Secure Databases: Protection Against User Influence

DAVID DOBKIN
University of Arizona
ANITA K. JONES
Carnegie-Mellon University
RICHARD J. LIPTON
Yale University

Users may be able to compromise databases by asking a series of questions and then inferring new information from the answers. The complexity of protecting a database against this technique is discussed here.

Key Words and Phrases: database, security, protection, information flow, inference, compromise, statistical query
CR Categories: 13.5, 4.33, 5.5

1. INTRODUCTION

In computer systems we encode information in databases. We often want to control what information a user can obtain from these databases. For example, we may wish to control the use of a census database so that, although it contains records describing individuals, only statistical information is available. No sequence of queries should be sufficient to deduce exact information about any individual described in the database. Determining and then enforcing a policy specifying what information in a database can be given in response to queries is the database security problem [6, 7].

Security is also an issue for operating systems; unfortunately, the solutions for operating systems are not sufficient to solve the database security problem. Most operating system protection mechanisms are "access control mechanisms" [5], that is, they enforce rules about who can perform what operation or access what

This work was supported in part by the Office of Naval Research under Grant N00014-75-C-0450, in part by the National Science Foundation under Contract DCR-75-07251, and in part by the National Science Foundation under Contract DCR-74-24193.

Authors' addresses: D. Dobkin, Department of Computer Science, University of Arizona, Tucson, AZ 85721; A. K. Jones, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213; R. J. Lipton, Department of Computer Science, Yale University, New Haven, CT 06520.

information. For example, users can access file objects via READ, WRITE, SORT, DELETE FILE, or APPEND. But different users may be permitted different access to individual files. While user *A* may be permitted to READ and WRITE File *X*, user *B* may be able only to READ and APPEND it. And user *A* may be permitted only to READ File *Y*, while user *B* may both READ and WRITE that one. There are two common schemes for effecting such protection mechanisms; one is the "authority list mechanism" used in most file systems (e.g., the MULTICS file system [11]) and the other is the capability mechanism [9, 11, 12].

In operating systems, protection mechanisms allow different users different access to an object; they allow some users to read part or all of the contents of a file and others to alter it in perhaps limited ways. We consider databases in which all users are essentially performing read access. An access control mechanism that only distinguishes between read and alter accesses is not useful. Thus the operating system approach is not sufficient for the database problem.

Another contrast between databases and operating systems concerns queries that involve many data elements. In the operating system, a complex operation can be broken down into a set of accesses to individual objects and each access permission determined independently of the others. In a database, a decision must be made whether the entire query should be permitted in the first place. This decision depends not only on the relationship of data elements being interrogated but also on the query history, the information that has already been divulged to the user.

Newer access control mechanisms take into account the flow of information out of one object and into another as part of the effect of an access. These access control mechanisms incorporate remembering the source from which information encoded in an object was derived [3, 4, 8]. Yet even such sophisticated mechanisms make no interpretation of the content of the database and have no notion of a history of information already given out. We conclude that such mechanisms are not appropriate tools for solving the database security problem.

Table I

Contributing group	Amount
Steel	270,000
Sugar	120,000
Oil	540,000
Democrats	186,000
Republicans	564,000
Independents	180,000
High favoritism	510,000
Low favoritism	174,000
Medium favoritism	246,000
Northeast	90,000
West	330,000
South	510,000

Let us now restate the database security problem: There is a set of data elements in the database called the UNKNOWN set that user U is not permitted to know. For some reason, perhaps as a result of previous queries, user U knows a set of data elements called the KNOWN set. Some elements in KNOWN may not be explicitly encoded in the database. User U asks a sequence of queries q_1, q_2, ..., q_n and enlarges his set of KNOWN data elements. The security of the database is compromised if the KNOWN and the UNKNOWN sets intersect.

We will now proceed by introducing an example to highlight the issues germane to database security.

