This is the author's original version of the paper. A revised version was published in Communications of the ACM 17, 7 (July 1974) pages 388-402. http://doi.acm.org/10.1145/361011.361067

PROTECTION AND CONTROL OF INFORMATION SHARING IN MULTICS

by

Jerome H. Saltzer

Massachusetts Institute of Technology
Department of Electrical Engineering and Project MAC

ABSTRACT

This paper describes the design of mechanisms to control sharing of Information in the Multics system. Seven design principles help provide insight into the tradeoffs among different possible designs. The key mechanisms described include access control lists, hierarchical control of access specifications, identification and authentication of users, and primary memory protection. The paper ends with a discussion of several known weaknesses in the current protection mechanism design.

An essential part of a general-purpose computer utility system is a set of protection mechanisms which control the transfer of information among the users of the utility. The Multics system*, a prototype computer utility, serves as a useful case study of the protection mechanisms needed to permit controlled sharing of information in an on-line, general-purpose, information-storing system. This paper provides a survey of the various techniques currently used in Multics to provide controlled sharing, user authentication, inter-user isolation, supervisor-user protection, user-written proprietary programs, and control of special privileges.

Controlled sharing of information was a goal in the initial specifications of Multics[8, 11], and thus has influenced every stage of the system design, starting with the hardware modifications to the General Electric 635 computer which produced the original GE 645 base for Multics. As a result, information protection is more thoroughly integrated into the basic design of Multics than is the case for those commercial systems whose original specifications did not include comprehensive consideration of information protection.

Multics is an evolving system, so any case study must be a snapshot taken at some specific time. The time chosen for this snapshot is summer, 1973, at which time Multics is operating at M.I.T. using the Honeywell 6180 computer system. Rather than trying to document every detail of a changing environment, this paper concentrates on the protection strategy of Multics, with the goal of communicating those ideas which can be applied or adapted to other operating systems.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2095 which was monitored by ONR Contract No. NOQ014-70-A-0362-0006.

* A brief description of Multics, and a more complete bibliography, are given in the paper by Corbató, Saltzer, and Clingen[6].

What is new?

In trying to identify the ideas related to protection which were first introduced by Multics, a certain amount of confusion occurs. The design was initially laid out in 1964-1967, and ideas were borrowed from many sources and embellished, and new ideas were added. Since then, the system has been available for study to many other system designers, who have in turn borrowed and embellished upon the ideas they found in Multics while constructing their own systems. Thus some of the ideas reported here have already appeared in the literature. Of the ideas reported here, the following seem to be both novel and previously unreported:

- The notion of designing a comprehensive computer utility with Information protection as a fundamental objective.
- Operation of the supervisor under the same hardware constraints as user programs, under descriptor control and in the same address space as the user.
- Facilities for user-constructed protected subsystems.
- An access control system applicable to batch as well as on-line jobs.
- Extensive human engineering of the user authentication (password) interface.
- Decentralization of administrative control of the protection mechanisms.
- Ability to allow or revoke access with immediate effect.

Multics is unique in the extent to which information protection has been permitted to influence the entire system design. By describing the range of protection ideas embedded in Multics, the extent of this influence should become apparent.

Design Principles

Before proceeding, it is useful to review several design principles which were used in the development of facilities for information protection in Multics. These design principles provided

guidance in many decisions, although admittedly some of the principles were articulated only during the design, rather than in advance.

1. Every designer should know and understand the protection objectives of the system. At the present rather shaky stage of understanding of operating system engineering, there are many points at which an apparently "don't care" decision actually has a bearing on protection. Although these decisions will eventually come to light as the system design is integrated, a system design cannot withstand very many reversals of early design decisions if it is to be completed on a reasonable schedule and within a budget. By keeping all designers aware of the protection objectives, the early decisions are more likely to be made correctly.

2. Keep the design as simple and small as possible. This principle is stated so often that it becomes tiresome to hear. However, it bears repeating with respect to protection mechanisms, since there is a special problem: design and implementation errors which result in unwanted access paths will not be immediately noticed during routine use, since routine use usually does not include attempts to utilize improper access paths. Therefore, techniques such as complete, line-by-line auditing of the protection mechanisms are necessary; for such techniques to be successful, a small and simple design is essential.

3. Protection mechanisms should be based on permission rather than exclusion. This principle means that the default situation is lack of access, and the protection scheme provides selective permission for specific purposes. The alternative, in which mechanisms attempt to screen off sections of an otherwise open system, seems to present the wrong psychological base for secure system design. A conservative design must be based on arguments on why objects should be accessible, rather than on why they should not; in a large system some objects will be inadequately considered and a default of lack of access is more fail-safe. Along the same line of reasoning, a design or implementation mistake in a mechanism which 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 which explicitly excludes access tends to fail by not excluding access, a failure which may go unnoticed.

4. Every access to every object must be checked for authority. This principle, when applied methodically, is the primary underpinning of the protection system. It forces a system-wide view of access control which includes initialization, recovery, shutdown, and maintenance. It also implies that a foolproof method of identifying the source of every request must be devised. In a system designed to operate continuously, this principle requires thai when access decisions are remembered for future use, careful consideration be given to how changes in authority are propagated into such local memories.

5. The design is not secret. The mechanisms do not depend on the ignorance of potential attackers, but rather on possession of specific, more easily protected, protection keys or passwords. This strong decoupling between protection mechanisms and protection keys permits the mechanisms to be reviewed and examined by as many competent authorities as possible, without concern that such review may itself compromise the safeguards. Peters[19] and Baran[2] discuss this point further.

