DEPARTMENT OF OPERATIONS RESEARCH
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK

TECHNICAL REPORT NO. ~~135~~ 71-120

June 1971

Revised December 1971

ON THE IMPLEMENTATION OF SECURITY MEASURES IN INFORMATION SYSTEMS

by

R. W. Conway, W. L. Maxwell, and H. L. Morgan

ON THE IMPLEMENTATION OF SECURITY MEASURES IN INFORMATION SYSTEMS[+]

R. W. Conway, W. L. Maxwell, and H. L. Morgan

Abstract: The security of an information system may be modeled by a matrix whose elements are decision rules and whose row and column indices are users and data items respectively. A set of four functions are used to access this matrix at translation and execution time. Distinguishing between data dependent and data independent decision rules enables one to perform much of the checking of security once at translation time rather than repeatedly at execution time. The model is used to explain security features of several existing systems, and serves as a framework for a proposal for general security system implementation within today's languages and operating systems.

Keywords: security, privacy, access control confidentiality, operating systems, access management, data banks, management information systems.

CR Categories: 3.50, 4.39, 4.22, 4.12, 3.73

[+] This report has been accepted for publication in the Communications of the ACM.

ON THE IMPLEMENTATION OF SECURITY MEASURES IN INFORMATION SYSTEMS

R. W. Conway, W. L. Maxwell, and H. L. Morgan+
Cornell University*

A. Introduction

There are two important issues involved in the growing discussions concerning the control of access to privileged information stored in computer files. Although there has not been much consistency of terminology among those writing in the field, we would propose identifying these issues in the following way:

Information privacy involves issues of law, ethics, and judgement. Whether or not a particular individual should have access to a specific piece of information is a question of information privacy. As computer professionals and citizens, we share the concern of many over the privacy question, but have no especial wisdom to bring to bear.

Information security involves questions of means--procedures to ensure that privacy decisions are in fact enforceable and enforced. On this issue, however, it is we as computer professionals who are primarily responsible to society. We must provide a technology that is sufficiently rugged to resist determined attack, and sufficiently economical to encourage use. We must also insist that such a security system be an integral part of any information system containing potentially sensitive data (e.g., personnel, credit bureau, law enforcement), or the computing profession will soon find itself with the same problems of conscience that nuclear physicists suffered in the late forties.

---

* Department of Operations Research and Department of Computer Science

+ Present Address: Information Science Dept., California Institute of Technology, Pasadena, California 91109.

Figure 1 shows the outcomes of each possible combination of privacy decision and security action. Each of the possible outcomes has a cost, in both economic and social terms. In most information systems to date the privacy decision has not been made explicit and security has not been deliberately implemented--so that "proper access" is the only outcome. One might hypothesize that the privacy question has been neglected at least in part due to a tacit understanding that the requisite security measures were either infeasible or uneconomic. Security measures have been rudimentary, primarily because of preoccupation with the mechanics of providing access to information for any user, but perhaps partly because privacy demands have not been emphasized. This state of affairs cannot exist much longer, as it seems very likely that public and legal pressure will soon demand that the privacy question be made explicit [15,19]. Security measures will be required in order to limit the frequency of the "successful invasion" outcome, and these measures will have to have considerable flexibility in order to avoid the "improper rebuff" outcome.

		Privacy decision	
		Access permitted	Access denied
Action of security system	Access obtained	proper access	successful invasion
	Access prevented	Improper rebuff	successful defense

Figure 1. Outcomes of security system actions.

The purpose of this paper is to discuss the nature of flexibility in a security system and to relate the costs of implementation and enforcement to that flexibility. It appears possible that earlier treatments of the subject [12,17] have not sufficiently emphasized the distinction between data-dependent and data-independent privacy decisions and as a consequence have overestimated the cost of security enforcement.

A conceptual model consisting of a "security matrix" and four security functions is proposed. This model is then used in both the explanative and normative senses, i.e., several existing security systems are related to the model, and a general implementation method is proposed based on the use of the model. We start by examining the privacy decisions which may be desired in an attempt at classification.

B. Selective Security

Most of the technical difficulty in information security arises from the fact that those responsible for making the privacy decisions with respect to a particular data bank should not be constrained to have to either permit or deny access to the entire data bank. It is not sufficient to simply partition the population of users into two disjoint and exhaustive subpopulations--those who are permitted access and those who are denied access with respect to a particular data bank. Rather they should be selective in deciding just what portions of the data should be accessible to each user.

Perhaps the point can be made by analogy to physical access to a large office building. The simplest privacy decision would be to enumerate those individuals who are entitled to have access to the building. This could be implemented by a security system that consisted of a lock on the outer door. In practice this is rarely adequate, and a much more selective privacy decision is made, and then implemented by a complex security structure that involves locks, keys, guards and badges applied to the building, individual floors and corridors, offices, desks, safes, filing cabinets, etc. Going further one might wish not only to control where an individual could go within the building but also what he might do at each location. One could also want to make the regions that are accessible to a particular individual dependent upon the time of day. And finally, although the analogy is beginning to labor under the strain, one could want an individual's accessible region to depend upon some dynamic characteristic of the space units. For example, a man is to have access to all offices that do not at the moment contain any "classified" documents.

