This paper surveys some aspects of data base security, with emphasis on basic principles and ways to express security requirements. Security policies and theoretical models are considered in detail, and the models are used to compare the security features of some data base management systems.

Data base security: requirements, policies, and models

by C. Wood, E. B. Fernandez, and R. C. Summers

Information is a critical resource in today's enterprises, whether industrial, commercial, or civic. Enterprises are automating not only basic operational functions such as invoicing, payroll, and stock control, but also management support functions such as sales forecasting, budgeting, and financial control. To support such functions, more and more computerized information is maintained in a data base, and that information is increasingly relied upon.

Although the data base approach brings many advantages, it also creates new, or more intense, problems in terms of security. In the traditional approach, each application system has its own files. A limited amount of data is shared by passing tailored files from one system to another. In a data base environment, many users with differing security requirements, some on line, have access to a common pool of data of varying degrees of sensitivity. Access to the data must be controlled, therefore, to ensure that privacy requirements are satisfied, leakage of enterprise secrets is prevented, and the possibility of accidental or malicious corruption of data is minimized. However, the centralized nature of data bases also provides a basis for system-wide solutions.

This paper is concerned with data base security, in particular with the control and monitoring of access to information stored in a data base. Its objectives are to provide a general understanding of the principles involved, to enable users to formally express their

security requirements, and to provide a framework for the evaluation of the security features of different data base management systems. To meet those objectives, the paper emphasizes theoretical models of data base security.

The use of models allows us to concentrate on significant aspects of the problem without simultaneously considering complex details of implementation. The use of theoretical or ideal models, rather than the implicit models adopted by particular systems, provides a general yardstick for system comparison. To apply the principles discussed here, it is necessary also to understand security mechanisms that can be used in designing, implementing, and controlling data base systems. The paper deals only briefly with those mechanisms; they can be studied in References 1 and 2.

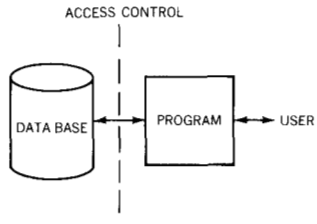
In the following section we define security terms used in the remainder of the article. Next, to provide perspective, we review all types of threats to data base security, as well as mechanisms and procedures that protect against them. We describe some security policies an enterprise may wish to adopt. Finally, we develop and compare models for controlling access to information and its flow within the computer. These models can be used to specify many of the security policies.

definitions

The terminology of data base security varies a great deal, perhaps because several different areas are involved, each with its own terminology. We have tried to choose the most widely used definitions.

Information security is the protection of information against unauthorized disclosure, alteration, or destruction. It follows that *data base security* is the protection of information maintained in a data base. The need for data base security derives in part from considerations of privacy. The term *privacy* is used broadly for all the ethical and legal aspects of personal data systems—systems that contain information about individuals. More specifically, privacy is the right of individuals to some control over information about themselves.

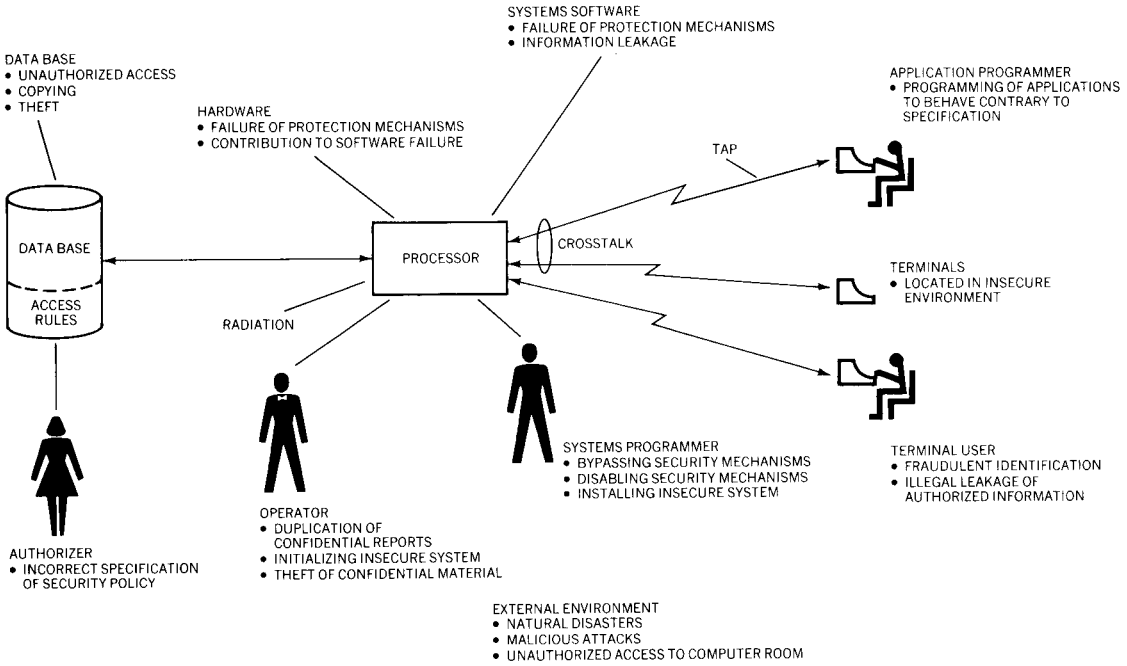
Figure 1 Data base access control



Since security has been defined in terms of protection against *unauthorized* actions, we must define what we mean by *authorization*. *Authorization* is the specification of rules about who has what type of access to what information. An authorized action abides by these *access rules*. The person who writes access rules is an *authorizer*.

The process of ensuring that information is accessed only in authorized ways is called *access control*. Access control is one of several possible objectives of security within a computing system; it is illustrated in Figure 1, which shows information from a

Figure 2 Security threats



data base being transferred to and from a user's program according to a set of access rules. *Information flow control* aims at preventing security leakage as information flows through the computer system.

Security threats and defenses

Figure 2 illustrates some of the many possible threats to the security of a computer system.[3] These threats can be broadly classified as either malicious or accidental. Malicious threats are attempts to circumvent the security mechanisms designed to protect a data base. Malicious attackers may exploit loopholes in the system, or they may abuse positions of privilege and trust by using the data to which they have access in an illegitimate way—for example by selling information to competitive enterprises. Accidental threats include hardware and software failures that cause an installation's security policies to be enforced incorrectly. Such threats may also result from human errors, such as accidentally making information available to the wrong people. Also accidental are natural disasters such as flood and fire, which may destroy information or prevent access to it.

security mechanisms and procedures

Because security threats arise from such a wide variety of sources, the mechanisms and procedures necessary to provide a

secure environment must cover many areas of an enterprise. Some of the more important are listed in Table 1.