6. The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. If this principle is followed, the effect of accidents is reduced. Also, if a question related to misuse of a privilege occurs, the number of programs which must be audited is minimized. Put another way, if one has a mechanism available which can provide "firewalls", the principle of least privilege provides a rationale for where to install the firewalls.

7. Make sure that the design encourages correct behavior in the users, operators, and administrators of the system. Experience with systems which did not follow this principle revealed numerous examples in which users ignored or bypassed protection mechanisms for the sake of convenience. It is essential that the human interface be designed for naturalness, ease of use, and simplicity, so that users will routinely and automatically apply the protection mechanisms.

The application of these seven design principles will be evident in many of the specific mechanisms described in this paper.

Finally, in the design of Multics there were two additional functional objectives worth dwelling upon. The first of these was to provide the option of complete decentralization of the administration of protection specifications. If the system design forces all administrative decisions (e.g., protection specifications) to be set by a single administrator, that administrator quickly becomes a bottleneck and an impediment to effective use of the system, with the result that users begin adopting habits which bypass the administrator, often compromising protection in the bargain. Even if responsibility can be distributed among several administrators, the same effects may occur. Only by permitting the individual user some control of his own administrative environment can one insist that he take responsibility for his work. Of course, centralization of authority should be available as an option. It is easy to limit decentralization; it seems harder to adapt a centralized design to an environment in which decentralization is needed.

The second additional functional objective was to assume that some users will require protection schemes not anticipated in the original design. This objective requires that the system provide a complete set of handholds so that the user, without exercising special privileges, may construct a protection environment which can interpret access requests however he desires. The method used is to permit any user to construct a protected subsystem, which is a collection of programs and data with the property that the data may be accessed

only by programs in the subsystem, and the programs may be entered only at designated entry points. A protected subsystem can thus be used to program any desired access control scheme.

The Storage System and Access Control Lists

The central fixture of Multics is an organized information storage system.[8] Since the storage system provides both reliability and protection from unauthorized information release, the user is thereby encouraged to make it the repository of all of his programs and data files. All use of information in the storage system is implemented by mapping the information into the virtual memory of some Multics process. Physical storage location is automatically determined by activity. As a result, the storage system is also used for all system data bases and tables, including those related to protection. The consequence of these observations is that one access control mechanism, that of the storage system, handles almost all of the protection responsibility in Multics.

Storage is logically organized in separately named data storage segments, each of which contains up to 262,144 36-bit words. A segment is the cataloguing unit of the storage system, and it is also the unit of separate protection. Associated with each segment is an access control list, an open-ended list of names of users who are permitted to reference the segment*. To understand the structure of the access control list, first consider that every access to a stored segment is actually made by a Multics process. Associated with each process is an unforgeable character string identifier, assigned to the process when it was created. In its simplest form, this identifier might consist of the personal name of the individual responsible for the actions of the process. (This responsible person is commonly called the principal, and the identifier the principal identifier.) Whenever the process attempts to access a segment or other object catalogued by the storage system, the principal identifier of the process is compared with those appearing on the access control list of the object; if any match is found access is granted.

Actually Multics uses a more flexible scheme which facilitates granting access to groups of users, not all of whose members are known, and which may have dynamically varying membership. A principal identifier in Multics consists of several parts; each part of the identifier corresponds to an independent, exhaustive partition of all users into named groups. At present, the standard Multics principal identifier contains three parts, corresponding to three partitions:

1. The first partition places every individual user of the installation in a separate access control group by himself, and names the group with his personal name. (This partition is identical to Lhe simple mechanism of the previous paragraph.)

2. The second partition places users in groups called projects, which are basically sets of users who cooperate in some activity such as constructing a compiler or updating an inventory file. One person may be a member of several projects, although at the beginning of any instance of his use of Multics he must decide under which project he is operating.

3. The third partition allows an individual user to create his own, named protection compartments. Private compartments are chiefly useful for the user who has borrowed a program which he has not audited, and wishes to insure that the borrowed program does not access certain of his own files. The user may designate which of his own partitions he wishes to use at the time he authenticates his identity*.

Although the precise description in terms of exhaustive partitions sounds formidable, in practice a relatively easy-to-use mechanism results. For example, the user named "Jones" working on the project named "Inventory" and designating the personal compartment named "a" would be assigned the principal identifier:

Jones.Inventory.a

Whenever his process attempts to access an object catalogued by the storage system, this three part principal identifier is first compared with successive entries of the access control list for the object. An access control list entry similarly has three parts, but with the additional convention that any or all of the parts may carry a special flag to indicate "don't care" for that particular parLition. (We represent the special flag with an asterisk in the following examples.) Thus, the access control list entry

Jones.Inventory.a

would permit access to exactly the principal of our earlier example. The access control list entry

Jones.*.*

would permit access to Jones no matter what project he is operating under, and independent of his personally designated compartment. Finally, the access control list entry

.Inventory.

would permit access to all users of the "Inventory" project. Matching is on a part by part basis, so there is no confusion if there happens to be a project named "Jones".

Using multi-component principal identifiers it is straightforward to implement a variety of standard security mechanisms. For example, the military "need-to-know" list corresponds to a series or access control list entries with explicit user names but (possibly) asterisks in the remaining fields. The standard government security compartments are examples of additional partitions, and would be implemented by extending the principal identifier to four or more parts, each additional part corresponding to one compartment in use at a particular installation. (Every person would be either in or out of each such compartment.) A restriction of access to users who are simultaneously in two or more compartments is then easily expressed.