Example. The following data describe fund raising for major political parties. $C1, \ldots, C9$ are specific contributors with the following attributes:

Contributor	Business area	Political leaning	Favoritism shown by administration	Geographic area
C1	Steel	Democrat	High	Northeast
C2	Steel	Republican	Medium	West
C3	Steel	Independent	Low	South
C4	Sugar	Democrat	Medium	Northeast
C5	Sugar	Republican	Low	Northeast
C6	Sugar	Independent	High	West
C7	Oil	Democrat	Low	South
C8	Oil	Republican	High	South
C9	Oil	Independent	Medium	West

Suppose that the only data that can be obtained from the database is the sum given by all contributors sharing a common attribute—contributions from the steel industry ($C1 + C2 + C3$) or contributions from those with Republican leanings ($C2 + C5 + C8$). The information that can be obtained from all possible queries is listed in Table I.

The political fund database is considered secure if the precise contribution of an inidividual cannot be determined. Is this database secure?

No, we can compute that $C1$ gave \$60,000. Values for contributors $C2, \ldots, C9$ can also be computed on the basis of the query responses listed in Table I.

We are interested in obtaining criteria that allow us to determine exactly when a database can be compromised.

2. BASIC CONCEPTS

We are interested in the question of determining the security of databases. We now define precisely what this means by presenting a general model. This model is an abstraction of the concept of database; we do not suggest that it be used in place of, say, the relational database model [1]. But we do propose this model as realistic for our discussion.

Definition. A *database* D is a function from $\{1, \ldots, n\}$ to N, the natural numbers. n is the number of elements or objects in the database; N is the set of possible attributes.

In our fund-raising example, $D(1)$ is \$60,000. $D(i)$ is the contribution of Ci. We will often use the following notation for databases. Instead of defining D explicitly

we just say that $\{d_1, \ldots, d_n\}$ is a database. We mean of course that $D(i) = d_i$ for $1 \leq i \leq n$.

We will now define "query" and "compromise."

Definition. Fix n as the number of objects in the database. A *query* q is a function of n variables. If $D = \{d_1, \ldots, d_n\}$ is a database and q is a query, then $q(D) = q(d_1, \ldots, d_n)$ is the result of the query q on the database D.

In our example $q(d_1, \ldots, d_9)$ is an allowed query provided

$$q(d_1, \ldots, d_9) = \sum_{k \in A} d_k$$

where A is a set of contributors that corresponds to an entry in Table I. Thus there are exactly 12 queries of this form.

A *security problem* has several components:

(1) A particular database $D = \{d_1, \ldots, d_n\}$ is given.
(2) A subset D_0 of D is given. We interpret $d_i \in D_0$ as meaning that d_i is known to the user before he begins his queries.
(3) A set of queries is given. We assert that not all sets of queries are allowed. (In Section 3 we restrict the "overlap" of queries.)

Given these components, we are to determine whether or not there is an allowed sequence of queries that can determine the value of some $d_i \notin D_0$. Thus a sequence of allowed queries $q_1, \ldots, q_m$ *compromises* a database provided there is an i such that, for any database D' with the same responses to the queries $q_1, \ldots, q_m$ as D, $d_i = d_i'$ ($D' = \{d_1', \ldots, d_n'\}$).

Our claim that $C1$ gave \$60,000 is equivalent to the statement: Any database with the same responses to the 12 queries of Table I must have $D(1) = \$60{,}000$.

Our definition of a security problem has two important features. First, we allow that a user may know in advance parts of the database. For example, suppose that $C1$'s contribution is known in advance. Then two queries suffice to determine the contribution of $C6$ as

$$C6 = \text{sugar} - \text{northeast} + C1.$$

Second, we allow that not all sequences of queries may be permitted. Suppose that a particular database allows averages of size k and a user knows just one value. Then in just two queries he can compromise the database. He asks:

(1) What is the average of $x, y_1, \ldots, y_{k-1}$?
(2) What is the average of $x', y_1, \ldots, y_{k-1}$?

If he already knows x, he can determine x'. The reason the user was so successful is that he was allowed to ask two queries that *overlapped* greatly; his queries overlapped in $k - 1$ elements. In the next section we will consider the problem of whether one can compromise such a database if no two average queries can overlap very much.