The variety of potential conditions for privacy of information is even broader. For example, imagine an employee personnel/payroll file for a large industrial concern. It might include data structures named NAME, SALARY HISTORY, CURRENT SALARY, PERFORMANCE EVALUATION, DEPARTMENT, MEDICAL HISTORY, and SOCIAL SECURITY NUMBER. One could conceive of circumstances under which each of the following would be a reasonable privacy decision with respect to some individual and this personnel/payroll file:

1. He has complete access to the entire file, for any purpose and action.
2. He has no access to any part of the file for any purpose.
3. He may see any portion of the file, but change none of its contents.
4. He may see exactly one record (his "own") of the file, but not alter its contents.

5. He may see exactly one record (his "own") of the file, and alter some, but not all of the fields of that record.
6. He may see only the NAME and MEDICAL HISTORY portion of each record in the file, and alter only the MEDICAL HISTORY portion.
7. He may see and alter only "financial" portions of each record in the file, but only during the hours of 9 AM to 5 PM and only from a terminal located in the payroll office.
8. He may see and alter only financial portions of each record, and only for those records for which the value of CURRENT SALARY is less than $15,000.
9. He may see financial information--but only in the aggregate, e.g., total salary by division, but not individual salaries.
10. He may see and alter PERFORMANCE EVALUATION only for those records for which the value of DEPARTMENT is 'Engineering'.

Although this list is by no means exhaustive it should begin to illustrate the variety of privacy conditions that will arise once people start giving serious consideration to the question. Each of the examples given above could be useful, or even essential, under certain circumstances.

To the best of our knowledge no security system exists today that would permit a system designer to enforce any arbitrary combination of these privacy constraints for any arbitrary combination of user and data element. With the present state of the art such a security system would increase overall processing cost by an order-of-magnitude--which may threaten the economic justification of the entire application, or at least tempt the designer to risk the consequences of an unsecured system. However, many of these privacy constraints have the important characteristic of being <u>independent of particular values of data</u>, and to the extent that such privacy constraints are considered adequate by the user it is possible to implement a security system at very modest cost. It seems to us important that the designers and implementers of security systems be aware of the crucial significance of data-dependence in security matters, and of the special procedures that may be employed to implement data-independent privacy requirements in an efficient manner. It is probably equally important that those responsible for making privacy decisions understand what kind of conditions are inherently difficult (and expensive) to implement so that these can be avoided wherever possible. This question is discussed in Section D.

C. The Security Matrix

Conceptually, the privacy decisions for a particular data bank may be recorded in a "security matrix" (a generalization of the "user security profiles" of Bingham [4]). The columns of this matrix correspond to particular data structures in the system--not necessarily disjoint, and the rows of the matrix correspond to the potential users of the system. Each element in the matrix, d_{ij}, is a decision rule embodying a specific privacy decision, specifying the conditions under which user i is entitled to access the data structure j, and the actions that i is permitted to perform upon j. In this respect, the security system may be considered to be table driven by this matrix.

As an example, suppose that the information in a particular system is describable as the PL/I structure shown in Figure 2 (data types omitted). Let us also suppose that the users in the system are identified as follows:

A - Company President
B - Chief payroll clerk
C - Systems programmer
D - Doctor
E - Employee

```
DECLARE
   1 EMPLOYEE (1000)
   2 NAME
   2 SALARY_HISTORY (6)
       3 CLASS
       3 RATE
       3 AMT
   2 CURRENT_SALARY
   2 PERFORMANCE_EVALUATION (6)
       3 DEPARTMENT
       3 SUPERVISOR
       3 DATE
       3 PERFORMANCE
   2 MEDICAL_HISTORY
       3 BLOOD_TYPE
       3 TREATMENT_HISTORY (12)
            4 DATE
            4 DOCTOR
            4 TREATMENT (8)
                 5 DATE
                 5 MEDICATION
                 5 CONDITION
   2 SOCIAL_SECURITY_NUMBER
   2 ADDRESS
       3 ADDR_LINE_1
       3 CITY
       3 STATE
       3 ZIP
```

Figure 2.

Figure 3 shows a portion of the security matrix for this system. Notice that the column headings include both major and minor structure names, as well as specific field names. Such detail is clearly required to permit the selective security discussed in Section B. In a real system, columns would be needed not only for the data structures, but also for the program libraries, operating system, and the security matrix itself. To be truly selective, one must also be able to distinguish between different generations of the same file, and to designate subfiles by content, e.g., a column for EMPLOYEES WITH DEPARTMENT='PURCHASING'.