---

* The Multics access control list corresponds roughly to a column of Lampson's protection matrix. [16]

* The third partition has not yet been completely implemented. The current system uses the third partition only to distinguish between interactive and absentee use of the system.

We have used the term "object" to describe the entities catalogued by the storage system with the intent of implying that segments are not the only kinds of objects. Currently, four kinds of objects are implemented or envisioned:

1. Segments
2. Message queues (experimental implementation)
3. Directories (called catalogues in some systems)
4. Removable media descriptors (not yet implemented)

For each object, there are several separately controllable modes of access to the object. For example, a segment may be read, written, or executed as a procedure. If we use letters r, w, and e for these three modes of access, an access control list entry fur a segment may specify any of the combinations of access in table I. Certain access mode combinations are prohibited either because they make no sense (e.g. write only) or correct implementation requires more sophisticated machinery than implied by the simple mode settings. (For example, an execute-only mode, while appealing as a method for obtaining proprietary procedures, leaves unsolved certain problems of general proprietary procedures, such as protection of return points of calls to other procedures. The protection ring mechanism described later is used in Multics to implement proprietary procedures. The execute-only mode, while probably useful for less general cases, has not been pursued.)

Mode	Typical use
(none)	access denied
r	read-only data
re	pure procedure
rw	writeable data
rew	impure procedure

Table I: Acceptable combinations of access modes for a segment.

In a similar way, message queues permit separate control of enqueueing and dequeueing of messages, tape reel media descriptors permit separate control of reading, writing, and appending to the end of a tape reel, and directories permit separate control of listing of contents, modifying existing entries, and adding new entries. Control of these various forms of access to objects is provided by extending each access control list entry Co Include access mode indicators. Thus, the access control list entry

```
Smith.*.*   rw
```

permits Smith to read and write the data segment associated with the entry.

It would have been simpler to associate an access mode with the object itself, rather than with each individual access control list entry, but the flexibility of allowing different users to have different access modes seems useful. It also makes possible exceptions to the granting of access to all members of a group. In the case where more than one access control list entry applies, with different access modes, the convention is made that the first access control list entry which matches the principal identifier of the requesting process is the one which applies. Thus, the pair of access control list entries:

```
Smith.Inventory.*   (none)
    *.Inventory.*    rw
```

would deny access to Smith, while permitting all other members of the "Inventory" project to read and write the segment*. To insure that such control is effective, when an entry is added to an access control list, it is sorted into the list according to how specific the entry is by the following rule: all entries containing specific names in the first part are placed before those with "don't cares" in the first part. Each of those subgroups is then similarly ordered according to the second part, and so on. The purpose of this sorting is to allow very specific additions to an access control list to tend to take precedence over previously existing (perhaps by default) less specific entries, without requiring that the user master a language which permits him arbitrary ordering of entries. The result is that most common access control intentions are handled correctly automatically, and only unusually sophisticated intentions require careful analysis by the user to get them to come out right.

To minimize the explicit attention which a user must give to setting access control lists, every directory contains an "initial access control list". Whenever a new object is created in that directory, the contents of the initial access control list are copied into the access control list of the newly created object**. Only if the user wishes access to be handled differently than this does he have to take explicit action. Permission to modify a directory's contents implies also permission to modify its initial access control list.

The access control list mechanism illustrates an interesting subtlety. One might consider providing, as a convenience, checking of new access control list entries at the time they are made, for example to warn a user that he has just created an access control list entry for a non-existent person. Such checks were initially implemented in Multics,

* This feature violates design principle three, which proscribes selective exclusion from an otherwise open environment because of the risk of undetected errors. The feature has been provided nevertheless, because the alternative of listing every user except the few excluded seems clumsy.

** An earlier version of Multics did not copy the initial access control list, but instead considered it to be a common appendix to every access control list in that directory. That strategy made automatic sorting of access control list entries ineffective, so sorting was left to the user. As a result, the net effect of a single change to the common appendix couLd be different for every object in the directory, leading to frequent mistakes and confusion, in violation of the seventh design principle. Since in the protection area, it is essential that a user be able to easily understand the consequences of an action, this apparently more flexible design was abandoned in favor of the less flexible but more understandable one.

but it was quickly noticed that they represented a kind of compromise of privacy: by creating an access control list entry naming an individual, the presence or absence of an error message would tell whether or not that individual was a registered user of the system, thereby possibly compromising his privacy. For this reason, a name-encoding scheme which required checking of access control entry names at the time they were created was abandoned.

It is also interesting to compare the Multics access control scheme with that of the earlier CTSS system[6]. In CTSS, each file had a set of access restriction bits, applying to all users. Sharing of files was accomplished by permitting other users to place in their directories special entries called links, which named the original file, and typically contained further restrictions on allowable access modes. The CTSS scheme had several defects not present in the Multics arrangement:

1. Once a link was in place there was no way to remove it without modifying the borrower's directory. Thus, revocation of access was awkward.

2. A single user, using the same file via different links, could have different access privileges, depending un which link he used. Allowing access rights to depend on the name which happens to be used for an object certainly introduced an extra degree of flexibility, but this flexibility more often resulted in mistakes than in usefulness.

3. As part of a protection audit, one would like to be able to obtain a list of all users who can access a file. To construct that list, on CTSS, one had to search every directory in the system to make a list of links. Thus such an audit was expensive and also compromised other users' privacy.

