attributes in other relations that are associated with an employee (through joins).

Note that classification by attribute is a special case of single attribute association where all values belong to the same block; classification by record is a special case of multiple attribute association where all fields in a record are uniformly classified by their association with the record.

Now, because of these associations, the database system can be allowed to perform selections only if the filter does not return nils for unauthorized data. This means that the filter must either suppress complete attributes (columns) or records (rows) so that the relation returned does not have "holes".

With the suppressed attribute approach, the filter would remove any attribute that has an unauthorized value in some record selected by the query. This requires a two-pass algorithm over the records: the first pass to authenticate the records and determine which (if any) fields have unauthorized data; the second to delete those fields. In the preceding example, the attribute PROJ would be deleted since it is referenced by the query and some of the records selected contain TOP-SECRET codes.

With the suppressed record approach, the filter would delete those records containing an unauthorized value in one of the requested fields. This can be implemented with a one-pass algorithm over the records where the decision to keep or delete a record is made as each record is authenticated. In the preceding example, the records for Baker and Smith would be deleted; thus, the user would be unable to deduce anything about their PROJ codes.

Suppressing records is more efficient and sensible to implement than suppressing attributes since the data is returned from the database system by record. Moreover, we would expect users to prefer this approach since they will get data for requested attributes. But there is another important reason for adopting record suppression: attribute suppression does not prevent inference with multiple attribute associations; that is, when data associated with one attribute (e.g., PROJ) is classified by its association with another attribute (e.g., NAME). In the preceding example, suppressing the PROJ field would not prevent inference of a TOP-SECRET PROJ code if the query selected only on the basis of a single TOP-SECRET value (e.g., the WHERE clause were simply "PROJ="GEMINI"). The filter would return the NAME field of all employees assigned to GEMINI, and deleting the PROJ field would not prevent a user from inferring the deleted value. Record suppression, by comparison, thwarts this inference because the user is not even aware of the existence of records containing the PROJ code GEMINI, let alone which employees have this value.*** Attribute suppression can prevent inferences with single attribute classification since the filter could know that GEMINI is a TOP-SECRET value of PROJ and disallow the selection. This could be done with record suppression as well, but is unnecessary since selections over unauthorized data have no effect (since they commute with the filter's selection to remove records with unauthorized data).

We therefore adopt record suppression and define an authorized view to be one whose records contain only authorized data. Applying this definition to a complete relation, however, will restrict much more data than necessary, since any record containing even one unauthorized field would be deleted, even if the query does not include that field. For example, if the user's query is simply

RETRIEVE EMPLOYEE.NAME ,

then all employee names can be returned securely, even if their PROJ codes are unauthorized to the user. Thus, we shall instead apply the definition of authorized view to a projection of a relation onto the set of attributes named either explicitly or implicitly (through *All*) in a query; as before, this includes any attributes named in a selection formula.

*** There is an exception. If GEMINI is the only TOP-SECRET code, then a user can deduce which employees have this code by requesting the set of all employee NAMEs and the set of all (NAME, PROJ) pairs and then computing the set difference over the names.

For a subset A of the data attributes of a relation R, let C denote the corresponding classification attributes. We shall write $C \leq cu$ as shorthand for the "and" over all terms $C_i \leq cu$ for C_i in C:

$$\wedge\{C_i \leq cu \mid C_i \in C\} .$$

The authorized view of relation R projected onto data attributes A with classification attributes C is thus:

$$AUTH\text{-}VIEW\text{-}R\text{-}A = \sigma_{C \leq cu}(\Pi_{A,C}(R)) .$$

Now, this definition of authorized view admits exactly the same set of data admitted by the original definition with the nils. This is because the set of all projections onto all possible subsets of attributes of R gives exactly the set of authorized data. In particular, for each record, there is exactly one subset A of attributes for which a projection onto A will return all authorized data in that record; every superset of A will result in that record being deleted, allowing the user to deduce which fields are unauthorized (i.e., nil). Thus, the new definition gives the maximal authorized view.