USER	EMPLOYEE	CURRENT SALARY	SALARY HISTORY	AMT	MEDICAL HISTORY
A	R,W	R,W*	R,W*	R,W*	R,W*
B	N	R,(W between 9 and 5)	R,W	R,W*	N
C	N	N	N	N	N
D	N	N	N	N	R,W
E					

Key:	
R	can read the field or any element of the structure
W	can change the field or any element of the structure
*	Implied by some other decision rule in the same row
N	can neither read nor change the structure or field

Figure 3. A Portion of the Security Matrix

The description of this conceptual matrix will not be pursued to the point where it might become operationally precise simply because it is prohibitively large. In most real world data processing situations, the number of users would be substantial and the number of data elements would be impossibly large. Nonetheless, the security matrix is a convenient model with which to describe the general security problem, and a convenient background against which to evaluate what is achieved by specific implementations.

To begin with, the matrix may be cited to indicate two limitations on the scope of this discussion. First, we are considering security measures only of the type that could be enforced through the use of such a matrix within a computer based system. This implies that we are specifically not treating the problem of physical security. In the world of locks, badges, and paper, a considerable technology has evolved over the years for the protection of valuable goods and information on paper. While the failures of physical security systems are not infrequent and are often newsworthy, it still seems possible to observe that adequate hardware and procedures for physical security are available to the data processing industry, if it were decided that the risks and costs justified their deployment [1,12].

One should note, however, that physical security in our context is mainly concerned with the central machine location. People accessing the system through remote terminals are constrained to submit to internal system security, and to the extent that this type of system is becoming more popular, the overall problem may have been eased. On the other hand, the user at a terminal is safely anonymous to the personnel at the central location, thus inhibiting use of the informal physical security measures which made it difficult for a stranger to walk into the computing center and obtain execution of a program accessing sensitive data.

Remote access systems are also subject to certain electronic threats involving wiretapping and masquerading, but until major advances are made in other aspects of security, these somewhat esoteric threats hardly seem to be an immediate concern to most installations [12,17].

The second restriction of the scope of this discussion concerns the problem of <u>authentication</u>. This is the problem of ensuring that the proper row of the security matrix is selected when a particular user addresses himself to the system. To be meaningful the security system must be designed under the assumption that an active potential infiltrator will exercise considerable ingenuity in the effort to misidentify himself to the system. The usual method of authentication is some variant of "password" protection, and the procedures used seem better suited to defend against a basically honest but occasionally befuddled user than against one who is seriously intent on penetrating security. More sophisticated (and costly) methods of authentication do exist (e.g., badge readers, "formula" passwords, built-in terminal identification) but are not yet in widespread use. Hoffman [12] has a discussion of authentication procedures, and Bingham [4] discusses possible hardware/software combinations for authentication.

For the remainder of this paper then, we shall assume that the system is physically secure, and that the particular user has been properly identified--while realizing that neither assumption is often justifiable. Yet one could argue that since potential violators will be concerned with the weakest link in an overall security system, potential protectors should be also. Considering the present state of current internal system security procedures at most installations, it is probably unnecessary for violators to waste time discovering passwords or risk the consequences of breaking and entering a computing center. However, if there are significant improvements in internal security, and if in fact there is an appreciable corps of enterprising violators, there will be increasing pressure upon provisions for physical security and authentication procedures.

The two compromises that characterize an embarassingly large fraction of the security procedures in current use might be described as "column" and "diagonal" systems. In a column system, there is only one data element--the complete data collection, and each decision rule in the resulting single column security matrix is a simple rule granting or denying unconditional and unrestricted access and use. The user has only to authenticate himself in order to obtain access to any datum in this collection. An associated accounting system may impose limits on the extent of use, but this is generally intended more to conserve or allocate system resources than to limit activity for security reasons.

In a "diagonal" system, the data collection is partitioned into a set of exhaustive and mutually exclusive files. Each file is uniquely identified with a particular user so that the rows and columns of the security matrix could be permuted so that all of the elements on the principal diagonal are decision rules granting unrestricted access and all other elements prohibit access.

Either of these special cases can be decomposed in such a way as to put the burden of access control solely on a password protection system. That is, each datum has a unique password, and a user consists of all the potential users who possess that password.

These special cases, however, are far from satisfying the selective security needs already described. In general, a practical implementation of the security matrix concept can be sought in one or more of three directions:

1. Reduction in the size of the matrix--by defining "virtual users", each representing a collection of users with identical security authorization; and by considering only data aggregates to reduce the number of columns.
2. Simplification of the entries in the matrix from the general "decision rule" to yes-no indication. This reduces the matrix from an array of functions which must be evaluated with each interrogation of the matrix to an array of bits.
3. Careful analysis of <u>when</u> and <u>how</u> the matrix should be interrogated and its specification employed.

Both the column and diagonal systems have carried out points 1 and 2 almost to their logical extremes. The third approach does not seem to have been examined in sufficient detail previously, yet seems to us to offer some promise for implementation of general security features at modest cost.