Multics retains the concept of a link as a naming convenience, but the Multics link confers no access privileges -- it is only an indirect address.

Early in the design of Multic«[8] an additional extension was proposed for an access control list entry: the "trap" extension, consisting of a one-bit flag and the name of a procedure. The idea was that for all users whose principal identifier matched with that entry, if the trap flag were on the procedure named in the trap extension should be called before access be granted. The procedure, supplied by the setter of the access control list entry, could supply arbitrary access constraints, such as permitting access only during certain hours or only after asking another logged in user for an OK. This idea, like that of the execute-only procedure, is appealing but requires an astonishing amount of supporting mechanism. The trap procedure cannot be run in the requesting user's addressing and protection environment, since he is in control of the environment and could easily subvert the trap procedure. Since the trap procedure is supplied by another user, it cannot be run in the supervisor's protection environment, either, so a separate, protected subsystem environment is called for. Since the current Multics protected subsystem scheme allows a subsystem to have access to all of its user's files, implementation of the trap extension could expose a user to unexpected threats from trap procedures on any data segment he touches. Therefore, at the least, a user should be able to request that he be denied access to objects protected by trap extensions, rather than be subject to unexpected threats from trap procedures. Finally, if such a trap occurs on every read or write reference to the segment, the cost would seem to be high. On the other hand, if the trap occurs only at the time the segment is mapped into a user's address space*, then design principle four, that every reference be validated, is violated; revocation of access becomes difficult especially if the system is operated continuously for long periods. The sum total of these considerations led to temporarily abandoning the idea of the trap extension, perhaps until such time as a more general domain scheme, such as that suggested by Schroeder[21] is available.

Both backup copying of segments (for reliability) and bulk input and output to printers, etc. are carried out by operator-controlled processes which are subject to access control just as are ordinary users. Thus a user can insure that printed copies of a segment are not accidentally made, by failing to provide an access control list entry which permits the printer process to read the segment**. Access control list entries permitting backup and bulk I/O are usually part of the default initial access control list. Bulk input of cards is accomplished by an operator process which reads them into a system directory, and leaves a note for the user in question to move them to his own directory. This strategy guarantees that there is no way in which one user can overwrite another user's segment by submitting a spurious card input request. These mechanisms are examples of the fourth design principle: every access to every object is checked for authority.

An administrative consequence of the access control list organization is that personal and project names, once assigned, cannot easily be reused, since the names may appear in access control lists. In principle, a system administrator could, when a user departs, unregister him and then examine every access control list of the storage system for instances of that name, and delete them. The system has been deliberately designed to discourage such a strategy, on the basis that a system administrator should not routinely paw through all the directories of all system users. Thus, the alternative scheme was adopted, requiring all user names, once registered, to be permanent.

Finally, the one most apparent limitation of the scheme as presently implemented is its "one-way" control of access. With the described access control list organization, the owner of a segment has complete control over who may access it. There are some cases in which users other than the owner may wish to see access restricted to an object which the owner has declared public. For example, an instructor of a class may for pedagogical purposes wish to require his students to write a

---

* Or, in traditional file systems, at the time the file is "opened".

** Of course, another user who has permission to read the segment could make a copy and then have the copy printed. Methods of constraining even users who have permission are the subject of continuing research[20].

particular program rather than make use of an equivalent one already publicly available in the system. Alternatively, a project administrator concerned about security may wish to insure that his project members cannot copy sensitive information into storage areas belonging to other users and which are not under his control. He may also want to prevent his project members from setting access control lists to permit access by users outside the project. This kind of control can be expressed in Multics currently only by going to the trouble of constructing a protected subsystem which examines all supervisor calls, thereby permitting complete control over which objects are mapped into the address space and what terms are added to access control lists. Fortunately, there have so far appeared only a few examples in which such control is required, and the escape suggested has proven adequate for those cases. A more general, yet quite simple, solution would be to associate with the user's process two constraining lists: a list of pathnames of directories whose contents he may access, and a list of access control list terms which he is permitted to place on access control lists. These two constraining lists would be set only by the project administrator or security officer. The constraining lists would be especially useful in the military security environment, since they would help in the construction of a list of items a defector might have had access to.

As is evident, the Multics access control list mechanism represents an engineering tradeoff among three conflicting goals: flexibility of expression, ease of understanding and use, and economy of implementation. Additional flexibility of expression was tried (e.g., the common access control list mechanism previously footnoted) with the conclusion that the additional confusion which results from accidental misuse of the generality can outweigh the benefits; apparently the correct direction is the opposite, toward simpler, less general, and more easily understandable protection structures.

Hierarchical Control of Access Specifications

Since in Multics every object, including a directory, must be catalogued in some directory, all objects are arranged into a single hierarchical tree of directories. This naming hierarchy also provides a hierarchy of control of access, through the ability to modify the contents of a directory. Since a directory entry consists of the name of some object and its access control list, having access to modify directory entries is interpreted to include the ability to modify the access control lists of all the objects catalogued in that directory. No further hierarchical control is provided; for example, there is no ability to say "Allow read access to Jones for all segments below this node in the naming tree". Such specifications are similar in nature to the "common access control list" mentioned before; they make it difficult for a user to be sure of all the consequences of a change to the access specification. For example, removing a specification such as that quoted above, which permits only reading, might render effective a forgotten access control term lower in the naming hierarchy which permits both reading and writing*.