But unlike the original definition, the definition here leads directly to authorized view equivalence. We now show how this is done.

Select Only Query

The query

$$\begin{aligned} q &= \text{RETRIEVE } R.All \text{ WHERE } F \\ &= \sigma_F(R) \end{aligned}$$

has no projections, so its secure version is given by

$$\begin{aligned} qsec &= \sigma_F(AUTH\text{-}VIEW\text{-}R\text{-}All) \\ &= \sigma_F(\sigma_{C\text{-}All \leq cu}(R)) , \end{aligned}$$

where *C-All* denotes all classification attributes.

The query from the filter to the database is the user's query, $q_1 = q = \sigma_F(R)$, and the result returned by the filter is:

$$\begin{aligned} q_2 &= \sigma_{C\text{-}All \leq cu}(q_1)) \\ &= \sigma_{C\text{-}All \leq cu}(\sigma_F(R)) \\ &= qsec . \end{aligned}$$

Select-Project Query

Consider now the query

$$\begin{aligned} q &= \text{RETRIEVE } R.A \text{ WHERE } F \\ &= \Pi_A(\sigma_F(R)) . \end{aligned}$$

Letting A' denote the attributes A plus those named in F, and C' the corresponding classification attributes, the secure version of q is

$$qsec = \sigma_F(AUTH\text{-}VIEW\text{-}R\text{-}A') = \sigma_F(\sigma_{C' \leq cu}(\Pi_{A',C'}(R))) .$$

The query presented by the filter to the database system is

$$q_1 = \Pi_{A',C'}(\sigma_F(R)) ,$$

and the result returned to the user is

$$q_2 = \Pi_A(\sigma_{C' \leq cu}(q_1)) = \Pi_A(\sigma_{C' \leq cu}(\Pi_{A',C'}(\sigma_F(R)))) = qsec .$$

Select-Project-Join Query

This case is a straightforward extension of select-project queries.

Summary of Filter Requirements

The following summarizes the processing steps in the filter for each type of classification.

Classification by Record

1. Filter must augment a query to include the classification, checksum, and possibly ID fields in each relation specified, even if all other attributes in that relation are removed (as in a join query where the requested attributes are all from one relation); this query is then presented to the database system.

2. Filter must check the classification fields in every record returned, suppressing unauthorized records and records that fail the authentication check.

Classification by Attribute

1. Filter must obtain the classification of each attribute named in a query either explicitly or implicitly; these could be obtained from the database system.

2. Filter must check the classifications of all attributes named in a selection (WHERE clause) to determine if the attributes are authorized to the user. If all attributes are authorized, the user's query is sent to the database system, augmented to include checksum fields and other fields needed for authenticating the data; otherwise, the query is disallowed.

3. Filter must delete all fields corresponding to unauthorized attributes from the records returned by the database system, and delete all that fails the authentication check.

Classification by Element

1. Filter must augment a query to include the data and classification field associated with every attribute named explicitly or implicitly in a query, plus the fields needed to authenticate the data; this query is then presented to the database system.

2. Filter must check the classification of every element in every record returned, suppressing those records that contain elements that are unauthorized or that fail the authentication check.

For all types of classification, the bulk of query execution can thus be handled by the database system. In particular, selections, joins, query optimization, subquery handling (for nested queries), and some projections can be managed by the database system. The main role of the filter is to augment the user's query to the database system to ensure that all relevant classification fields (and checksums) are returned (for record and element classification), and that all named attributes are either authorized (for attribute classification) or returned (for element classification). Of course, this requires that the filter have access to the record structures of the relations stored in the database, and that it know the query structure passed from the user interface so that it can access and manipulate it. The latter can be simplified if the interface parses the query first, and passes a parse tree (or equivalent structure) to the filter.

