I. BASIC PRINCIPLES OF INFORMATION PROTECTION

A. Considerations Surrounding the Study of Protection

1) General Observations: As computers become better understood and more economical, every day brings new applications. Many of these new applications involve both storing information and simultaneous use by several individuals. The key concern in this paper is multiple use. For those applications in which all users should not have identical authority, some scheme is needed to ensure that the computer system implements the desired authority structure.

For example, in an airline seat reservation system, a reservation agent might have authority to make reservations and to cancel reservations for people whose names he can supply. A flight boarding agent might have the additional authority to print out the list of all passengers who hold reservations on the flights for which he is responsible. The airline might wish to withhold from the reservation agent the authority to print out a list of reservations, so as to be sure that a request for a passenger list from a law enforcement agency is reviewed by the correct level of management.

The airline example is one of protection of corporate information for corporate self-protection (or public interest, depending on one's view). A different kind of example is an online warehouse inventory management system that generates reports about the current status of the inventory. These reports not only represent corporate information that must be protected from release outside the company, but also may indicate the quality of the job being done by the warehouse manager. In order to preserve his personal privacy, it may be appropriate to restrict the access to such reports, even within the company, to those who have a legitimate reason to be judging the quality of the warehouse manager's work.

Many other examples of systems requiring protection of information are encountered every day: credit bureau data banks; law enforcement information systems; time-sharing service bureaus; on-line medical information systems; and government social service data processing systems. These examples span a wide range of needs for organizational and personal privacy. All have in common controlled sharing of information among multiple users. All, therefore, require some plan to ensure that the computer system helps implement the correct authority structure. Of course, in some applications no special provisions in the computer system are necessary. It may be, for instance, that an externally administered code of ethics or a lack of knowledge about computers adequately protects the stored information. Although there are situations in which the computer need provide no aids to ensure protection of information, often it is appropriate to have the computer enforce a desired authority structure.

The words "privacy," "security," and "protection" are frequently used in connection with information-storing systems. Not all authors use these terms in the same way. This paper uses definitions commonly encountered in computer science literature.

The term "privacy" denotes a socially defined ability of an individual (or organization) to determine whether, when, and to whom personal (or organizational) information is to be released.

This paper will not be explicitly concerned with privacy, but instead with the mechanisms used to help achieve it.[1]

The term "security" describes techniques that control who may use or modify the computer or the information contained in it.[2]

Security specialists (e.g., Anderson [6]) have found it useful to place potential security violations in three categories.

1) Unauthorized information release: an unauthorized person is able to read and take advantage of information stored in the computer. This category of concern sometimes extends to "traffic analysis," in which the intruder observes only the patterns of information use and from those patterns can infer some information content. It also includes unauthorized use of a proprietary program.

2) Unauthorized information modification: an unauthorized person is able to make changes in stored information--a form of sabotage. Note that this kind of violation does not require that the intruder see the information he has changed.

3) Unauthorized denial of use: an intruder can prevent an authorized user from referring to or modifying information, even though the intruder may not be able to refer to or modify the information. Causing a system "crash," disrupting a scheduling algorithm, or firing a bullet into a computer are examples of denial of use. This is another form of sabotage.

The term "unauthorized" in the three categories listed above means that release, modification, or denial of use occurs contrary to the desire of the person who controls the information, possibly even contrary to the constraints supposedly enforced by the system. The biggest complication in a general-purpose remote-accessed computer system is that the "intruder" in these definitions may be an otherwise legitimate user of the computer system.

Examples of security techniques sometimes applied to computer systems are the following:

1. labeling files with lists of authorized users,
2. verifying the identity of a prospective user by demanding a password,
3. shielding the computer to prevent interception and subsequent interpretation of electromagnetic radiation,
4. enciphering information sent over telephone lines,
5. locking the room containing the computer,
6. controlling who is allowed to make changes to the computer system (both its hardware and software),
7. using redundant circuits or programmed cross-checks that maintain security in the face of hardware or software failures,
8. certifying that the hardware and software are actually implemented as intended.