Although it would appear that the hierarchical scheme provides an inordinate amount of power to a project administrator and, above him, to a system administrator, in practice it forces a careful consideration of the lines of authority over protected information, and explicit recognition of an authority hierarchy which already existed. In some environments, it would probably be appropriate to publicly log all modifications of directory access above some level, so as to provide a measure of control of the use of hierarchical authority. More elaborate controls might include requiring cooperative consent of some quasi-judicial committee of users for modification of high-level directory access. Such controls are relatively easy for an installation or a project to implement, using protected subsystems.

It is possible, by choosing access modes correctly, to use the hierarchical access control scheme in combination with the initial access control list to accomplish a totally centralized control of all access decisions. If, for example, a project administrator creates a directory for a user, places an initial access control list in that directory, and then grants to the new user permission only to add new entries to the directory, all such new entries would automatically receive a copy of the initial access control list determined by the administrator -- the user would have no control over who may use the objects he creates. By policy, a system administrator could run an entire installation under this tight control, and retain for himself complete authority to determine what access control list is placed on every object, as in IBM'S Resource Security System[14]. Alternatively, any smaller portion of the naming hierarchy can be kept under absolute control by the person having authority to modify access control lists at the top node of the portion.

The other obvious alternative to a hierarchical control of modification of access control lists would be some form of self-control. That is, the ability to modify an access control list would be one of the modes of access controlled by the list itself. A very general version of this alternative has been explored by Rotenberg[20]. This alternative has not been tried out in the Multics context, partly because the implications of the hierarchical method were easier to understand in the first implementation. Probably the chief advantage of self-control of access modification would be that one could provide an individual a fully private work area in which no one -- manager, security officer, or system administrator -- could intrude. On the other hand, the implementation of a "locksmith" while easy to do may require introducing hidden access paths which are then subject to misuse*.

---

* Early versions of Multics provided a limited form of higher-level specification in the form of ability to deny all use of a directory, and therefore of the objects contained within it. For the reasons suggested, this feature has been disabled.

* A locksmith would be an administrator who can provide accountable intervention when mistakes are made. For example, if an organization's key data base is under the exclusive control of a manager who has been disabled in an automobile accident, the locksmith could then provide another manager with access to the file. It seems appropriate to formalize the concept of a locksmith so that appropriate audit trails and authority to be a locksmith

Also, one wonders how a self-control scheme would fit smoothly into an organization which does not usually give an individual the privilege of choosing his own office door lock. Clearly, the social and organizational consequences of the choice between these two design alternatives deserve further study.

Authentication of users

All of the machinery of access control lists, access modes, protected subsystems, and hierarchical control depend on an accurate principal identifier being associated with every process. Accuracy of identification depends on authentication of the user's claimed identity. A variety of mechanisms are used to help insure the security of this authentication. The general strategy chosen by Multics is to maintain individual accountability on a personal basis. Every user uf a given installation (with one class of exception, noted later) is registered at the installation, which means that a unique name, usually his last name plus one or two initials, is permanently entered in a system registry. Associated with his name at the time he is registered is a password of up to eight ASCII characters. Whenever any person proposes to use the system, he supplies his unique name, at which point the system demands also that he provide his password.

Thus far, the authentication mechanism of Multics is essentially the same as for most other remote-accessed systems. However, Multics uses several extra measures related to user authentication, which are not often found in other systems. For one. all use of the system, whether interactive or absentee (batch) is authenticated interactively. That is, initiation of a batch job is not done on the basis of information found in a card reader. Arriving card decks are read in and held in on-line storage by a system process, for which an operator is responsible. All absentee jobs, whether they are to be controlled by files created from cards or files constructed interactively or files constructed by another program, must be initiated by some job already on the system, and whose legitimacy has been previously authenticated. Although a chain of absentee job requests can be developed, the chain must have begun with an interactive job, which requires interactive authentication. In the simplest case, the individual responsible goes to an interactive console, identifies and authenticates himself, and requests execution of the job represented by the incoming card deck. If necessary, the request will automatically wait until the card deck arrives, so that the user need not wait for the operator or for a card reader queue*. Thus, no job is every run without prior positive identification of the responsible party. Note that for installations in which responsibility for card controlled jobs is considered unimportant, it is rather trivial to construct a Multics program, run under the responsibility of the card reader operator, which accepts and runs as a job anything found in the card reader. All such jobs would be run in processes bearing the principal identifier of the card reader operator, and are thus constrained in the range of on-line information which they can access. The inviolate principle of access control remains that on-line authentication of identity, by presenting a password, is required in order to start a process labeled with a particular desired principal identifier. Note also that the fact that a job happens to be operated without an interactive terminal has no bearing on its privileges, except as explicitly controlled by its principal identifier. Finally, to handle the situation where a busy researcher asks a friend to submit the batch job, a proxy login scheme permits the friend to identify himself, under his own password, and then request that the job be run under the principal identifier of the original researcher. The system will permit proxy logins only if the person responsible for the principal identifier to be used has previously authorized such logins by giving a list of proxies*.

As to protection of passwords, several facilities are provided. The user may, after authenticating himself, change his password at any time he feels that the old one may have been compromised. A program is available which will generate a new random eight-character password with English digraph statistics, thereby making it pronounceable and easy to memorize, and minimizing the need for written copies of the password. Users are encouraged to obtain their passwords from this program, rather than choosing passwords themselves, since human-chosen passwords are often surprisingly easy to guess. Passwords are stored in the file system in mildly encrypted form, using a one-way encryption scheme along the lines suggested by Wilkes[29], As a result, passwords are not routinely known by any system administrator or project administrators, and there is never any occasion for which it is even appropriate to print out lists of passwords. If, through some accident, a stored password is exposed, its usefulness is reduced by its encrypted form.