An analysis of how the security matrix is accessed shows that while the complete matrix may be very large, even in reduced form, it is also very sparse--in the sense that most of its entries are denials of access--and that only a very small fraction of the matrix is relevant at any instant of time. The authentication of a user selects a particular row of the matrix, and the user will designate some aggregate data element (e.g., when he "opens" a file) that will select a relatively small subset of the column entries for that row. Thus, rather than really being accessed randomly, a very small controlling sub-matrix could be identified and remain in control for a significant amount of processing time. This might alleviate some of the problems of the large size of the general matrix.

Similarly, when considering when the matrix is accessed, one should distinguish roughly between two approaches to data processing. The first is the traditional "batch processing" run, in which essentially similar actions are performed on large numbers of similar data structures in the course of a single run. The frequency of repetition demands that these actions be performed efficiently. The second class might be characterized as a "random inquiry" process. This would have a relatively low frequency of repetition, and one could tolerate significantly higher overhead. The first type of run has dominated the field in the past. The second is rapidly growing in importance. The point for present purposes is that a data collection must be secure with respect to both types of approach. The selection of the relevant security sub-matrix is required only <u>once per approach</u> and not once for each data element selected from the collection. In the random inquiry only one data access may be required, so that the addition of one reference to some form of security matrix represents a significant percentage increase in overhead, although the absolute amount of work involved is minor and entirely tolerable. On the other hand, the batch processing run could involve thousands of accesses into the data collection, but this need not require equally many examinations of the security matrix.

It is significant that what little security provisions exist in systems today are almost completely passive, i.e., the provision called "threat monitoring" in the literature [17] has rarely been adequately implemented. Current systems typically will submit to unlimited abuse without appealing for help or even noting the fact that they are under attack. In most cases a persistent infiltrator could sit at a terminal and try character combinations at random until a valid password was encountered. (MTS [16] at least inconveniences the infiltrator by requiring him to re-dial in after three password failures). While systems steadfastly deny access until a valid password is submitted, they generally do not recognize that a systematic assault is in progress. A successful threat, by definition, is not recognized as such, while in most cases an unsuccessful threat is neither detected nor punished. One must conclude that a security system is not really very serious until it includes at least some means of recording its successful defenses against attack.

Some recent systems include "entry logs" which record a history of authentication successes and failures. Threats that occur when a properly authenticated user attempts to violate the restrictions of his row in the security matrix could also be logged. Examination of such a log will reveal successful defenses, and if the log is monitored in real-time more imaginative and effective countermeasures could be deployed. We conclude that either very little ingenuity has been expended on the design of such countermeasures, or system designers have been judiciously quiet concerning their accomplishments in this area.

D. Data Dependent and Data Independent Conditions

It is informative to examine the examples of privacy decisions listed in Section B to see which require the actual value of some datum to be evaluated. Restricting a particular user from ever seeing a field named SALARY in any record of a file is independent of the specific values in that field, while restricting a user from seeing values of SALARY in excess of $10,000 is data dependent. Granting a user "read-only" access to an entire file is data independent, but permitting a user to alter the contents of only those records for which DEPARTMENT = 'PURCHASING' is data dependent. This distinction is crucial in determining how security restrictions are enforced, and what such enforcement will cost in system resources.

Data-dependence must be interpreted in a very general sense. The decision may be dependent upon the value of any datum in the system and not just the particular datum to which access is sought. For example, if a user is permitted access to salary data only between the hours of 9 and 5, then the current time is the datum on which the security decision depends.

Data-dependent privacy decisions obviously cannot be evaluated until the relevant data itself is available. This means not only that evaluation must be deferred until the system has accessed the data element but also that evaluation must be repeated for each potential data element in the same class. The implication is that an interpretive mode of enforcement performed at execution time is required, increasing by an order-of-magnitude the execution time in comparison with an unsecured version of the same request for information. To the extent that the necessary privacy decision is data-dependent this high cost is inescapable, and one simply has to balance the risks of running unsecured against the cost of providing security. However, not all privacy decisions are data-dependent, and it is possible to implement security enforcement for the data independent privacy decisions at very modest cost compared to that required for data-dependent decisions. Most writers and designers, noting that data-dependent privacy decisions can only be enforced interpretively at execution time, have apparently planned the enforcement of all privacy decisions in this way. This has given the erroneous impression that security enforcement is necessarily very costly, thereby dampening enthusiasm for implementing secure systems.

In fact, data-independent privacy decisions can be enforced by examining the request and the appropriate element of the matrix just once--at the time the request is received for translation (by a compiler, assembler, etc.). Access to the security matrix is required, but not access to the data base. Thus, if a user may never see the SALARY field of any record, a single check at translation time can stop the request from getting access to the data base in the first place. This method of enforcement assumes that:

(a) all requests for access to the data base are entered into the system as source input to a translator (assembler, COBOL, PL/I, MARK IV, ASAP, etc.) and

(b) the translated form of a request must be held in a secure manner so that it cannot be altered by the user after the translation-time checks have taken place.

Guaranteeing that these two assumptions are satisfied in a particular system may still be a less costly process than interpretively enforcing all of the data-independent privacy decisions.