The security of a data base depends on a complex set of protective measures—human, software, and hardware. One weak link in the chain of security measures can compromise the security of the whole system. In this paper we concentrate on that part of security directly associated with the control and monitoring of access to information in the data base. Figure 3 shows some of the safeguards that can be built into a data base system. *Authentication* (1) verifies the identity of a user when he logs on. The most common authentication method uses *passwords,* which are combinations of characters known only to the user. Another common method requires the user to insert a machine-readable badge. More elaborate techniques depend on sensing fingerprints, the size of the user's hand, or some other physical characteristic of the user.

Once the user is authenticated and attempts a transaction (2), his authorization to use that transaction is checked (3). Execution of the transaction may involve several application programs, which are stored in a program library (4). *Application programmers* build and maintain the library, while an *application administrator* controls the development and use of the programs. In the multilevel data base architecture described in the ANSI/SPARC proposals,[4] the data base is described at different levels. High-level descriptions, which define the conceptual schema, are designed by an *enterprise administrator,* while lower-level descriptions such as the internal schema are built by a *data base administrator.* Authorization rules control access to the objects in the data base and to specific portions of the program library. These rules are written by a *security administrator* or *authorizer,* using some appropriate language. A *security monitor* or *security officer* checks the day-to-day operational application of the rules. When application programs are executed, their requests for data go to the data base management system (DBMS) (5). This software has access to the data base descriptions needed to organize data access, and it checks authorization and semantic integrity (6). The DBMS keeps a log of accesses to the data base (7). A *security auditor* checks the log at prescribed intervals for compliance with security policies.

Data base access requests, once validated, are translated into I/O calls, which are passed to the operating system (8). Additional checks are possible here for proper use of files or of operating system functions. The hardware can provide additional protection (9), such as enforcing the correct use of information types (for example, no execution of data). The physical volumes on which the data base is stored can be protected by encryption and by backup copies (10).

Figure 3 Security checking for transaction processing

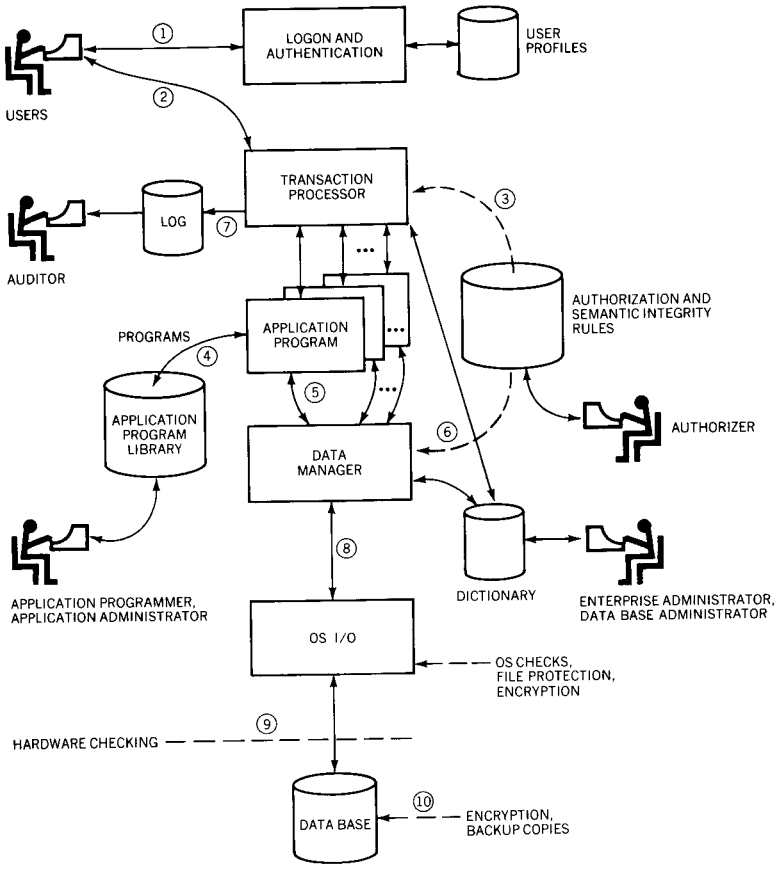


Table 1 Security mechanisms and procedures

Area	*Mechanisms and procedures*
External procedures	Security clearance of personnel Protection of user ID and passwords Information classification and security policy formulation DP audit Application program controls
Physical environment	Provision of secure areas for files, processors, terminals Radiation shielding
Data storage	Encryption of stored data Duplicate copies of data base
Processor—software	Authentication of user Access control Threat monitoring Journaling of data base transactions
Processor—hardware	Protection Reliability
Communication lines	Encryption of transmitted data

Policies for data base security

For a systematic approach to data base security, it is important to distinguish between security policies and security mechanisms. *Security policies* are high-level guidelines concerning information security. Selected from among alternatives, they are dictated by user needs, installation environment, institution regulations, and legal constraints. *Security mechanisms* are sets of functions used to implement and enforce the various security policies. Depending on their nature, the functions involved in a given mechanism can be implemented in hardware, software, or firmware, or by administrative procedures.

Security decisions are made at each level of an information system, from corporate actions to hardware implementation. These decisions constitute the set of security policies of an enterprise. The policies depend on the activities of the enterprise. Military, commercial, and educational enterprises usually have quite different security policies. Legal considerations also play a role. For example, privacy legislation dictates certain policies for government agencies and credit bureaus. Policies of a computing installation are directed by the higher-level policies of the enterprise and also depend on the available hardware and software. In general, policies at the lower levels are affected by those at higher levels. For example, a policy that requires decentralization at a high level requires low-level mechanisms that support decentralized operation. Conversely, the lower levels can affect higher levels. For example, a given high-level policy may not be practical if low-level mechanisms cannot support it adequately.

Considering system structure, policies can be viewed as being implemented by low-level mechanisms. If the ability to change policies is required, it is necessary to separate mechanisms from policies. In other words, the policy should not be built into the mechanism. This separation allows the same mechanism to implement different policies. The advantages of this strategy have been demonstrated in the design of operating systems.[5] It is inevitable that the operating environments of an enterprise and of a data base system will change; therefore, changes in the chosen security policies will be necessary. These changes should not require that the security mechanisms be redesigned or replaced; they should be programmable using the existing mechanisms.