When the user is requested to give his password, at login time, the printer on his terminal is turned off, if possible, or else a background of garbling characters is first printed in the area where he is to type his password. Although the user could be indoctrinated to tear off and destroy the piece of paper containing his password, by routinely protecting it for him the system encourages a concern for security on the part of the user. In addition, if the user's boss (or someone from four levels of management higher) happens to be looking over his shoulder as he logs in, the user is not faced with the awkward social problem of scrambling to conceal his password from a superior who could potentially take offense at an implication that he is not to be trusted with the information.

A time-out is provided to help protect the user who leaves his terminal, is distracted, and forgets to log out. If no activity occurs for a period, a logout is automatically generated. The length of the time-out period can be adjusted to suit the needs of a particular installation. Similarly, whenever service is interrupted by a system failure for more than a moment, a new login

---

can be well-defined. The alternative of sending a system programmer into the computer room with instructions to directly patch the system or its data may leave no audit trail and almost certainly encourages sloppy practice.

* The automatic wait is not yet implemented.

* The proxy login it not yet implemented.

is required of all interact ive users, since some users may have given up and Left their terminals.

Finally, several logging and penetration detection techniques help prevent attacks via the password routine. If a user provides an incorrect password, the event of an incorrect login attempt is noted in a threat-monitoring log, and the user is permitted to try again, up to a limit of ten times at which point the telephone (or network) connection is forcibly broken by the system, introducing delay to frustrate systematic penetration attempts*. Whenever a user logs in, the time and physical location (terminal identification) of his previous login are printed out in his greeting message, thus giving him an opportunity to notice if his password has been used by someone else in his absence. Similarly, monthly accounting reports break down usage by shift and services used, and may be reviewed on-line at any time, thereby providing an opportunity for the individual to compare his pattern of use with that observed by the system, and perhaps to thereby detect unauthorized use. If either of these mechanisms suggests unauthorized use, the individual involved may ask the system administrator to check the system log, which contains an entry for every login and logout giving date and time, terminal type used, and terminal identification, if any.

For a project which maintains especially sensitive information, the project administrator may designate the initial procedure to be executed by some or all processes created using the name of that project as part of its principal identifier. This initial procedure, supplied by the project administrator, has complete control of the process, and can demand further authentication (e.g., a one-time password or a challenge-response scheme,) perform project logging of the result, constrain the user to a subset of the available facilities, or initiate a logout sequence, thereby refusing access to the user. In the other direction, some projects may wish to allow unlimited public access to their files. If so, the project administrator may indicate that his project will accept login of unauthenticated users. In such a case, the system does not demand a password, instead assigning the personal name "anonymous" to the principal identifier of the process involved, using the name of the responsible project for the second part of the principal identifier. The principal identifier "anonymous" is the one exception to the registration scheme mentioned earlier. Allowing anonymous users does not compromise the security of the storage system, since the principal identifier is constrained, and all storage system access is based on the principal identifier. The primary use of anonymous users has been for educational purposes, in which all students in a class are to perform some assignment. Sometimes, this feature is coupled with the project-designated initial procedure, so that the project may implement its own password scheme, or control what facilities are made available, so as to limit its financial liability. Some statistical analysis and data-base development projects also permit anonymous use of data-retrieval programs.

The objective of many of these mechanisms, such as simple registration of every user, the proxy login, the anonymous user, concealment of printed passwords, and user changeable passwords, together with a storage system which permits all authorized sharing of information, is to provide an environment in which there is never any need for anyone to know a password other than his own. Experience with the earlier CTSS system demonstrated that by omitting any of these features, the system itself may encourage borrowing of passwords, with an attendent reduction in overall security.

Primary Memory Protection

We may consider the access control list to be the first level of mechanism providing protection for stored information. Most of the burden of keeping users' programs from interfering with one another, with protected subsystems, and with the supervisor is actually carried by a second level of mechanism, which is descriptor-based. This second level is introduced essentially for speed, so that arbitration of access may occur on every reference to memory. As a result, the second level is implemented mostly in hardware in the central processing unit of the Honeywell 6180. Of course, this strategy requires that the second level of mechanism be operated in such a way as to carry out the intent expressed in the first level access control lists.

As described by Bensoussan et al.[4] the Multics virtual memory is segmented to permit sharing of objects in the virtual memory, and to simplify address space management for the programmer. The implementation of segmentation uses addressing descriptors, a technique used, for example, in the Burroughs B5000 computer systems[9]. The Burroughs implementation of a descriptor is exclusively as an addressing and type-labeling mechanism, with protection provided on the basis that a process may access only those objects for which it has names. In Multics, the function of the descriptor* is extended to include modes of access (read, write, and execute) and to provide for protected subsystems which share object names with their users. Evans and LeClerc[lO] were among the first to describe the usefulness of such an extension.