The significance of data-dependency in other contexts is, of course, well known. In the design of language translators and operating systems there are numerous decisions which permit some freedom in the timing of implementation--for example, the "binding" of variables in a translator. In general, efficiency is served by early implementation and flexibility is served by later implementation. This is implicit in the translator-writer's maxim: "as soon as possible" or "as late as necessary". We simply observe that the same distinction and choice of implementation time exists with respect to security decisions. The next section describes a security model that emphasizes this distinction.

E. A Functional Model of a Security System

A model of a security system includes the security matrix and four functions: F_t and S_t, the translation-time fetch and store functions, and F_r and S_r, the run-time fetch and store functions.

Each of the functions has two arguments--u, a user identification and d, a datum name. The functions thus reflect a particular element of the security matrix. F_t is called whenever during the translation of a request (job, task, etc.) a fetch reference is made to a data element, i.e. in the right-hand-side of an assignment statement or appearance in an output list. $F_t(u,d)$ interrogates the specified element of the matrix and takes one of three possible actions:

1. If the user is permitted data-independent read access to the datum, then conventional object code for fetching the datum is generated.

2. If the user is denied read access to this datum on a data-independent basis, translation is aborted and the system is alerted to perform threat monitoring.

3. If the user is permitted data-dependent read access to the datum, a call to $F_r(u,d)$ is generated in the object code.

Similarly, S_t is invoked whenever a left-hand-side (store) usage of a datum is encountered, and has the obvious analogous actions.

$F_r(u,d)$ is called at execution time and actually performs the data-dependent check required. If the check passes, F_r returns the value of d. If the check fails, it returns a null of some sort. Similarly, $S_r(u,d)$ stores a value in d only if the data-dependent check is successful. Note that F_r should not invoke threat monitoring since it is expected to legitimately and innocently fail. However, failure of S_r generally suggests an attempted invasion and threat monitoring should be invoked.

To the extent that the security matrix specifies data-independent privacy decisions, security enforcement can be completely achieved by the use of F_t and S_t--the run-time functions F_r and S_r are not required. F_t and S_t represent very slight incremental effort for a translator, and when calls upon F_r and S_r are not required there is no degradation of execution performance. Hence data-independent privacy conditions can be implemented at modest cost. Data-dependent privacy conditions require the exercise of F_r and S_r for each datum and incur an appreciable execution-time penalty.

The principal point is that if one fails to distinguish between data-dependent and data-independent privacy conditions then F_t and S_t must always insert a call upon F_r and S_r and the substantial penalty of interpretive execution must always be paid. If the distinction is made, and F_t and S_t are sophisticated enough to capitalize upon it, then at least data-independent privacy can be economically enforced.

Another aspect of security is the encrypting of the data-base. It is a simple matter to include instructions in S_t and S_r to translate the datum into unintelligible form as it is stored, and corresponding instructions in F_t and F_r to translate the encrypted datum back into useful form. Simple and efficient algorithms for this translation have been given [17,18,5] and the execution cost on contemporary machines is quite modest [13]. Encrypting a data-base in this manner effectively thwarts the invader who would circumvent the security provisions of the system by obtaining access to the data-base entirely outside the managing system. While the translation would probably succumb to the efforts of a skilled cryptographer the "work factor" [3] is probably sufficiently high so as to make other avenues of attack more attractive.

F. <u>Examples of Existing Implementations</u>

While exemplary information systems with flexible, rugged and efficient security provisions may exist, we do not know of published descriptions. The tools generally available for system implementation--languages, data or file management systems and operating system utilities--do not currently encourage or facilitate elaborate security measures. The 1969 Survey of Generalized Data Base Management Systems [7] described nine systems. While seven of these have some form of security provisions, only three of the seven were actually operating when the survey was written. The most complex security available in the running systems was that of ADAM, which permitted only read/write protection on specified fields. The most widely used file management system, MARK IV, offered no security provisions. It is interesting to note that in NIPS/FFS, a file maintenance system for the National Military Command System, security provisions are completely described by the following paragraph:

> "Each file may be assigned a classification. If the classification given in a report specification (which is printed on each page of the report) does not match this, or none is given, a warning page precedes the report, and a console message is printed."

In the more recent CODASYL study [8], two new systems are mentioned. IBM's IMS system, which seems to include somewhat broader security provisions, and the Data Base Task Group proposal [6] which, if implemented, would offer a comprehensive set of privacy controls on actions and access.

We will briefly describe three university-developed systems. These represent some interesting ideas related to the previous discussion. None is widely used, and we do not suggest that they adequately represent current practice.

Hoffman [13] has implemented a security system for the Student Health System (SHS) at Stanford University, based on his procedural (or "formulary") model for access control and privacy in information systems. In this system, all security is provided at run-time through a set of data base access procedures. Each request for a datum must go through the procedure ACCESS, which in turn calls a procedure (contained in the formulary) specific to the user to obtain the grant/deny access decision. Privacy transformations can then be performed by ACCESS before giving the user the desired value (or storing the value in the data base). While the formulary could contain any general procedures, and ACCESS could be asked for specific field of records, in fact in the SHS system, the only formulary procedures which exist provide fetch or fetch and store access, and the only data name which ACCESS recognizes is NEXTRECORD. Thus, in effect, this conceptually very powerful system in current implementation is able to grant or deny read only or read/write access to the entire file. SHS therefore is an example of a column system, with the potential for rather complex formularies (decision rules). The ACCESS procedure is the implementation of F_r and S_r, and F_t and S_t always provide calls to F_r and S_r.