Unsolved Inference Problems

The authorized view equivalence scheme does not address all types of inference. These fall into four general classes:

1. **Aggregate Functions.** The filter cannot pass queries for sums, averages, counts, and other statistical (numerical, etc.) functions over aggregates of data to the database system for evaluation. This is because the filter would not know the classification of the data used to compute the query, and there would be no guarantee that unauthorized data could not be inferred from the computational results returned. Instead, the user interface must ask for the raw data and perform the computation itself. This means that the filter cannot, for example, release aggregate statistics with a security classification lower than that of the raw data; to do so, would require complex inference controls in the filter of the type used by census agencies (e.g., see[8] for a survey).

2. **Constraints**. Inferences may arise from constraints on the data imposed by integrity conditions or external factors if these constraints are not reflected in the classification policy. For example, consider a system where classification is by attribute. If an attribute B can be derived from a lower classified attribute A, then unauthorized data associated with B may be inferred from authorized data associated with A. To prevent this type of inference, data that derives other data stored in the database must be classified at least as high as the derived data. Since the rules for deriving data are often not explicit in the database, preventing this type of inference can be extremely difficult.

3. **Classification**. As noted earlier, a user can infer something about the classification of suppressed data if the user knows that the data exists in the system. Moreover, if there is only one possible classification code for the suppressed data, then the user can infer its exact classification. Of course, in general we would expect that users would not have enough information about data that is unauthorized to them to realize that it has been suppressed, or even to ask for it in the first place.

4. **Low Entropy Data**. Also noted earler, if unauthorized data has low entropy, inference may be possible by taking the set difference of a set of records and the same set constrained by a selection over unauthorized data, allowing the contents of the records in the difference to be deduced.

The last two categories of inference in particular point out the desirability to restrict a user's knowledge about the database to authorized data. If a user does not know about unauthorized data, including the names of unauthorized attributes or unauthorized values associated with multilevel attributes (i.e., where the data associated with the attribute is classified by element), these types of inference are less likely. Obviously, the filter should not provide this information to users in response to queries about the database schema. Assuming such "meta queries" are simply queries on the system relations of the database, then the techniques described in Section 3 will also protect the system data.

TROJAN HORSE LEAKS

The authorized view equivalence scheme thwarts user inferences where the database system is not party to the attempted subversion. Subversive attempts on the part of the database system to return unauthorized data directly (by modifying classification labels or by stuffing the data into authorized records) are prevented by the filter's checksums.

This leaves the threat of a Trojan Horse (TH) in the database system leaking unauthorized data by encoding it in authorized data. The threat is present with all three types of classification: record, attribute, and element.

It is difficult to assess the seriousness of this threat in a general way since the difficulty of performing such encodings depends on the structure and contents of the database. We observe, however, that to leak data, the following conditions must be satisfied:

1. There must be a conspiracy between the TH and the user receiving the data. This means that the person planting the TH in the database system must either be the user or else collaborate with the user.

2. Either the TH must regularly return unauthorized data, which the cognoscente recognize and know how to decipher, or else the user must have some way of signalling the TH to return unauthorized data. If the database system is not informed of the user originating a query, then the signal would have to be through some agreed upon query.

3. There must be a way of encoding the unauthorized data in the response to an agreed upon query for authorized data, and the user must know the encoding scheme. In particular, the TH must return a response that looks like the response to the user's query (i.e.,

consists of same set of attributes from the same relations), but which in fact answers a different (hypothetical) query (namely, one for the leaked data). In addition, either the user must know exactly what data elements are being returned and how to locate them in the response, or else the encoding must include enough information to identify the unauthorized data elements in the response.

4. Since the encoding will depend on the presence of authorized data satisfying certain constraints (e.g., having the same value as the unauthorized data), leakage will fail in the absence of such data. Thus, there should be a low signal (leaked data) to noise (authorized data) ratio.

5. There must be a way of recognizing unauthorized data in the database. If the classification labels are stored in the clear, then this is trivial. But if the labels are encrypted[3, 4], the TH would have to deduce classification by access patterns to the database, or else would require assistance from the user. Encrypting the labels, however, has the disadvantage of making it impossible for the database system to perform selections based on classification (to reduce the volume of data returned to the filter).