3. APPLICATIONS TO A PARTICULAR MODEL: AVERAGES

In this section, we assume that we are given a database $\{x_1, \ldots, x_n\}$ of numbers and that queries may be made about the sum of any subset of the database

consisting of exactly k elements (this is equivalent to averages of k-element sets). We assume the further restriction that no two queries may overlap in more than r positions. And we assume that the values of $x_1, \ldots, x_l$ are known in advance by the user ($0 \leq l < k - 1$). We then wish to study the behavior of the quantity $S(n, k, r, l)$, the smallest number of queries that suffices to compromise the database. Compromising the database will consist of generating the value of one previously unknown element, e.g., x_{l+1}.

Before proceeding, we present some sample values of the function $S(n, k, r, l)$.

Example. $S(n, 3, 2, 0) \leq 4$, $n \geq 4$. Let the queries be Q_1, Q_2, Q_3, Q_4 where

$$Q_1 = x_1 + x_2 + x_3$$

$$Q_2 = x_1 + x_2 + x_4$$

$$Q_3 = x_1 + x_3 + x_4$$

$$Q_4 = x_2 + x_3 + x_4.$$

Then x_4 can be found as $\frac{1}{3}(-2Q_1 + Q_2 + Q_3 + Q_4)$ and this is optimal.

Example. $S(n, 4, 1, 1) \leq 6$, $n \geq 11$. Let the queries be $Q_1, \ldots, Q_6$ where

$$Q_1 = x_1 + x_3 + x_4 + x_5$$

$$Q_2 = x_1 + x_6 + x_7 + x_8$$

$$Q_3 = x_1 + x_9 + x_{10} + x_{11}$$

$$Q_4 = x_2 + x_3 + x_6 + x_9$$

$$Q_5 = x_2 + x_4 + x_7 + x_{10}$$

$$Q_6 = x_2 + x_5 + x_8 + x_{11}.$$

Then $\frac{1}{3}[(Q_1 + Q_2 + Q_3) - (Q_4 + Q_5 + Q_6)] = x_1 - x_2$, which yields the value of x_2 since x_1 is known.

We begin our study of the properties of the function S by establishing a lower bound on its value.

THEOREM 1. $S(n, k, r, l) \geq 1 + (k - (l + 1))/r$.

PROOF. Suppose that after t queries we can determine the value of x_{l+1}, and let the queries be represented as

$$Q_i = \sum_{j=1}^{k} x_{i_j}, \qquad i = 1, \ldots, t$$

for $1 \leq i_1 < \ldots < i_k \leq n$ where we assume the set $\{i_1, \ldots, i_k\} \cap \{j_1, \ldots, j_k\}$ has at most r members. This can then be represented as being able to satisfy the relation

$$\sum_{i=1}^{t} \alpha_i Q_i = \sum_{j=1}^{l+1} \beta_j x_j \tag{1}$$

in terms of coefficients of the x_i with $\beta_{l+1} \neq 0$. We proceed now by a counting argument, observing that the left-hand side of (1) can be rewritten as

$$\sum_{i=1}^{t} \alpha_i Q_i = \sum_{i=1}^{t} \alpha_i \sum_{j=1}^{k} \delta_{i_j} x_j = \sum_{\sigma=1}^{n} \left(\sum_{i=1}^{t} \alpha_i \delta_{i_\sigma} \right) x_\sigma$$

where δ_{i_σ} is 1 if x_σ belongs to query i and 0 otherwise. Thus t must be such that at most $l+1$ of the terms

$$\sum_{i=1}^{t} \alpha_i \delta_{i_\sigma}, \qquad \sigma = 1, \ldots, n$$

are nonzero. In order for such a term to be zero, it must be the case that $\delta_{i_\sigma} = 0$, $i = 1, \ldots, t$, or that i, j are distinct such that $\delta_{i_\sigma} = \delta_{j_\sigma} = 1$. Thus every x_σ that appears in some query must appear in at least two queries for $\sigma > l+1$. After the first query k x_i have been accessed, and $(k-(l+1))/r$ more queries are required to access all the $x_i (i > l+1)$ at least twice, since at most r of these elements can occur in each new query due to the overlap restriction. Hence $t \geq 1 + (k-(l+1))/r$ is a lower bound for $S(n, k, r, l)$. □

As a dividend of the argument above, we observe that $k - pr$ new variables of the database are added in the $p+1$th query, $1 \leq p \leq (k-(l+1))/r$, and thus the following corollary results, giving a lower limit on the size of a database which can be compromised.