Graham [11] has described the protection scheme used in the MULTICS system. The use of special hardware features to effect some of the security checks is discussed. In MULTICS, there is a hierarchy of files and programs. Each segment (program or file) has a clearance level. This level is placed in a hardware register when the program is in control. If the program calls another program (or accesses a file) which is not at the same security level or a lower level (i.e. more restricted), a hardware interrupt is taken. The interrupt routine must then examine a "gate list", or list of acceptable users who may access this less restricted segment. Thus, when an access request is made to a segment, the "gatekeeper" program checks to see that the requestor is on the segment's "gate list". If so, the security levels of the two segments are compared, and the requestor is permitted access only if his program has a sufficient level of clearance. The "gatekeeper" distinguishes types of access as read, write, and execute. In this system, the security matrix is stored in the form of a set of gate lists, which indicate whether user i can access datum j for read/write/execute purposes. All checks are made at run-time, but the F_r and S_r functions are made relatively efficient through the hardware provisions.

The third example is a file maintenance system called ASAP [2,9] which was designed and implemented by the authors to serve as a test vehicle for a number of new concepts. Security measures were a primary consideration in the design. It is described at some length below since it, not surprisingly, illustrates most of the points of the previous discussion. Both data-dependent and data-independent security features are provided, although there is no threat monitoring. The data-independent features can deny access to specific fields of records, and can inhibit certain types of processing. These are implemented through F_t and S_t functions used during translation, thereby reducing the amount of checking which must be done at run-time.

The security matrix is contained in a special dictionary, which is created by the user, and contains descriptions of the files, reports, and other entities known to the system, as well as the security information. Associated with each field name is a security "class". There are eight such classes, labeled 1 to 8, with no hierarchy implied. Similarly, each aggregate name (e.g., a report containing information from several fields), has a class equivalent to the union of all of the classes of fields in the aggregate. Each allowable user of the system has an associated list of those classes to which he is permitted access. Thus, when he references a field, a simple comparison is made to determine whether or not he can see that field. In addition, there is associated with each user a list of the ASAP actions (PRINT, CALL, UPDATE, SET, etc.) that he is permitted to use.

A limited form of data-dependent security is provided by associated a set of qualifications (e.g. Boolean conditions) with each user. During execution, each record is checked against the appropriate qualification before being passed to the user's program for processing.

For example, consider a law firm's personnel file as shown in Figure 4. These definitions would create a dictionary which could process two different record types--employees and partners. There are three different 'users' permitted to use this system, and they authenticate themselves by prefacing their processing request with their respective passwords. A user authenticated as a 'LIMITED PARTNER' can examine the records for EMPLOYEES whose SALARY is less than $30,000, including all fields of the record except PMEDHIST, for PARTNERS whose PLEVEL = 'LP'. He cannot alter any field of any record. A user authenticated as a 'GENERAL PARTNER' can read or alter any field of any record. He is also authorized to call external procedures from ASAP and to add new users to the system. A user authenticated as a 'PHYSICIAN' has access to all records, but only to the unrestricted identification information and the medical history.

```
DEFINE RECORD EMPLOYEES
    NAME 25
    SOCSECNO 9 KEY
    DEPT 5
    SALARY 7 COMPUTE    SECURITY 1
    MEDHIST 30          SECURITY 2
    JOBEVAL 30          SECURITY 1
DEFINE RECORD PARTNERS
    PNAME 25
    PSOCSECNO 9 KEY
    PLEVEL 2
    PSALARY 7 COMPUTE   SECURITY 1
    PCTOWNER 3 COMPUTE  SECURITY 3
    PMEDHIST 30         SECURITY 2
DEFINE USER 'LIMITED PARTNER' SECURITY 1
    EMPLOYEES WITH SALARY > 30000
    PARTNERS WITH PLEVEL = 'LP'     'PASSWORD 1'
    USER 'GENERAL PARTNER' SECURITY 1 2 3 UPDATE CALL USER 'PASSWORD 2'
    USER 'PHYSICIAN' SECURITY 2 UPDATE 'PASSWORD 3'
```

Figure 4.