The following scenario illustrates how these conditions might be satisfied. The leaked data are integers between 1 and 1000.

1. The penetrator creates an unclassified relation VALS with an integer attribute, and inserts 1000 records into VALS covering the domain 1 to 1000.

2. The penetrator plants a TH in the database system by modifying the query processor to respond to the query "RETRIEVE VALS.All" by translating and executing a query which is encoded in the relation LEAK. The amount of code required for the TH will depend on the capabilities provided by the database system for dynamic query interpretation.

3. The penetrator creates a relation LEAK with the format used by the TH.

4. The penetrator inserts records for a query into LEAK that, when translated, leak classified data in the range 1 to 1000 by returning the records from VALS having the identical values. Thus, the data returned looks like a legitimate response to the penetrator's original query.

The TH used in this scenario is particularly devastating because it is not bound to any particular leakage. The penetrator has only to update the relation LEAK to obtain other unauthorized data. Thus, the TH behaves like a "universal Trojan Horse" which executes user-programmable leakage queries.

SUMMARY AND CONCLUSIONS

Disclosure of classified data in multilevel database systems is threatened by direct user access, user inference, Trojan Horse release, and Trojan Horse leaks. Earlier work showed how the problems of direct user access and Trojan Horse release can be solved by using a trusted filter and cryptographic checksums, but left the problems of inference and leaks open. The inference threat was found to be particularly troublesome because it did not require a Trojan Horse in the database system, and seemed to suggest that most query processing, including selections, could not be performed securely in the database system.

This paper has shown how the problem of user inference can be solved using authorized views and a commutative filter, which ensures that the result returned to a user is equivalent to one that would have been obtained had the query been posed against an authorized view of the database. The technique allows query selections, some projections, query optimization, and subquery handling to be performed by the database system.

We have not solved the Trojan Horse leakage problem. Whether this a serious shortcoming, or whether an acceptable solution might be found, is not known and requires further study.

Acknowledgments

The idea for this paper came while I was preparing my slides for the 1984 Symposium on Security and Privacy and realized that not all of the conclusions in my paper were justified; in particular, the inference problem could be managed without moving all selections and projections into the filter. A provocative discussion with Bret Hartman, Tad Taylor, and Kim Wilson at SRI following the symposium helped me clarify my ideas. A later discussion on Trojan Horses with Peter Denning and

Steve Lipner at the University of Newcastle upon Tyne led Peter to propose the universal Trojan Horse. Peter, Jim Anderson, and Richard Graubart provided many helpful comments on earlier versions of this paper. I am deeply grateful to the NSF for supporting this research through Grant MCS83-13650.

References

1. Anderson, J. P., "On the Feasibility of Connecting RECON to an External Network", Tech. report, James P. Anderson Co., March 1981.

2. Committee on Multilevel Data Management Security, "Multilevel Data Managment Security", Tech. report, Air Force Studies Board, National Research Council, 1982.

3. Denning, D. E., "Field Encryption and Authentication", *Proc. of CRYPTO 83*, Plenum Press, 1983.

4. Denning, D. E., "Cryptographic Checksums for Multilevel Data Security", *Proc. of the 1984 Symp. on Security and Privacy*, IEEE Computer Society, 1984, pp. 52-61.

5. Graubart, R. D., "The Integrity-Lock Approach to Secure Database Management", *Proc. 1984 Symp. on Security and Privacy*, IEEE Computer Society, 1984, pp. 62-74.

6. Downs, D. and Popek, G. J., "A Kernel Design for a Secure Data Base Management System", *Proc. 3rd Conf. Very Large Data Bases*, IEEE and ACM, New York, 1977, pp. 507-514.

7. Ullman, J. D., *Principles of Database Systems*, Computer Science Press, 1982.

8. Denning, D. E. and Schlorer, J., "Inference Controls for Statistical Database Security", *IEEE Computer*, Vol. 16, No. 7, July 1983, pp. 69-82.