Some of the more important high-level security policies are reviewed below, with the low-level policies they may imply. Most of the low-level policies concern the selection and use of DBMS security features. Many of the low-level policies can be expressed in terms of access rules. Of course the designers of a DBMS make policy decisions when determining which security features to include in the system. In practice, the features currently provided in DBMSs are limited, thus restricting the choice of policies.

centralized or decentralized control

A fundamental security choice is between *centralized* and *decentralized* control. With centralized control, a single authorizer (or group) controls all security aspects of the system. An example of a system with centralized control is INGRES.[6] In other environments, decentralized control may be required for efficiency or convenience.

ownership and administration

A related policy concerns the concepts of *ownership* and *administration* of data bases in an enterprise. The owner of a data base sometimes is considered to be the person responsible for creating the data. For example, the payroll manager might be considered the owner of a payroll data base updated exclusively by the payroll department. With many shared data bases, however, it is difficult to identify a unique owner. A stock control data base, for example, might be updated by the production, sales, purchasing, and shipping departments. While there may or may not be the concept of ownership, there is always the need for an administrator of a data base. This function may be performed by the owner, if one exists, or by a security administrator.

the need-to-know policy

An important example of a high-level policy is restricting information to those people who really need the information in performing their assigned functions. This policy is mandatory for data bases subject to privacy legislation, and it is a sound principle on which to base any security system, as it restricts the number of possible sources of information leaks and it minimizes the possibility that the integrity of the data base will be compromised. This *need-to-know policy* is sometimes called the *policy of least privilege* because all users and programs operate with the least privilege necessary to perform their functions.

maximized sharing

An alternative policy is *maximized sharing*, the intention of which is to make maximal use of the information in the data base. This policy does not necessarily mean that all users are given access to all the information, because there may still be privacy requirements and sensitive data. However, sharing is maximized within these security constraints. A medical research data base, for example, might contain information on certain diseases. The main objective would be to allow researchers maximal access to the information, but any data that could be related to a specific patient would be protected.

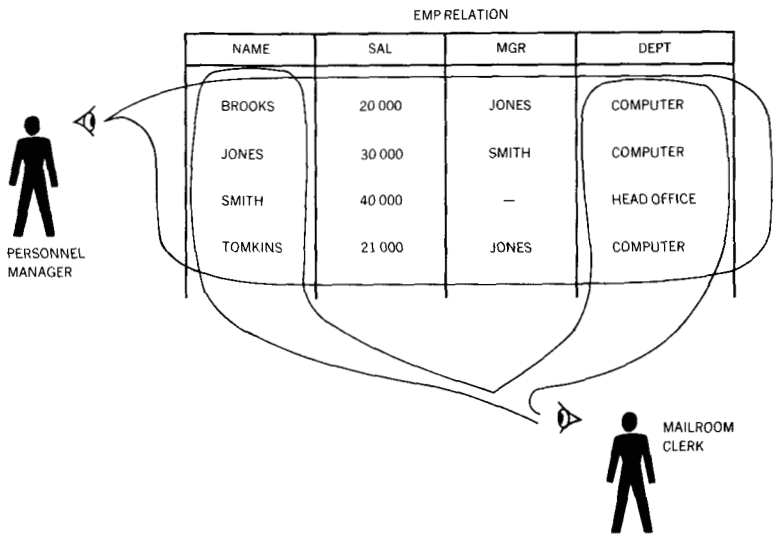
open and closed systems

The choice between an *open* and a *closed* system can have important security implications. In a closed system, access is allowed only if explicitly authorized. In an open system, access is allowed unless explicitly forbidden. A closed system is inherently more secure, but it may have more overhead if sharing is to be maximized.

access rules

The policy of least privilege has various implications for the types of access rules required, depending on the strictness with which

Figure 4 Name-dependent access control

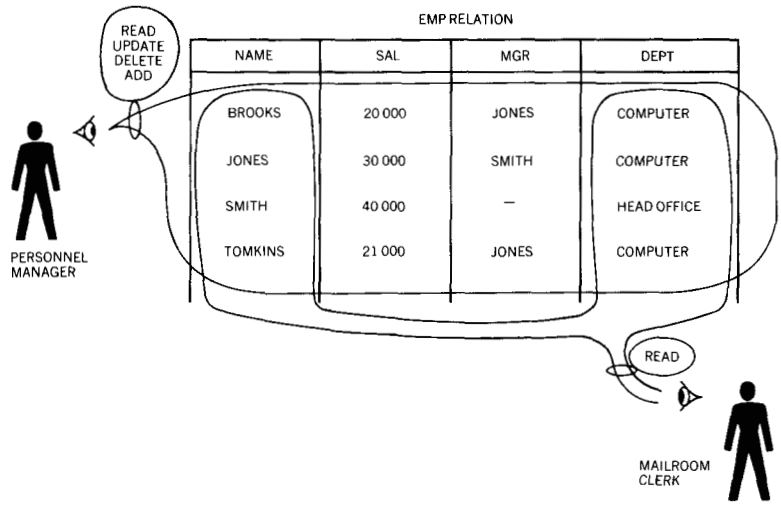


the policy is interpreted. As a minimum, it should be possible to specify the data objects accessible to a given user. (A data object is a group of occurrences of data items and relationships with a name recognized by the DBMS. In a relational DBMS, for example, a data object might be a relation or an attribute, and in a CODASYL DBMS, it might be a set or a record type.) Another policy decision pertains to the *granularity* of control; the smaller the object, the finer the granularity. A strict interpretation of the policy of least privilege requires use of the finest granularity allowed by the DBMS. With relational systems, then, the object is a column or attribute; with IMS it is a data item or field.[7] This type of control is *name-dependent access control,* sometimes referred to as *content-independent access control* because a decision on whether to allow a data access request can be made without accessing data values from the data base. Consider the employee relation EMP with attributes NAME, SAL, MGR, and DEPT, represented in Figure 4. A personnel manager might need access to the complete relation, but a mailroom clerk would require access only to the NAME and DEPT attributes.

content-dependent access control

The policy of least privilege can be extended even further by specifying access rules that *are* dependent on the content of data item occurrences (as opposed to just their names). This type of control is known as *content-dependent access control.* When applied in conjunction with the previous policy, it provides finer granularity of control. For example, managers may have access to data on the salaries of the employees they manage, but not on the salaries of other employees. With this type of control, data values must be retrieved from the data base to determine whether the access request should be allowed.