The system operates in the following way. If a LIMITED PARTNER were to submit the ASAP request:

```
ASAP 'PASSWORD 1'
FOR ALL EMPLOYEES WITH DEPT = 'ACNTG'
     PRINT A LIST OF NAME, SALARY, JOBEVAL
     ORDERED BY SALARY.
```

the system would simply supply only those records for which the value of SALARY is less than $30,000. There is no indication on the results that certain records were not accessed. This is a data-dependent condition and is enforced at run-time. If this same LIMITED PARTNER were to submit any of the following requests:

```
FOR ALL EMPLOYEES SELECTED BY KEY IN DATA CARDS,
          UPDATE RECORD.
FOR ALL EMPLOYEES PRINT LIST:  NAME, MEDHIST.

FOR ALL EMPLOYEES WITH NAME = 'JOHN A. JONES'
          INCREASE SALARY BY 1000.
```

these would be detected at translation time and no execution would take place. All of these are violations of data-independent privacy conditions and are trapped by F_t or S_t during compilation.

Clearly, the ability to define users is all important and should be reserved to only one or two people. There are built-in countermeasures to some of the obvious threats to this system. Both files and secure information in the dictionary are stored in encrypted form (file encrypting is optional) considerably increasing the work factor to obtain information from them by dumping them with a utility program.

We believe that the types of security and the implementation strategy employed in ASAP are not dependent upon the characteristics of this particular source language. They could be applied to a conventional language, or even to a multi-language environment, as described next.

G. General Implementation

The previous section gave examples of security implementations in an application program, a specialized operating system, and a generalized data-base management system. It is important to consider the security question relative to languages such as COBOL, FORTRAN and PL/I and relative to operating systems such as OS/360. If privacy decisions cannot be practically enforced in this environment then discussions of security and privacy are somewhat limited, or at least premature. We believe that this is not the case and that security enforcement is practical--although we do not know of this having been done in a general, flexible manner.

It would appear to be quite feasible to provide security similar to that offered by ASAP either by modifying or extending a general purpose operating system. The major task would be the addition of a library routine (analogous to the ASAP dictionary) to store, manage and protect the security matrix. Secondly the I/O service routines (OPEN, CLOSE, GET, PUT) of the operating system would be modified, or have other routines superimposed upon them. These routines would represent the F_r and S_r functions. If implemented in this manner it would be independent of the translator employed, and since security is enforced at run-time there is no requirement to secure the translated request during the interval between translation and execution. On the other hand it means that all privacy conditions are treated as data-dependent conditions with the attendant overhead. In fact the overhead is even greater than that involved in data-dependent conditions in a system such as ASAP for the F_r and S_r routines must perform complete run-time masking of the data elements. That is, since F_r has no way of knowing what the user program is going to attempt to do with the data element it must complete[illegible] the element according to the specification of the security matrix. This is essentially the Hoffman [13] security model, and it may be that the execution penalty is prohibitive in many situations.

To reduce this penalty one can identify those privacy conditions that are data-independent and implement the F_t and S_t functions so that when possible the requisite tests are performed once per request rather than once per data access. To do so means becoming involved in the translation process and, equally importantly, providing a means of securing the user request against modification between the time of translation and the time of execution.

F_t and S_t can be implemented either by modifying the standard compiler or by interposing a preprocessor in front of a standard compiler. The former is certainly the most efficient method of implementation, and perhaps some day standard compilers will include such provisions. Until that time one is faced with the not altogether pleasant prospect of making and maintaining non-trivial modifications deep in the interior of a sophisticated translator. At least at the moment, the preprocessor approach seems more reasonable. There already exist very high-performance compilers for both FORTRAN and PL/I that could be used as a basis for a security-preprocessor. One could discard the code-generation phase of these compilers and use the preliminary phases to implement security features. These would draw upon the same dictionary routine for the security matrix that is required for the data-dependent run-time enforcement. Although the preprocessor would have to perform rather complete syntactic analysis of the source program to perform the necessary checks, and this would duplicate the analysis performed by the standard compiler, the performance of these preprocessors is such that this could readily be tolerated. For example, a preprocessor for PL/I based on the Cornell PL/C compiler [10] would scan 15,000 PL/I source statements per minute on a 360/65. (This is the actual speed of the PL/C lexical and syntactic analyzer.).

With either a preprocessor or a modified compiler certain serious problems remain. When data elements are referenced by name there is basically little difficulty for these can be checked in context and usage against the security matrix. However, the programmer has a number of ways by which he can access data elements other than by name. The most obvious way is to use an array and deliberately force the subscripts to go outside of the declared bounds. PL/I offers a remedy for this problem, since the security system (either preprocessor or modified compiler) could insist that the program have SUBSCRIPTRANGE enabled with a suitably fatal ON SUBSCRIPTRANGE unit. This would, however, impose additional execution overhead for the subscript monitoring. Neither FORTRAN nor COBOL offer this type of monitoring and it is not possible for a preprocessor for those languages to protect against this circumvention if the use of arrays is allowed--as of course it must be. Either the standard compiler would have to be modified to provide this protection or run-time testing would be required.

A similar problem exists in the improper use of pointers in PL/I. A structure similar to the target record could be declared, and a pointer to the record set into a pointer to this structure. This could be negated by the run-time testing of the record (by F_r and S_r) but this is costly protection and it might be possible to detect this type of threat by sophisticated analysis of the source program. Similarly, the passing of a record address as a parameter to a subroutine in which the record is not declared, but where a similar structure is declared, presents the same type of problem.