COROLLARY. *If* $n < k^2/2r + k/2 + (l+1)/2 - ((l+1)^2)/2r$, *then* $S(n, k, r, l) = \infty$ *corresponding to a noncompromisible database.*

PROOF. This follows from the argument above since

$$k + \sum_{i=1}^{(k-(l+1))/r} (k - ir) = \frac{k^2}{2r} - \frac{k}{2} + \frac{l+1}{2} - \frac{(l+1)^2}{2r}.$$

These results provide, then, a measure of the limitations of compromising a database. We turn next to the question of actually implementing algorithms to perform these functions in order to get a sense of the tightness of these bounds.

THEOREM 2.

(a) $S(n, k, 1, 0) \leq 2k-1, \qquad n \geq k^2 - k + 1$

(b) $S(n, k, 1, 1) \leq 2k-2, \qquad n \geq (k-1)^2 + 2$

(c) $S(n, kr+\alpha, r, 2\alpha-1) \leq 2k, \qquad n \geq k^2 r + 2\alpha$

(d) $S(n', kr, r, r-1) \leq S(n, k, 1, 0), \qquad n' \geq rk^2.$

PROOF. In each case, our proof consists of an algorithm that performs the given task within the desired bound.

(a) Let the queries be (see Figure 1a)

$$Q_i = \sum_{j=1}^{k} x_{k(i-1)+j}, \qquad i = 1, \ldots, k-1$$

$$Q_{k+i-1} = \sum_{j=1}^{k-1} x_{k(j-1)+i} + x_{k^2-k+1}, \qquad i = 1, \ldots, k.$$

Then

$$\frac{\sum_{i=0}^{k-1} Q_{k+i} - \sum_{i=1}^{k-1} Q_i}{k} = x_{k^2-k+1}.$$

(b) Let the queries be (see Figure 1b)

$$Q_i = x_1 + \sum_{j=2}^{k} x_{(k-1)(i-1)+j}, \qquad i = 1, \ldots, k-1$$

$$Q_{k-1+i} = \sum_{j=1}^{k-1} x_{1+i+(j-1)(k-1)} + x_{(k-1)^2+2}, \qquad i = 1, \ldots, k-1.$$

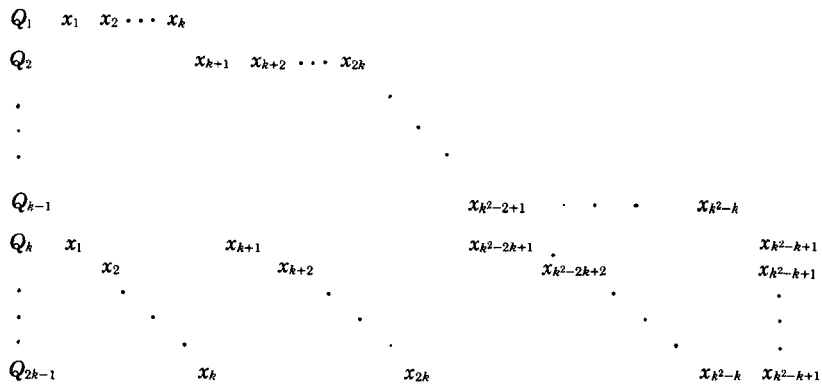

Fig. 1a. The queries for Theorem 2(a)