Figure 5 Access-type-dependent access control



The security policies discussed so far allow users either no access or unlimited access to specified data objects. More control over the use of data is achieved by including in the access rule the type of access to the data object the user is allowed, such as READ, UPDATE, INSERT, DELETE, or some combination. Thus while the personnel manager may have access to EMP for all types of operations, the mailroom clerk might only be allowed to READ the NAME and DEPT attributes (see Figure 5). These users therefore have the minimum set of access rights necessary to perform their jobs. The use of this policy can be simplified by *ordering* the access types so that access of a higher type implies lower-order accesses. In IMS, for instance, REPLACE and DELETE both imply READ.

functional access control

When users need only summary or statistical data, the policy of least privilege requires that they not have access to the underlying detailed data. (We assume here that mathematical functions such as average, sum, and standard deviation are supported by the DBMS at the user interface.) To specify this *functional access control*, the concept of access types can be extended to include functions. Thus we can specify that a user may have access to average salary data but not to individual salary values. While this restriction does not guarantee the security of the individual values,[8] it is an important support for more elaborate methods (which can provide more protection, although still not complete security).

context-dependent access control

Another policy is *context-dependent access control*, which restricts the fields that can be accessed together. For example, if a relation contains employee names and salaries, it may be desir-

able to prevent some users from finding out the salaries of particular employees. One approach would be to prevent access to the relation by those users. To maximize sharing, however, the system would allow separate access to names and salaries while preventing users from accessing them together in the same request or in a specific set of requests (for example, all the requests of a program). The converse of this policy is the requirement that certain fields appear together. For example, information about a person being arrested could be given only if the disposition of the arrest were also included.

history-dependent access control

In general it is not sufficient to control only the context of the immediate request if users are to be prevented from making certain semantic deductions. For example, if the employee relation also contained a project identifier attribute, a user could list names and projects, then salaries and projects, and probably make some correlation between names and salaries. Preventing this kind of semantic deduction requires *history-dependent control*,[9] which considers the immediate request in the context of past requests.

History-dependent control is really only a partial solution to a more general problem. Ideally it would be desirable to express rules in terms of controlling access to facts that are represented in the data base either directly or indirectly. The DBMS would then determine which specific request to allow in light of these more general rules. However, such a determination, which would require the DBMS to make semantic inferences, is beyond the limits of current technology.

The policies described above control access to the data base, but not the use made of the data once it is accessed. Control over the use of accessed data within a program is necessary, for example, to prevent the leakage of information from an authorized program to an unauthorized one.

nondiscretionary access control

We have implicitly assumed that some authorizer can provide other users of the system with access to data. This type of control is *discretionary access control*. A simpler but less flexible approach is *nondiscretionary access control*, by which the use of the system is compartmentalized so that data in one category or compartment cannot be accessed by users in another category. It is possible, of course, to mix discretionary and nondiscretionary access control. For instance, a department manager may allow discretionary access rules to be specified for personnel within the department, but with the nondiscretionary rule that the department's data cannot be accessed by people outside the department. This approach has been taken in the design of some military systems.[10]

A basic model of data base access control

protection in operating systems

Models of data base access control have grown out of work on the theory of protection in operating systems. One of the most influential protection models was developed by Lampson[11] and extended by Graham and Denning.[12] The basis of their model is the access rule, which specifies the types of access a *subject* can have for an *object*. In the context of operating systems, objects are entities, known to the operating system, to which access must be controlled, such as main memory pages, programs, auxiliary memory devices, and files. Subjects are the entities that request access to objects, usually a process-domain pair, a *process* being a program in execution, and a *domain* the environment in which the process is executing. Examples of domains in an IBM System/370 are the supervisor and problem program states. Access types might be OWN, EXECUTE, ALLOCATE, and READ. The set of all access rules can conveniently be thought of as forming an access matrix A, in which columns $o_1, o_2, \cdots, o_n$ represent objects, and rows $s_1, s_2, \cdots, s_m$ represent subjects. The entry $A[s_i, o_j]$ contains a list of access types, $t_1, t_2, \cdots$, which specifies the access privileges held by subject s_i for object o_j. The objects accessible by a subject, together with the mode of access, are sometimes termed the *capabilities* of the subject.

This model assumes that all attempted accesses to an object are intercepted and checked by a controlling process sometimes known as a monitor. Thus when subject s_i initiates access t_k to object o_j, the monitor checks to determine whether $t_k \in A[s_i, o_j]$. As the flow of control during program execution proceeds from one subject to another, the access rules need to be modified dynamically so that existing access rights of a subject can be copied or granted to a new subject. Access rights in the matrix are flagged if copying them is to be allowed. A subject that has not been debugged can thus be prevented from indiscriminately giving away access rights that it has been granted. The importance of this approach is that the effects of errors are confined. Errors can no longer propagate in an uncontrolled way throughout the whole system. Thus reliability is enhanced (as other parts of the system can often continue executing correctly) and debugging is simplified.

operating system and data base system security

This model treats the security of all system objects in a uniform way. Therefore one approach to data base security is to consider it as just a subset of operating system security. Thus the objects in the access matrix would be not only resources such as memory pages, devices, and files, but also data base objects. The operating system could then be extended to handle all security within the system. There are some fundamental differences between operating system security and data base system security, however, as listed below:

- There are more objects to be protected in a data base.
- The lifetime during which data is used normally is longer in a data base.
- Data base security is concerned with differing levels of granularity, such as file, record type, field type, and field occurrence.
- Operating systems are concerned with the protection of real resources. In data base systems the objects can be complex logical structures, a number of which can map to the same physical data objects.
- There are different security requirements for the different architectural levels—internal, conceptual, and external.
- Data base security is concerned with the semantics of data, not just its physical characteristics.

An operating system extended to handle these differences would be highly complex. It therefore seems a good design principle to treat data base security as a responsibility of the DBMS rather than the operating system. DBMS security mechanisms use the basic security services provided by the operating system, and operating systems may indeed provide services primarily intended for DBMS use. As a further justification, most DBMSs in practice are designed to run on existing operating systems. It is then appropriate to develop special models for data base security. One such model, based on work done by Fernandez, Summers, and Coleman,[13] is described below.

access-matrix-based models

For controlling data base access we have similar concepts of access rules and access matrix, but objects are now sets of data item occurrences. The names of these sets must be recognized by the DBMS. We use the variable O (capital letters indicate set variables) to represent these data objects. For a given data base, O may take on any of a finite set of values $\{O_1, \cdots, O_i, \cdots, O_n\}$. For example, in a relational DBMS the possible values of O would be the names of all the relations and attributes defined to the system. Subjects are now end users—the people who request data base access. In a given installation there is a set of potential users $\{s_1, \cdots, s_i, \cdots, s_m\}$. The variable s is defined over this set. Access types are operations such as the familiar READ, WRITE, UPDATE, APPEND, and DELETE. For a given DBMS, a set of legal access types $\{t_1, \cdots, t_i, \cdots, t_k\}$ is defined. The variable t may take on any of these values.

It is important to note that the data base access matrix is more static than the operating system protection matrix. It is modified explicitly only by an authorizer who wishes to specify a new access rule or revoke an old one. Figure 6 shows part of an access matrix that represents the rules governing access to the EMPLOYEE relation. The attributes of the EMPLOYEE relation are EMP_NAME, PERS_NO, ADDRESS, TEL_NO, and SALARY. From

Figure 6 Access matrix

SUBJECT \ OBJECT	EMP_NAME	PERS_NO	ADDRESS	TEL_NO	SALARY
PERSONNEL_MANAGER	ALL	ALL	ALL	ALL	ALL
ADMIN_CLERK	READ	READ	READ	READ	—

the figure we see that the personnel manager has unrestricted access (indicated by the ALL entry) to all attributes, while the administration clerk has READ access to all attributes except SALARY. A null entry implies that no access is allowed to that object.