It is apparent that a wide range of considerations are pertinent to the engineering of security of information. Historically, the literature of computer systems has more narrowly defined the term *protection* to be just those security techniques that control the access of executing programs to stored information.[3] An example of a protection technique is labeling of computer-stored files with lists of authorized users. Similarly, the term *authentication* is used for those security techniques that verify the identity of a person (or other external agent) making a request of a computer system. An example of an authentication technique is demanding a password. This paper concentrates on protection and authentication mechanisms, with only occasional reference to the other equally necessary security mechanisms. One should recognize that concentration on protection and authentication mechanisms provides a narrow view of information security, and that a narrow view is dangerous. The objective of a secure system is to prevent all unauthorized use of information, a negative kind of requirement. It is hard to prove that this negative requirement has been achieved, for one must demonstrate that every possible threat has been anticipated. Thus an expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy. In contrast, a narrow concentration on protection mechanisms, especially those logically impossible to defeat, may lead to false confidence in the system as a whole.[4]

2) Functional Levels of Information Protection: Many different designs have been proposed and mechanisms implemented for protecting information in computer systems. One reason for differences among protection schemes is their different functional properties--the kinds of access control that can be expressed naturally and enforced. It is convenient to divide protection schemes according to their functional properties. A rough categorization is the following.

a) Unprotected systems: Some systems have no provision for preventing a determined user from having access to every piece of information stored in the system. Although these systems are not directly of interest here, they are worth mentioning since, as of 1975, many of the most widely used, commercially available batch data processing systems fall into this category--for example, the Disk Operating System for the IBM System 370 [9]. Our definition of protection, which excludes features usable only for mistake prevention, is important here since it is common for unprotected systems to contain a variety of mistake-prevention features. These may provide just enough control that any breach of control is likely to be the result of a deliberate act rather than an accident. Nevertheless, it would be a mistake to claim that such systems provide any security.[5]

b) All-or-nothing systems: These are systems that provide isolation of users, sometimes moderated by total sharing of some pieces of information. If only isolation is provided, the user of such a system might just as well be using his own private computer, as far as protection and sharing of information are concerned. More commonly, such systems also have public libraries to which every user may have access. In some cases the public library mechanism may be extended to accept user contributions, but still on the basis that all users have equal access. Most of the first generation of commercial timesharing systems provide a protection scheme with this level of function. Examples include the Dartmouth Time-Sharing System (DTSS) [10] and IBM's VM/370 system [11]. There are innumerable others.

c) Controlled sharing: Significantly more complex machinery is required to control explicitly who may access each data item stored in the system. For example, such a system might provide each file with a list of authorized users and allow an owner to distinguish several common patterns of use, such as reading, writing, or executing the contents of the file as a program. Although conceptually straightforward, actual implementation is surprisingly intricate, and only a few complete examples exist. These include M.l.T.'s Compatible Time-Sharing System (CTSS) [12], Digital Equipment Corporation's DECsystem/10 [13], System Development Corporation's Advanced Development Prototype (ADEPT) System [14], and Bolt, Beranek, and Newman's TENEX [15][6]

d) User-programmed sharing controls: A user may want to restrict access to a file in a way not provided in the standard facilities for controlling sharing. For example, he may wish to permit access only on weekdays between 9:00 A.M. and 4:00 P.M. Possibly, he may wish to permit access to only the average value of the data in a file. Maybe he wishes to require that a file be modified only if two users agree. For such cases, and a myriad of others, a general escape is to provide for user-defined *protected objects* and *subsystems*. A *protected subsystem* is a collection of programs and data with the property that only the programs of the subsystem have direct access to the data (that is, the protected objects). Access to those programs is limited to calling specified entry points. Thus the programs of the subsystem completely control the operations performed on the data. By constructing a protected subsystem, a user can develop any programmable form of access control to the objects he creates. Only a few of the most advanced system designs have tried to permit user-specified protected subsystems. These include Honeywell's Multics [16], the University of California's CAL system [17], Bell Laboratories' UNIX system [18], the Berkeley Computer Corporation BCC-500 [19], and two systems currently under construction: the CAP system of Cambridge University [20], and the HYDRA system of Carnegie-Mellon University [21]. Exploring alternative mechanisms for implementing protected subsystems is a current research topic. A specialized use of protected subsystems is the implementation of protection controls based on data content. For example, in a file of salaries, one may wish to permit access to all salaries under $15 000.

Another example is permitting access to certain statistical aggregations of data but not to any individual data item. This area of protection raises questions about the possibility of discerning information by statistical tests and by examining indexes, without ever having direct access to the data itself. Protection based on content is the subject of a variety of recent or current research projects [22]-[25] and will not be explored in this tutorial.

e) Putting strings on information: The foregoing three levels have been concerned with establishing conditions for the release of information to an executing program. The fourth level of capability is to maintain some control over the user of the information even *after* it has been released. Such control is desired, for example, in releasing income information to a tax advisor; constraints should prevent him from passing the information on to a firm which prepares mailing lists. The printed labels on classified military information declaring a document to be "Top Secret" are another example of a constraint on information after its release to a person authorized to receive it. One may not (without risking severe penalties) release such information to others, and the label serves as a notice of the restriction. Computer systems that implement such strings on information are rare and the mechanisms are incomplete. For example, the ADEPT system [14] keeps track of the classification level of all input data used to create a file; all output data are automatically labeled with the highest classification encountered during execution.