---

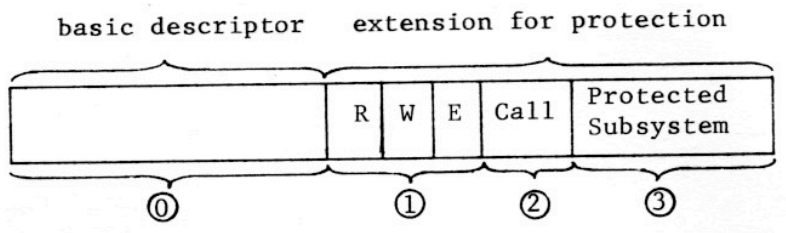
* With ASCII passwords chosen to match English digraph frequency, a little less than four bits of information are represented by each character (despite the eight or nine bits required to store the characters.) An eight character password thus carries about 30 bits of information, which would require about 10^9 guesses using an information theoretic optimum guessing strategy. If one mounted a simultaneous attack from 100 computer-driven terminals, and the system-imposed delays average only 10 milliseconds per attempt, about 10^5 seconds, or one full day of systematic attack would be required to guess a password. Although use of a uniformly random password generator would increase this work factor by several orders of magnitude, resistance to use of hard-to-remember passwords and the need to make written copies might act to wipe out the gain. Of course, this work factor calculation presumes that the attacker has no further basis on which to narrow the range of password possibilities, for example, by knowing that the user in question may have chosen his own password or by wiretapping a previous login.

* With the exception of type identification, which is not provided in Multics.

As shown in figure one, there are three classes of descriptor extensions for protection purposes: mode control, protected subsystem entry control, and control on which protected subsystems may use the descriptor at all. Every reference of the processor to the segment described by this descriptor is thus checked for validity.

The virtual address space of a Multics process is implemented with an array of descriptors, called a descriptor segment, as in figure two. Every reference LO the virtual memory specifies both a segment number (which is interpreted as an index into the descriptor segment) and a word number within the segment.

Figure two also helps illustrate why the protection information is associated with the addressing descriptor rather than with the data itself*. Each computation is carried out in its own address space, so each computation has its own private descriptor segment. Using this mechanism, a single physical segment may appear in different address spaces with different access privileges for different users, even though they are referring to the same physical data. Since in a multiprocessor system such as Multics two such processes may be executing simultaneously, a single protection specification associated with the data is not



⓪ Physical address and size of the segment based on this descriptor.

① Bits separately controlling permission to read, write, and execute the contents of the segment based on this descriptor.

② Control of permission to enter a protected subsystem which has entry points in the segment based on this descriptor.

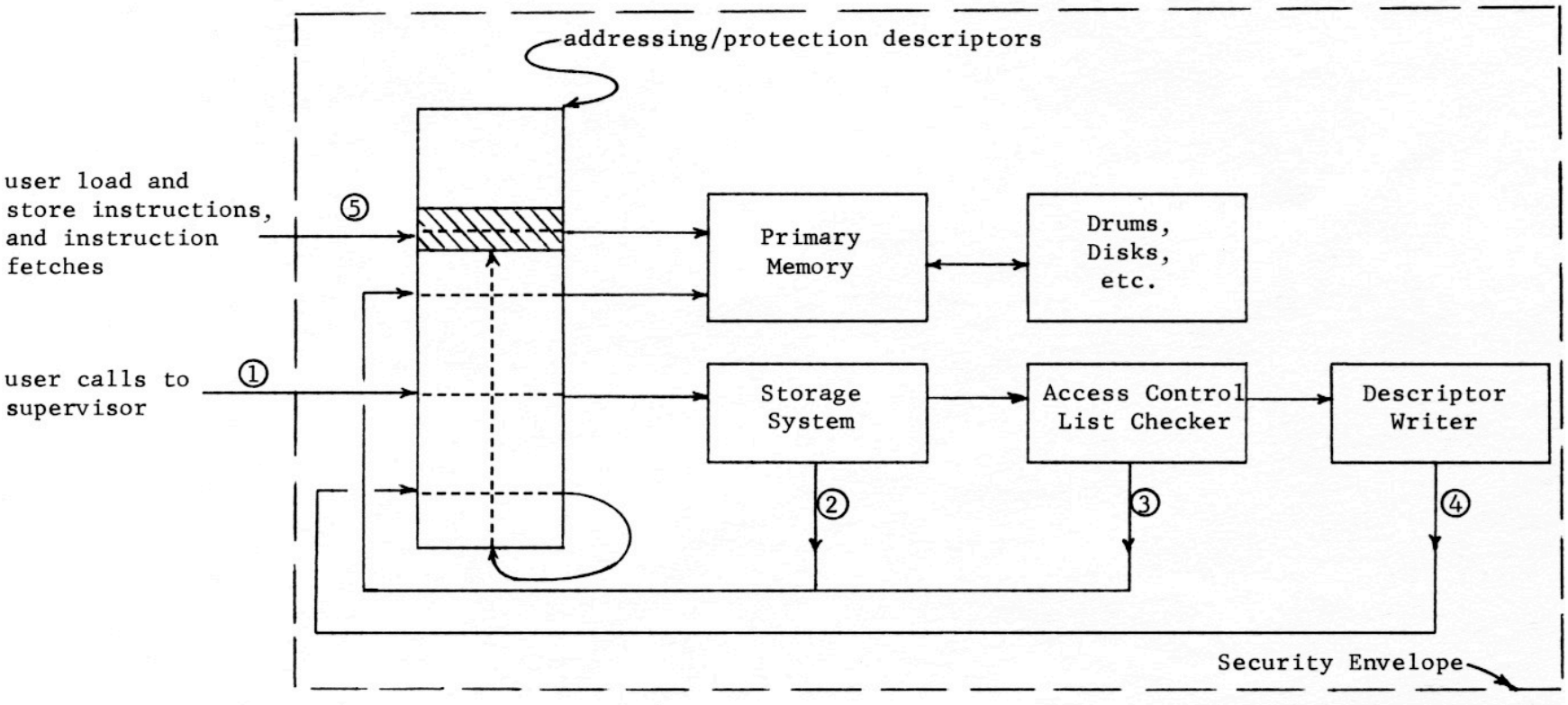
③ Controls on which (hierarchically arranged) protected subsystems may use this descriptor.