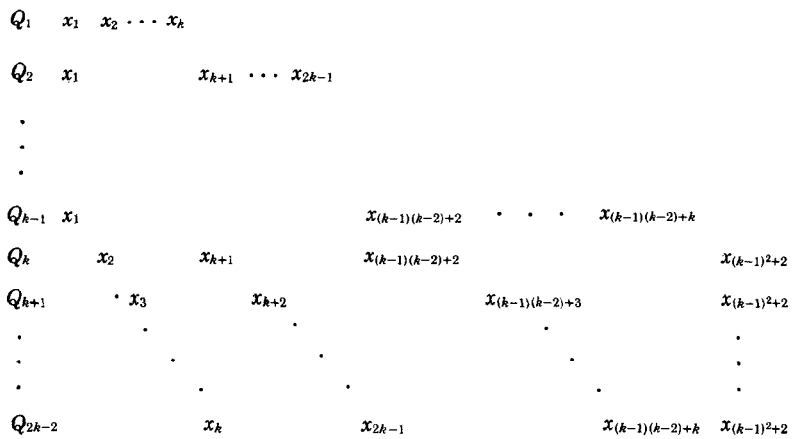

Fig. 1b. The queries for Theorem 2(b)

Then

$$\frac{\sum_{i=1}^{k-1} (Q_i - Q_{k-1+i})}{k-1} = x_1 - x_{(k-1)^2+2}$$

and $x_{(k-1)^2+2}$ may be determined since x_1 is known in advance.

(c) Let the queries be

$$Q_i = \sum_{j=1}^{kr} x_{kr(i-1)+j} + \sum_{l=1}^{\alpha} x_{k^2r+l}, \qquad i = 1, \ldots, k$$

$$Q_{k+i} = \sum_{j=1}^{k} \sum_{l=1}^{r} x_{kr(j-1)+(i-1)r+l} + \sum_{m=1}^{\alpha} x_{k^2r+\alpha+m} \qquad i = 1, \ldots, k.$$

Then

$$\sum_{i=1}^{k} (Q_i - Q_{k+i}) = k\left[\sum_{l=1}^{\alpha} x_{k^2r+l} - \sum_{m=1}^{\alpha} x_{k^2r+\alpha+m}\right]$$

and x_{k^2r+1} can be determined if the values of $x_{k^2r+2}, \ldots, x_{k^2r+\alpha}$, $x_{k^2r+\alpha+1}, \ldots, x_{k^2r+2\alpha}$ are known in advance.

(d) This statement follows by letting $y_i = x_{(i-1)r+1} + \ldots + x_{ir}$, $1 \le i \le k^2 - k + 1$ using an algorithm for $S(n, k, 1, 0)$ to find y_1, and using known values of $x_1, \ldots, x_{r-1}$ and y_1 to find x_1.

The reason we are interested in the results of this section is that we feel they are hard evidence of how complex it will be to secure a database. The complex and obscure techniques used in Theorem 2 to crack the database demonstrate how difficult it will be to determine whether a series of simple queries can compromise a database. On the other hand, Theorem 1 points out that there do exist mechanisms (bounding the overlap and the number of queries) that can protect a database.

4. APPLICATIONS TO A PARTICULAR MODEL: MEDIANS

In this section we assume that we are given a database $\{x_1, \ldots, x_n\}$ of distinct numbers[1] and that queries can be made about the median of k elements for some fixed odd k. A median query $q(y_1, \ldots, y_k)$ returns the median's value but *not* which y_i is equal to this value. We also assume that no values are known in advance. We then wish to study the quantity $M(n, k)$, the smallest number of queries that suffices to determine some element of the database perfectly. Our main result is the following theorem.

THEOREM 3. $M(n, k) \le 3/2(k + 1) + 2$ *provided* $n \ge k + 2$.

PROOF. Let $p = (k + 1)/2$. Also let $x_1, \ldots, x_{k+2}$ be $k + 2$ objects of the database. First perform all possible k medians involving the objects $x_1, \ldots, x_{k+1}$. Clearly there are $\binom{k+1}{k}$ such medians. These medians result in exactly two answers, say h and l with $h < l$. Let

$$H = \{x_i \mid x_i \le h\}$$

and

$$L = \{x_i \mid x_i \ge l\}.$$

Then $x_i \in H$ iff the median of $\{x_1, \ldots, x_{k+1}\} - \{x_i\}$ is l; $x_i \in L$ iff the median is h. An easy argument shows that $|H| = |L| = p$.