There is a consideration that cuts across all levels of functional capability: the *dynamics of use*. This term refers to how one establishes and changes the specification of who may access what. At any of the levels it is relatively easy to envision (and design) systems that statically express a particular protection intent. But the need to change access authorization dynamically and the need for such changes to be requested by executing programs introduces much complexity into protection systems. For a given functional level, most existing protection systems differ primarily in the way they handle protection dynamics. To gain some insight into the complexity introduced by program-directed changes to access authorization, consider the question "Is there any way that O'Hara could access file X?" One should check to see not only if O'Hara has access to file X, but also whether or not O'Hara may change the specification of file X's accessibility. The next step is to see if O'Hara can change the specification of who may change the specification of file X's accessibility, etc. Another problem of dynamics arises when the owner revokes a user's access to a file while that file is being used. Letting the previously authorized user continue until he is "finished" with the information may not be acceptable, if the owner has suddenly realized that the file contains sensitive data. On the other hand, immediate withdrawal of authorization may severely disrupt the user. It should be apparent that provisions for the dynamics of use are at least as important as those for static specification of protection intent.

In many cases, it is not necessary to meet the protection needs of the person responsible for the information stored in the computer entirely through computer-aided enforcement. External mechanisms such as contracts, ignorance, or barbed wire fences may provide some of the required functional capability. This discussion, however, is focused on the internal mechanisms.

3) Design Principles: Whatever the level of functionality provided, the usefulness of a set of protection mechanisms depends upon the ability of a system to prevent security violations. In practice, producing a system at any level of functionality (except level one) that actually does prevent all such unauthorized acts has proved to be extremely difficult. Sophisticated users of most systems are aware of at least one way to crash the system, denying other users authorized access to stored information. Penetration exercises involving a large number of different general-purpose systems all have shown that users can construct programs that can obtain unauthorized access to information stored within. Even in systems designed and implemented with security as an important objective, design and implementation flaws provide paths that circumvent the intended access constraints. Design and construction techniques that systematically exclude flaws are the topic of much research activity, but no complete method applicable to the construction of large general-purpose systems exists yet. This difficulty is related to the negative quality of the requirement to prevent *all* unauthorized actions.

In the absence of such methodical techniques, experience has provided some useful principles that can guide the design and contribute to an implementation without security flaws. Here are eight examples of design principles that apply particularly to protection mechanisms.[7]

a) Economy of mechanism: Keep the design as simple and small as possible. This well-known principle applies to any aspect of a system, but it deserves emphasis for protection mechanisms for this reason: design and implementation errors that result in unwanted access paths will not be noticed during normal use (since normal use usually does not include attempts to exercise improper access paths). As a result, techniques such as line-by-line inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential.

b) Fail-safe defaults: Base access decisions on permission rather than exclusion. This principle, suggested by E. Glaser in 1965,[8] means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. The alternative, in which mechanisms attempt to identify conditions under which access should be refused, presents the wrong psychological base for secure system design. A conservative design must be based on arguments why objects should be accessible, rather than why they should not. In a large system some objects will be inadequately considered, so a default of lack of permission is safer. A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation, since it will be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure which may go unnoticed in normal use. This principle applies both to the outward appearance of the protection mechanism and to its underlying implementation.

c) Complete mediation: Every access to every object must be checked for authority. This principle, when systematically applied, is the primary underpinning of the protection system. It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised. It also requires that proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.

d) Open design: The design should not be secret [27]. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords. This decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers without concern that the review may itself compromise the safeguards. In addition, any skeptical user may be allowed to convince himself that the system he is about to use is adequate for his purpose.[9] Finally, it is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.

e) Separation of privilege: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. The relevance of this observation to computer systems was pointed out by R. Needham in 1973. The reason is that, once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information. This principle is often used in bank safe-deposit boxes. It is also at work in the defense system that fires a nuclear weapon only if two different people both give the correct command. In a computer system, separated keys apply to any situation in which two or more conditions must be met before access should be permitted. For example, systems providing user-extendible protected data types usually depend on separation of privilege for their implementation.

f) Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. Thus, if a question arises related to misuse of a privilege, the number of programs that must be audited is minimized. Put another way, if a mechanism can provide "firewalls," the principle of least privilege provides a rationale for where to install the firewalls. The military security rule of "need-to-know" is an example of this principle.

g) Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users [28]. Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security. Further, any mechanism serving all users must be certified to the satisfaction of every user, a job presumably harder than satisfying only one or a few users. For example, given the choice of implementing a new function as a supervisor procedure shared by all users or as a library procedure that can be handled as though it were the user's own, choose the latter course. Then, if one or a few users are not satisfied with the level of certification of the function, they can provide a substitute or not use it at all. Either way, they can avoid being harmed by a mistake in it.

h) Psychological acceptability: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. Also, to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors.