In summary it would appear that a translator-independent F_r/S_r security system could be implemented by adding a security matrix routine and modifying the I/O service routines of a general purpose operating system, but this would treat all privacy conditions as data-dependent and the execution-time degradation would be severe. Performance could be improved by implementing the F_t/S_t functions by some combination of a preprocessor and modifications to the standard translators but this would also require some limitation on the freedom of the programmer to use the full facilities of the source language.

H. Conclusions

In current practice security measures in information systems are neither elaborate, flexible nor impenetrable. There are indications that more demanding privacy requirements will soon be imposed upon systems designers and that security provisions will have to be substantially improved.

It is useful to view security questions in terms of a security matrix and four functions representing translate and run-time fetch and store operations. In particular this model facilitates the delineation of data-dependent and data-independent security conditions. The former requires interpretive implementation in the form of run-time checking routines and impose a significant degradation of performance. The latter, at least in principle, needs to be exercised only once per request at translation time, rather than once per datum and thereby offer the opportunity to implement certain security measures with very modest execution penalty.

The best prospects for enhancement of security would seem to lie in the more recent generalized data management systems. These systems offer a controlled environment and sufficient restraint upon user activity to enforce reasonably general security conditions. Although many of these systems are interpretive with an execution penalty that is unacceptable for much production work at least one offers fairly general and flexible security conditions with implementation by a compiler that does, in fact, utilize translation-time checking of data-independent conditions. Hence feasibility has been demonstrated and users could require this type of capability from future data management systems.

Providing comparable security measures in a general-purpose language environment is a far more difficult task. Satisfactory measures can be incorporated into the data management facilities of general operating systems but this implies complete reliance upon run-time checking and the execution penalty would be severe. General-purpose languages such as PL/I, COBOL and FORTRAN provide programmers with too much flexibility to permit a translator to take advantage of data-independent security conditions at compilation time. Restricted versions of these languages that proscribe the constructions by which users could circumvent translation-time checks could be used and either special security-conscious compilers or pre-processors to conventional compilers could be employed.

To the extent that systems capable of realizing the efficiency of translation-time checking of security conditions become available, users, in particular those responsible for privacy decisions, should be made aware of the distinction between data-dependent and data-independent conditions. Insofar as is possible they could try to avoid specifying inherently difficult and expensive data-dependent conditions.

I. Acknowledgements

The authors wish to acknowledge the extremely helpful comments of Professor Ed Sibley of the University of Michigan.

References

1. Allen, Brandt, "Danger Ahead! Safeguard your computer." Harvard Business Review, Nov.-Dec. 1968, 97-101.

2. ASAP System Reference Manual. Compuvisor, Inc. 1971.

3. Baran, Paul, "On distributed communications: IX. Security, secrecy, and tamper-free considerations" RM-3765-PR, Rand Corporation, Aug. 1964.

4. Bingham, H.W., Security techniques for EDP of multilevel classified information. Document RADC-TR-65-415, Rome Air Development Center, Rome, New York, Dec. 1965 (unclassified).

5. Carrol, J.M. and P. M. McLelland, "Fast infinite key privacy transformation for resource sharing systems." Proc. AFIPS FJCC 1970.

6. CODASYL Data Base Task Group Report. October 1969. (Available from ACM.)

7. CODASYL Systems Committee, A Survey of Generalized Data Base Management Systems. May 1969. (Available from ACM).

8. CODASYL Systems Committee, Feature Analysis of Generalized Data Base Management Systems. May 1971 Report. (Available from ACM.)

9. Conway, R.W., W.L. Maxwell, and H.L. Morgan, "Selective Security Capabilities in ASAP - A file management system." Proceedings of the AFIPS 1972 Spring Joint Computer Conference (to appear).

10. Conway, R.W., H.L. Morgan, R. Wagner and T. Wilcox, User's Guide to PL/C. Dept. of Computer Science, Cornell University, 1970.

11. Graham, R.M., "Protection in an information processing utility". Comm. ACM 11 (May 1968).

12. Hoffman, Lance J., "Computers and Privacy: A survey." Computing Surveys 1, No. 2 (June 1969) 85-103.

13. Hoffman, Lance J., "The formulary model for access control and privacy in computer systems." SLAC Rept. No. 117., Stanford University, May 1970.

14. Martin, James, and A. Norman, The Computerized Society. Prentice-Hall, New Jersey, 1970.

15. Miller, A. R., Assault on Privacy: Computers, Data Banks, and Dossiers. University of Michigan Press, Ann Arbor 1971.

16. MTS User's Guide. University of Michigan Computer Center, 1970.

17. Petersen, H.E. and R. Turn, "System implications of information privacy," Proc. AFIPS 1967 SJCC Vol. 30, 291-300.

18. Skatrud, R.O., "The applications of cryptographic techniques to data processing." Proc. AFIPS FJCC 1969, 111-117.

19. Westin, A., Privacy and Freedom. Atheneum, New York, 1967.