It is worth emphasizing that a model serves to aid the understanding of the logical functioning of a system and does not imply any particular implementation. Thus access rules do not have to be stored in matrix form. In fact that would be an inefficient way of storing them because, in general, the access matrix is sparse—that is, any given subject has access to only a small subset of all possible objects in the data base.

An access control model should be general enough to represent the security policies described in the previous section. The access matrix is capable of modeling name-dependent access control down to any level of granularity supported by the DBMS. To represent access rules that are content-dependent, the model must be extended so that the access rule contains a predicate, p. The predicate can be considered to allow an arbitrary set, O', of data item occurrences to be defined as the effective object for the access rule. That is:

$$O' = O{:}p$$

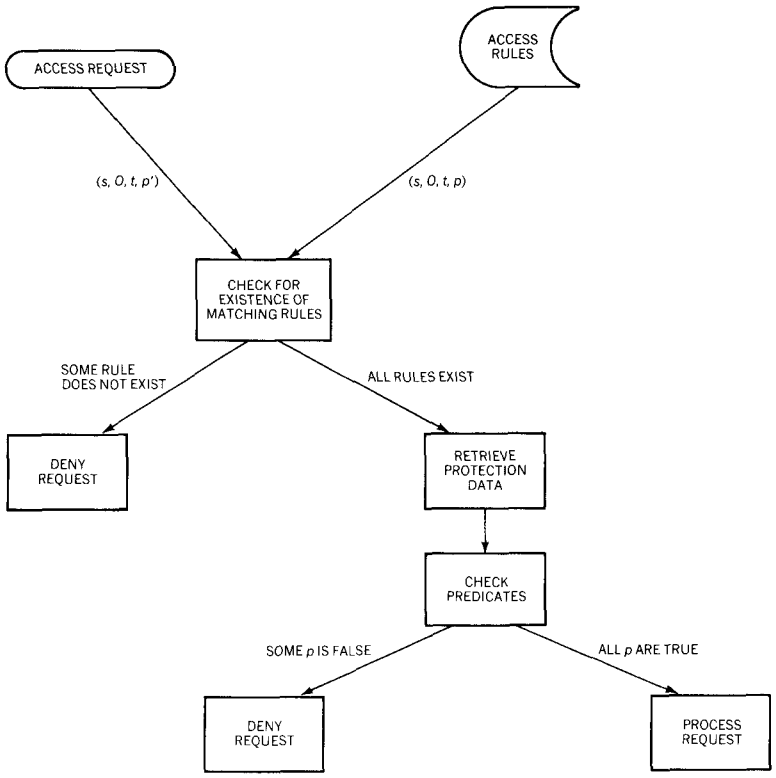
The predicate may also place additional constraints on the access rule (such as allowing access only at a certain time of day) by referring to system variables. The predicate in this case can be considered to be composed of a data predicate, p_d, and a system predicate, p_s, connected by a boolean operator. The data predicate p_d should then be substituted for p in the above expression for O'.

Figure 7 An access rule

s	O	t	p
PAYROLL_CLERK	EMPLOYEE	READ	SALARY < 20 000

An access rule can now be represented by the tuple (s, O, t, p), which specifies that subject s has access t to those occurrences of O for which predicate p is true. The data that must be retrieved to evaluate the predicate is known as *protection data*. Figure 7 shows a simple example of an access rule that gives the payroll

Figure 8 Model of access validation



clerk READ access to the EMPLOYEE relation for those employees who earn less than $20 000. By using suitable predicates, certain types of context-dependent access control can also be specified. For example, a predicate could enumerate fields that should appear together in a query.

Access control is not achieved just by specifying access rules. There must also be a *validation process* which ensures that all accesses to the data base are authorized by access rules. A possible model of the validation process is indicated in Figure 8. All data base access requests are intercepted and passed to the validation process in the form (s, O, t, p'), indicating that user s has requested access t to the set of data item occurrences defined by $O{:}p'$. It is assumed here that the identity s of the requesting user has been authenticated. If a rule with the same (s, O, t) exists, protection data for evaluating the access rule predicate is retrieved; otherwise the request is denied. If the predicate in the access request refers to data items not included in the requested object, it is necessary to make sure that the subject also has READ access to these data items. For example, a query may request a list of the names of employees who earn over $100 000. The re-

quester must not only have access to employee names, he must also have READ access to salary information. If any of the relevant access rules do not exist, the request is denied. If they do exist, the predicates in the rules must be evaluated. If any are false, the request is denied; otherwise the request is allowed to proceed.

We have assumed for simplicity that the request is either completely satisfied or denied, as would always be the case if the request was for a specific field. When the request is for a record, then, if some of the fields in that record are authorized and some are not, the enforcement process could allow the authorized fields to be passed to the user, rather than denying the whole request. This is a policy decision that must be made by the designer of the security procedures. Likewise, a request for a set of records may be modified so that only the subset that satisfies the predicates is returned to the user. The technique of partially satisfying a user request is known as *query modification*.[6]

An alternative to query modification is to give users access to objects that are defined specifically to provide the users with data base access tailored to their needs. Such derived objects, called *views* or *external schemas*, have been used, for example, in System R.[14] The controlling of access to views can also provide context-dependent, content-dependent, and functional access control.[15]

The time when the various steps occur in the validation process depends on the implementation; it may range from compilation time to execution time. Although it is most secure to perform all the validation steps when the access request is executed, for greater efficiency some systems, such as System R, perform the steps as early as possible.

specifying security policies

We can use the access rule concept as defined in the basic model to formally express some of the policies described in the previous section. For example, the rules that authorize the mail clerk to read the NAME and DEPT attributes of the EMP relation of Figure 5 are:

('MAIL_CLERK',EMP.NAME,READ,-)
('MAIL_CLERK',EMP.DEPT,READ,-)

where the predicate is null (that is, it is always true). The rule that authorizes the personnel manager to read the salaries of the employees he or she manages is:

('PERS_MGR',EMP.SAL,READ,WHERE EMP.MGR='PERS_MGR').

extensions to the basic model

By defining the access rule as (s, O, t, p) we have left unexpressed some important requirements of authorization and request validation. We extend the model by introducing three new components

of the access rule, rules for validating authorizations and requests, and additional interpretations of subject, object, and predicate. The extensions are based on the model of Hartson and Hsiao[9] and on the design for a secure data base system developed at the IBM Los Angeles Scientific Center.[16]

One requirement is for control over the set of access rules of a system. The model as specified so far does not allow for some important policies about who may write access rules. One such policy permits only the authorizer who wrote the rule to change it. For this purpose, the access rule specifies the authorizer, a, so that the rule becomes (a, s, O, t, p). The model must also cover important policies for delegation of rights. By a *right* we mean a certain kind of access to an object; a right is the (O, t, p) of the access rule. A subject s_1 who holds the right may be allowed to delegate the right to another subject s_2; such delegation is equivalent to inserting a new access rule $(s_1, s_2, O_1, t_1, p_1)$. Since a portion of the rule is copied, we use the term *copy flag*, f, for an additional component of the access rule. The extended rule, then, becomes (a, s, O, t, f, p), where f specifies whether s is allowed to delegate the access right. To express policy choices (such as how to control delegation) in the model, we speak of *validation rules*. Some validation rules govern changes to access rules; others govern the way requests are validated.