Analysts of traditional physical security systems have suggested two further design principles which, unfortunately, apply only imperfectly to computer systems.

a) Work factor: Compare the cost of circumventing the mechanism with the resources of a potential attacker. The cost of circumventing, commonly known as the "work factor," in some cases can be easily calculated. For example, the number of experiments needed to try all possible four letter alphabetic passwords is 26^4 = 456 976. If the potential attacker must enter each experimental password at a terminal, one might consider a four-letter password to be adequate. On the other hand, if the attacker could use a large computer capable of trying a million passwords per second, as might be the case where industrial espionage or military security is being considered, a four-letter password would be a minor barrier for a potential intruder. The trouble with the work factor principle is that many computer protection mechanisms are *not* susceptible to direct work factor calculation, since defeating them by systematic attack may be logically impossible. Defeat can be accomplished only by indirect strategies, such as waiting for an accidental hardware failure or searching for an error in implementation. Reliable estimates of the length of such a wait or search are very difficult to make.

b) Compromise recording: It is sometimes suggested that mechanisms that reliably record that a compromise of information has occurred can be used in place of more elaborate mechanisms that completely prevent loss. For example, if a tactical plan is known to have been compromised, it may be possible to construct a different one, rendering the compromised version worthless. An unbreakable padlock on a flimsy file cabinet is an example of such a mechanism. Although the information stored inside may be easy to obtain, the cabinet will inevitably be damaged in the process and the next legitimate user will detect the loss. For another example, many computer systems record the date and time of the most recent use of each file. If this record is tamperproof and reported to the owner, it may help discover unauthorized use. In computer systems, this approach is used rarely, since it is difficult to guarantee discovery once security is broken. Physical damage usually is not involved, and logical damage (and internally stored records of tampering) can be undone by a clever attacker.[10]

As is apparent, these principles do not represent absolute rules--they serve best as warnings. If some part of a design violates a principle, the violation is a symptom of potential trouble, and the design should be carefully reviewed to be sure that the trouble has been accounted for or is unimportant.

4) Summary of Considerations Surrounding Protection: Briefly, then, we may outline our discussion to this point. The application of computers to information handling problems produces a need for a variety of security mechanisms. We are focusing on one aspect, computer protection mechanisms--the mechanisms that control access to information by executing programs. At least four levels of functional goals for a protection system can be identified: all-or-nothing systems, controlled sharing, user-programmed sharing controls, and putting strings on information. But at all levels, the provisions for dynamic changes to authorization for access are a severe complication.

Since no one knows how to build a system without flaws, the alternative is to rely on eight design principles, which tend to reduce both the number and the seriousness of any flaws: Economy of mechanism, fail-safe defaults, complete mediation, open design, separation of privilege, least privilege, least common mechanism, and psychological acceptability.

Finally, some protection designs can be evaluated by comparing the resources of a potential attacker with the work factor required to defeat the system, and compromise recording may be a useful strategy.

B. Technical Underpinnings

1) The Development Plan: At this point we begin a development of the technical basis of information protection in modern computer systems. There are two ways to approach the subject: from the top down, emphasizing the abstract concepts involved, or from the bottom up, identifying insights by, studying example systems. We shall follow the bottom-up approach, introducing a series of models of systems as they are, (or could be) built in real life.

The reader should understand that on this point the authors' judgment differs from that of some of their colleagues. The top-down approach can be very satisfactory when a subject is coherent and self-contained, but for a topic still containing *ad hoc* strategies and competing world views, the bottom-up approach seems safer.

Our first model is of a multiuser system that completely isolates its users from one another. We shall then see how the logically perfect walls of that system can be lowered in a controlled way to allow limited sharing of information between users. Section II of this paper generalizes the mechanics of sharing using two different models: the capability system and the access control list system. It then extends these two models to handle the dynamic situation in which authorizations can change under control of the programs running inside the system. Further extensions to the models control the dynamics. The final model (only superficially explored) is of protected objects and protected subsystems, which allow arbitrary modes of sharing that are unanticipated by the system designer. These models are not intended so much to explain the particular systems as they are to explain the underlying concepts of information protection.

Our emphasis throughout the development is on direct access to information (for example, using LOAD and STORE instructions) rather than acquiring information indirectly (as when calling a data base management system to request the average value of a set of numbers supposedly not directly accessible).