We now form the median of $H' \cup L \cup \{x_{k+2}\}$ where

$$H' = H - \text{any two elements of } H.$$

There are three cases:

(1) The answer $= l$. Then $x_{k+2} < l$.
(2) The answer $> l$. Then $x_{k+2} > l$.
(3) The answer $< l$. This is impossible.

[1] As observed in [2], this assumption is not very restrictive.

Thus this query determines whether $x_{k+2} >$ or $< l$. Without loss of generality assume that $x_{k+2} < l$.

We now fix L_0 as a set of $p - 1$ elements of L. Let us finally examine the medians of the sets

$$H \cup \{x_{k+2}\} \cup L_0 - \{x_i\} \quad \text{for each} \quad x_i \in H \cup \{x_{k+2}\}.$$

We now claim that one median (say m) occurs p times and one median occurs once. This follows by a simple argument. Then $m = x_i$ where x_i is not in the set when m does not occur. □

This result demonstrates clearly that even the simple operation of median can be used to compromise a database; indeed, this can be done in very few queries.

5. CONCLUSIONS

A precise model of the security problem for databases has been presented. In this model we were able to demonstrate how to control the queries a user could make in order to stop him from compromising the database. While we did this only for queries about averages and medians, we can extend the model to handle queries of other types. This model gives rise to a number of interesting combinatorial problems which have applications to problems of applicability to designers or databases. While we have presented an introduction to problems in this area here, a number of related problems remain open. For example, suppose we change our constraints on overlapping queries to allow queries to overlap only in certain coordinates. Or, suppose we allow queries of varying lengths. Or, suppose we may ask for medians but wish to determine a specific database entry. Furthermore, in each case studied here, we have considered worst case behavior, the number of queries necessary to guarantee that the database is compromised. We could do a similar analysis for best case behavior, asking for the fewest queries after which the database may be compromised. Many of these issues will be studied in a forthcoming paper [2].

ACKNOWLEDGMENT

We wish to thank Rich DeMillo, Mary-Claire van Leunen, and Steven Reiss for a number of helpful comments on an earlier draft.

REFERENCES

1. CODD, E.F. A relational model of data for large and shared data banks. *Comm ACM 13*, 6 (June 1970), 377–387.
2. DEMILLO, R., DOBKIN, D., AND LIPTON, R. Even data bases that lie can be compromised. *IEEE Trans. Software Eng. SE-4*, 1 (1978), 73–75.
3. DENNING, D.E. A lattice model of secure information flow. *Comm. ACM 19*, 5 (May 1976), 236–243.
4. FENTON, J.S. Memoryless subsystems. *Comptr. J. 17*, 2 (1974), 143–147.
5. GRAHAM, G.S., AND DENNING, P.J. Protection—principles and practice. Proc. AFIPS 1972 SJCC, Vol. 40, AFIPS Press, Montvale, N.J., pp. 417–429.
6. HAQ, M. Insuring individuals' privacy from statistical data base users. Proc. AFIPS 1975 NCC, Vol. 43, AFIPS Press, Montvale, N.J., pp. 941–946.
7. HOFFMAN, L.J., AND MILLER, W.F. Getting a personal dossier from a statistical data bank. *Datamation 16*, 5 (May 1970), 74–75.

8. JONES, A.K., AND LIPTON, R.J. The enforcement of security policies for computation. Proc. 5th Symp. Oper. Syst. Principles. *Oper. Syst. Rev. (ACM) 9*, 5 (1975), 197-206.
9. JONES, A.K., AND WULF, W.A. Towards the design of secure systems. *Software—Practices and Experience 5*, 4 (Oct. 1975), 321-336.
10. LAMPSON, B.W. Protection. Proc. 5th Princeton Symp. Inform. Sci. and Syst., 1971, pp. 437-443.
11. ORGANICK, E.I. *The MULTICS System: An Examination of Its Structure.* M.I.T. Press, Cambridge, Mass., 1972.
12. WULF, W.A., ET AL. HYDRA: The kernel of a multiprocessor system. *Comm. ACM 17*, 6 (June 1974), 337-345.

Received March 1976; revised June 1978