We extend the access rule further by specifying actions to be taken when the rule is used during request validation. These actions can be taken either before or after the access decision is made, and invocation after the decision can be contingent on what decision was made. One use of this contingency is for actions to be taken when the request is denied, such as notifying a security monitor or logging the illegal request. (Usually such enforcement actions reflect a system-wide policy that applies to all subjects and objects, or a policy that applies to an object regardless of subject. For complete flexibility, however, we include action specifications in the access rule.) We introduce a list of pairs, $(c_1, ap_1), \cdots, (c_n, ap_n)$, which specify auxiliary procedures to be invoked and their conditions of invocation. The *extended access rule* now becomes $(a, s, O, t, f, p, [(c_1, ap_1), \cdots, (c_n, ap_n)])$. Table 2 summarizes the elements of the model.

interpretations of the access rule

Programs or applications can appear in access rules as subjects. It is useful for a program to appear as a subject when it is desirable that the program's rights amplify the user's rights, allowing, for example, for sorting of a file that the user cannot read. In the Hartson and Hsiao model,[9] each rule can have extensions which specify the rights of programs. In some other cases restrictions are needed, as when users' rights are limited by the rights of the applications they are using.[13] Programs and applications can also appear as objects. The relevant access types are then EXECUTE

Table 2 Elements of the security model

Element	*Interpretation*
Basic access rule (s, O, t, p)	Controls access to protected objects
Extended access rule $(a, s, O, t, f, p, [(c_1, ap_1), \cdots, (c_n, ap_n)])$	
Authorizer, a	Person who writes access rules
Subject, s	User, application, transaction, terminal, etc.
Object, O	Data, program, application, etc.
Access type, t	READ, UPDATE, APPEND, AUTHORIZE, etc.
Copy flag, f	Control for delegation of rights
Predicate, p	Condition for access
Auxiliary procedure ap	Rule-specific extension of validation process
Auxiliary procedure c	Condition for auxiliary procedure invocation
Request (s, O, t, p)	Specification of access event
Validation process	Checking requests against rules
Validation rules	Control of validation process and of access rules

and USE. Thus the following rule can be used to specify that user s is allowed to use the application ENROLLMENT:

(s, USE, ENROLLMENT).

Validation rules govern the interpretation of access rules. For example, one use of the predicate is to specify the data occurrences to be accessed—that is, to provide content-dependent access. Another use is to specify the system states under which access is allowed. In the first use only, the predicate can be made true by query modification. For example, a payroll clerk's request to read all salaries of employees in a given department can be modified to request only salaries under $20 000. This policy of modification can be expressed by a validation rule. There is not always a single obvious way to make an access decision. As another example, if the subject in a rule is allowed to be a user *group*, a policy is needed for handling overlapping groups. If two groups had different access predicates for (O, t), for example, the predicate for a user who belonged to both groups could be found by a validation rule that specified the OR of the two predicates.

Multilevel models

The models described provide for an arbitrary assignment of access rights to subjects. Multilevel models differ in several re-

Table 3 Elements of the multilevel model

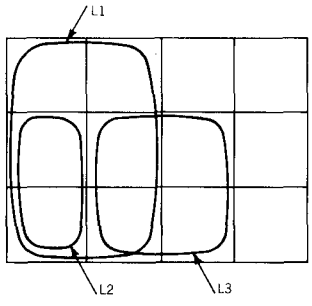
Element	*Interpretation*
Subject, s	Process
Object, o	Data, files, etc.
Classifications	Clearance level of subject, classification level of object
Categories	Access privileges
Security level	⟨Classification, category set⟩
Access type	No observe, no alter; observe only; observe and alter; alter only
Access matrix	Discretionary security
Request	Changes current access or other aspects of system state
(s, o, t)	Current access
Decision	Yes, no, error, or ?
Rules	Determine decision, next state

spects. First, they deal with nondiscretionary access control. One reason for the importance of nondiscretionary models is that formal statements about their security can be made.[17] Multilevel models differ as well in treating not only access to information, but also the flow of information within a system. Like discretionary models, multilevel models were developed for operating systems and later applied to data base systems.

In this section we describe a simplified version of the model developed by Bell and La Padula.[18,19] This model introduces the concepts of *level* and *category*. Each subject is assigned a clearance level, and each object a classification level. For the military environment, these levels might be *top secret, secret, confidential*, and *unclassified.* A subject generally represents a process being executed on behalf of a user and having the same clearance level as the user. The objects can be areas of storage, program variables, files, I/O devices, users, or anything else that can hold information. Each subject and object also has a set of *categories*, such as *nuclear* or *NATO.* A *security level* is a composite:

(classification level, set of categories).

Figure 9 Ordering of security levels



One security level is said to *dominate* another if, and *only* if, its classification or clearance level is greater than or equal to the other *and* its category set contains the other. Clearance and classification levels are ordered (for example, *secret* > *confidential* > *unclassified*), but security levels are only partially ordered, so some subjects and objects are not comparable. For example in Figure 9, security level L1 dominates security level L2 since its classification level is higher and its set of categories includes the set of categories of L2. Security levels L1 and L3, on the other hand, are not comparable. The elements of the model are summarized in Table 3.

An access of an object can either *observe* the object (extract information from it) or *alter* the object (insert information into it). The set of access types is determined, then, by all the possible combinations of these effects. The access types are:

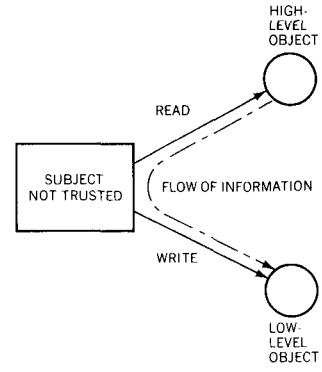
- Neither observe nor alter.
- Observe only (READ).
- Alter only (APPEND).
- Observe and alter (WRITE).

The model considers the *states* of a secure system, which are described by:

- The current access set, which is a set of triples (subject, object, access type), or (s, o, t).[20]
- An access matrix.
- The security level of each object.
- The maximum and current security levels of each subject.

Note that the system state for these models does not include the values in the data base.

Figure 10 Information flow showing need for the *-property



changing system state

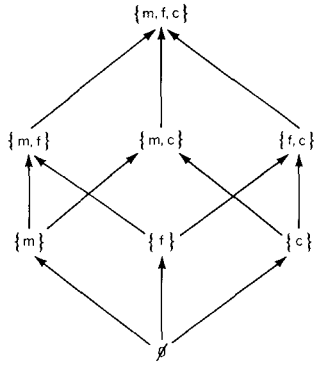
Any change to the system's state is caused by a *request*. Requests can be for access to objects, for changes to security levels or to the access matrix, or to create or destroy objects. The system's response to a request is called a *decision*. Given a request and a current state, the decision and the new state are determined by a *rule*. (Rules here correspond to the validation rules of the discretionary models, not to the access rules.) These rules of system operation prescribe how each type of request is to be handled. Proving that a system is secure involves proving that each rule is security-preserving. Then, if the system state is secure, any request will result in a new secure state.

secure system states

A secure state is defined by two properties: the *simple security property*, and the **-property* (which has also been called the *confinement property*). The simple security property is: for every current access (s, o, t) with an *observe* access type, the level of the subject dominates the level of the object. This condition can be expressed as *no reading upward in level.*

The simple security condition does not prevent a combination of accesses, each secure in itself, from providing a potential for compromise. As can be seen from Figure 10, a malicious subject could extract information from a *top secret* object and put it into a *confidential* object. The *-property is introduced to prevent such flow of higher-level information into lower-level objects. The *-property is defined as follows: a current access (s , o , t) implies that

Figure 11 A lattice model (adapted, with permission, from Denning[21])



- If t = READ, level (o) is dominated by current level (s).
- If t = APPEND, level (o) dominates current level (s).
- If t = WRITE, level (o) equals current level (s).

The simple security and *-properties represent nondiscretionary security, in which access is governed by the level of the subject and object. The *discretionary security property* is satisfied if every current access is authorized by the current access matrix.

Denning[21] has treated information flow aspects of the multilevel models in a more general way. The concepts of classification and category are subsumed under a single concept of security classes, and a variable *class-combining operator* is introduced in place of a fixed one. An information flow model that describes a specific system is defined by five components: a set of objects, a set of processes, a set of security classes, a class-combining operator, and a *flow relation*. The class-combining operator, $\oplus$, specifies the class of the result of any operation. For example, if we concatenate two objects, a and b, whose classes are A and B, the class of the result is A $\oplus$ B. A flow relation between two classes, for example A $\rightarrow$ B, means that information in class A is permitted to flow into class B. A flow model is secure if a flow relation cannot be violated.

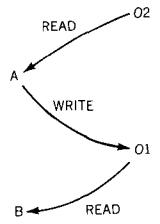
an information flow model

If certain reasonable assumptions are made, three components of the model (classes, $\oplus$, and $\rightarrow$) form a mathematical structure called a *lattice*. (These three components represent the authorization structure of a specific system.) A lattice consists of a partially ordered set, plus least-upper-bound and greatest-lower-bound operators. The lattice shown in Figure 11 represents a system that contains personal data of three types: medical (m), financial (f), and criminal (c). The classes shown are all the possible subsets of {m, f, c}; they represent combinations of the data types. Information flows (as shown by the arrows) only into classes at least as inclusive. Thus for this lattice, the class-combining operator $\oplus$, which is the least-upper-bound operator, yields the union of the two classes. A flow violation would occur, for example, on an attempt to move information produced from combining medical and financial data into the class designated medical only.

To guarantee that programs are secure, that they do not violate the information flow requirements expressed by a lattice model, both *explicit* and *implicit* flows must be considered. For example, the statement *if* $a = 0$ *then* $b = c$ produces an explicit flow from c to b when $a = 0$, but it *always* causes an implicit flow from a to b, since it is possible to determine whether $a = 0$ by examining b after execution of the statement. A program is secure if all explicit and implicit flows are secure.

A number of mechanisms have been proposed to enforce secure information flow.[21] They involve compile-time certification of programs or run-time enforcement (which may be supported by hardware), or combinations of the two.

Figure 12 Example—security policy



Comparison of models

Models can be classified broadly into two categories, those that are extensions of the access matrix approach and those that control information flow. An advantage of access matrix models is their flexibility in allowing a wide range of security policies to be specified easily. For example, type-dependent and content-dependent access rules can be represented simply. The main disadvantage is that the flow of information is not controlled.

As an illustration, suppose the security policy of an enterprise is to provide user A with READ access to object $O2$ and WRITE access to object $O1$, and user B with only READ access to object $O1$ (see Figure 12). While this policy can be represented by an access matrix (Figure 13), there is nothing to prevent A from copying $O2$ into $O1$ and thus allowing B to access the information in $O2$.

Figure 13 Example—access matrix

s	O	t	p
A	O2	READ	—
A	O1	WRITE	—
B	O1	READ	—

This illegal flow of information is prevented in the second category of models. However, because of the structuring of the multilevel model it is not possible to represent arbitrary security policies. For example, the simple policy that allows A to access $O3$ and $O2$, B to access $O2$ and $O1$, and C to access $O1$ and $O3$ cannot be handled. Denning's model, with its more general approach of partially ordering classes, can handle this situation. The creation of new data base objects with new security requirements, however, may require a complete restructuring of the class lattice. Moreover, type-dependent and content-dependent access rules cannot be represented simply. If program variables can change security class during execution, compile-time analysis does not suffice. Control of information flow then requires execution-time checks, which may cause unacceptable overhead. In summary, the two approaches represent different compromises between efficiency, flexibility, and security.

Applying the models to data base systems

application of the basic model

There have been partial implementations of the basic model for access control but no implementation of the extended model. In System R,[14] for example, an authorization rule grants a user a certain type of access to a data object. In READ, INSERT, DELETE, and DROP (the ability to delete the entire object from the system) the object is a table (a base relation or a view), and in UPDATE the object can be a column of the table. Additionally, the grantee may

be authorized, in turn, to grant the access rights to other users. To recall granted rights, it is necessary to keep track of who authorized the grant. In summary, an authorization rule can be represented as (a, s, O, t, f). (As a view can be defined by using an arbitrary query, there is no requirement for a predicate in the access rule.)

Part of IMS security can be approximately characterized by the authorization rule (s, O, p), in which s is the user ID as verified by RACF (the Resource Access Control Facility),[22] O is a transaction code, and p is a restricted predicate that can specify only the value of a password. The transaction code invokes an application program for which access types to certain segments have been specified. Therefore there is the additional rule (s, O, t), in which s is the transaction code, O is the segment type, and t is the access type. Other security features of IMS allow the restriction of certain transaction codes and system commands to a specified terminal (interpreting a terminal as a subject, and transaction codes and commands as objects).

Query-by-Example (QBE)[23] provides a uniform way of manipulating information on a screen for data description, queries, and authorization. Assume, for example, a table named EMP with fields NAME and MGR. To insert a new row into EMP, the user enters the row (using the INSERT command *I*) under the displayed column headings, as shown below:

EMP	*NAME*	*MGR*
I.	SMITH	JONES

Giving a user named JOHN print (*P*) access to the table is done similarly:

EMP	*NAME*	*MGR*
I.AUTH(P.)JOHN	N	M

The underlines indicate that N and M are *example* elements, representing *any* name or manager. To give JOHN access only to his employees, the authorizer would enter:

EMP	*NAME*	*MGR*
I.AUTH(P.)JOHN	N	JOHN

The QBE authorization language has essentially the power of the QBE query language, since authorization and queries are expressed in the same way. Thus an access rule in QBE has the form (s, O, t, p), in which O is the name of a table or column and p is an arbitrary predicate.

The multilevel model has been applied to data base system design at I. P. Sharp Associates,[10] SRI International,[24] the Systems Development Corporation,[25] and the MITRE Corporation.[26] The de-

signs all assume relational data bases, but they differ in whether the protected objects are domains or entire relations. The Sharp design decomposes each data object into three components: a set of values, a descriptor (which gives the format of the values), and a permission matrix which lists the authorized users of the object and their permitted types of access. Directories and lists of active users are also protected objects. Access to all of these kinds of objects must obey the rules of the model. MITRE has implemented a version of INGRES[6] using a special version of the UNIX operating system,[27] which enforces a multilevel policy.

Summary

We have indicated some of the wide range of security policies an enterprise may wish to implement, and we have described models that could be used in designing a data base system flexible enough to support many of those security policies. Additionally, we have shown how models can be used for the formal specification of data base access rules and in comparing the security features of existing data base systems.

CITED REFERENCES AND NOTES

1. D. E. Denning and P. J. Denning, "Data Security," *ACM Computing Surveys* **11,** No. 3, 227-249 (September 1979).
2. E. B. Fernandez, R. C. Summers, and C. Wood, *Principles of Data Base Security*, The Systems Programming Series, Addison-Wesley Publishing Company, Inc., Reading, MA (to be published).
3. Figure 2 is adapted, with permission, from W. H. Ware, *Security Controls for Computer Systems*, Report R-609-1, RAND Corporation (1979).
4. *ANSI/X3/SPARC. DBMS Framework Report of the Study Group on Data Base Management Systems*, D. Tsichritzis and A. Klug (editors), AFIPS Press, Montvale, NJ (1977).
5. W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System," *Communications of the ACM* **17,** No. 6, 337-345 (June 1974).
6. M. Stonebraker and E. Wong, "Access Control in a Relational Data Base Management System by Query Modification," *Proceedings, ACM 74* (ACM Annual Conference, November 1974), pp. 180-187.
7. *IMS/VS Version 1 General Information Manual*, IBM Systems Library, order number GH20-1260, available through IBM branch offices.
8. D. E. Denning, P. J. Denning, and M. D. Schwartz, "The Tracker: A Threat to Statistical Data Base Security," *ACM Transactions on Database Systems* **4,** No. 1, 76-96 (March 1979).
9. H. R. Hartson and D. K. Hsiao, "A Semantic Model for Data Base Protection Languages," *Proceedings, Second International Conference on Very Large Data Bases*, North-Holland Publishing Company, Amsterdam (1976).
10. G. Kirby and M. Grohn, "The Reference Monitor Technique for Security in Data Management Systems," *Data Base Engineering* **1,** No. 2, 8-16 (June 1977).
11. B. W. Lampson, "Protection," *Proceedings, 5th Annual Princeton Conference on Information Sciences and Systems*, 437-443 (1971), reprinted in *ACM Operating Systems Review* **8,** No. 1, 18-24 (January 1974).
12. G. S. Graham and P. J. Denning, "Protection—Principles and practice," *AFIPS Conference Proceedings* **40,** 417-429 (1972).

13. E. B. Fernandez, R. C. Summers, and C. D. Coleman, "An Authorization Model for a Shared Database," *Proceedings, 1975 SIGMOD International Conference*, 23–31, Association for Computing Machinery, 1133 Avenue of the Americas, New York (1975).
14. P. P. Griffiths and B. W. Wade, "An Authorization Mechanism for a Relational Database System," *ACM Transactions on Database Systems* **1,** No. 3, 242–255 (September 1976).
15. D. D. Chamberlin, J. N. Gray, and I. L. Traiger, "Views, authorization, and locking in a relational data base system," *AFIPS Conference Proceedings* **44,** 425–430 (1975).
16. R. C. Summers and E. B. Fernandez, *Data Description for a Shared Data Base: Views, Integrity, and Authorization*, IBM Los Angeles Scientific Center Report (August 1975), order number G320-2671, available through IBM branch offices.
17. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," *Communications of the ACM* **19,** No. 8, 461–471 (August 1976).
18. D. E. Bell and L. J. La Padula, *Secure Computer System: Unified Exposition and MULTICS Interpretation*, Report ESD-TR-75-306, MITRE Corporation, Bedford, MA (March 1976).
19. J. K. Millen, "Security Kernel Validation in Practice," *Communications of the ACM* **19,** No. 5, 243–250 (May 1976).
20. Bell and La Padula[18] use different symbols: (*S*, *O*, *a*).
21. D. E. Denning, "A Lattice Model of Secure Information Flow," *Communications of the ACM* **19,** No. 5, 236–243 (May 1976).
22. *OS/VS2 MVS Resource Access Control Facility (RACF) General Information Manual*, IBM Systems Library, order number SC28-0722, available through IBM branch offices.
23. M. M. Zloof, "Query-by-Example: a data base language," *IBM Systems Journal* **16,** No. 4, 324–343 (1977).
24. P. G. Neumann et al., *A Provably Secure Operating System: The System, its Application, and Proofs*, Stanford Research Institute (now SRI International), Menlo Park, CA (1977).
25. T. H. Hinke and M. Schaefer, *Secure Data Management System*, Report TM-(L)-5407/007/00, Systems Development Corporation, Los Angeles, CA (June 1975).
26. B. N. Wagner, *Implementation of a Secure Data Management System for the Secure UNIX Operating System*, NTIS accession number AD-A056 902, Report ESD-TR-78-154, MITRE Corporation, Bedford, MA (July 1978).
27. *UNIX* is a trademark of Bell Laboratories. See *The Bell System Technical Journal* **57,** No. 6, Part 2 (July-August 1978).

C. Wood and R. C. Summers are located at the IBM Scientific Center, 9045 Lincoln Boulevard, Los Angeles, CA 90045. E. B. Fernandez is at the Data Processing Division Branch Office, 2200 Whitney Avenue, Hamden, CT 06518.

Reprint Order No. G321-5124.