Figure 1 -- A Multics descriptor.

sufficient. Having the protection specification associated with the descriptor allows for such controlled sharing to be handled easily.

An unusual feature of the descriptors used in Multics is embodied in the second and third extensions of figure one. Together, they allow hardware enforcement of protected subsystems. A protected subsystem is a collection of procedures and data bases which are intended to be used only by calls to designated entry points, known in Multics

* The alternate option is chosen, for example, in the IBM 360/67 and the IBM 370 "Advanced Function" virtual memory systems[24].



① Call to storage system to add object to virtual memory.

② VM access by storage system to locate object in directory structure. (Includes recursive invocation of storage system to add directories to VM).

③ VM access by access control list checker to read principal identifier and access control list.

④ VM access to write new addressing and protection descriptor into descriptor segment.

⑤ Caller accesses new object.

Figure 2 -- Descriptor management in Multics. The Multics supervisor is treated as a protected subsystem.

as gates. If this intention is hardware enforced, it is possible to construct proprietary programs which cannot be read, data base managers which return only statistics rather than raw data to some callers, and debugging tools which cannot be accidentally disabled. The descriptor extensions are used to authenticate subroutine calls to protected subsystems. Two important advantages flow from using a hardware checked call:

1. Calls to protected subsystems use the same structural mechanisms as do calls to unprotected subroutines, with the same cost in execution time. Thus a programmer does not need to take the fact that he is calling a protected subsystem into account when he tries to estimate the performance of a new program design.

2. It is quite easy to extend to the user the ability to write protected subsystems of his own. Without any special privileges, any user may develop his own proprietary program, data-screening system, or extra authentication system, and be assured that even though he permits others to use his protected subsystem. the information he is protecting receives the same kind of security as does the supervisor itself.

In support of call protection, hardware is also provided to automatically check the addresses of all arguments as they are used, to be sure that the caller has access to them. Checking the range of the argument values is left to the protected subsystem.

Protected subsystems are formed by using the third field of the descriptor extension of figure one. To simplify protected subsystem implementation, Multics imposes a hierarchical constraint on all subsystems which operate within a single process: each subsystem is assigned a number, between 0 and 7, and it is permitted to use all of those descriptors containing protected subsystem numbers greater than or equal to its own. Among the descriptors available to a subsystem may be some permitting it to call to the entry points of other protected subsystems. This scheme goes by the name rings of protection, and is more completely described by Graham[12] and by Schroeder and Saltzer[22].* As far as is known, the only previously existing systems to permit general, user-constructed protected subsystems are the M.I.T. PDP-1 time-sharing system[1] and the CAL operating system[15].

The descriptor-based strategy permits two further simplifying steps to be taken:

1. All information in the storage system is read and written by mapping it into the virtual memory, and then using load and store instructions whose validity is checked by the descriptor mechanism.

2. The supervisor itself is treated as an example of a protected subsystem, which operates in a virtual memory arbitrated by descriptors, exactly the same as do the user programs which it supports.

The reasons why the first step provides simplification for the user have been discussed extensively in the literature[4,13]. The second step deserves some more comment. By placing the supervisor itself under the control of the descriptors, as in figure two, a rather substantial benefit is achieved: the supervisor then operates with the same addressing and machine language code generation environment as the user, which means that supervisor programs may be constructed using the same compilers and debugging tools available to a user. The effect on protection is non-trivial: programs constructed and checked out with more powerful tools tend to have fewer errors, and errors in the supervisor which compromise protection often escape notice.

Perhaps equally important is that the determination of whether one is in or out of the supervisor is not based on some processor mode bit which can be accidentally left in the wrong state when control is passed to a user program. Instead, the addressing privileges of the current protected subsystem are governed by the subsystem identification, located in the descriptor of the segment which supplied the most recent instruction. Every transfer of control to a different program is thus guaranteed to automatically produce addressing privileges appropriate to the new program. If a supervisor procedure should accidentally transfer to a location in a user procedure, that procedure will find that the protection environment has automatically returned to the state appropriate for running user procedures.

Finally, the descriptors are adjusted to provide only the amount of access required by the supervisor, in consonance with design principle six. For example, procedures are not writeable, and data bases are not executable. As a result, programming errors related to using incorrect addresses tend to be immediately detected as protection violations, and do not persist into delivered systems. If one reviews the operation of Multics starting with the initial loading of the system on an empty machine, he will find that only the first hundred or so instructions do not use descriptors. Once a descriptor segment has been fashioned, all memory references by the processor from that point on are arbitrated by descriptors.

These mechanisms do not prohibit the supervisor from making full use of the hardware when appropriate. Rather, they protect against accidental overuse of supervisor privileges. Clearly, the supervisor must be able to write into the descriptor segment, in order to initially set it up, and also to honor requests to map additional objects of the storage system into segments of the virtual memory. This adjustment of descriptors is done with great care, using a single procedure whose only function is to construct descriptors which correspond to access control list entries. A call to the storage system which results in adjustment of a descriptor is illustrated in figure two. In this figure, it is worth noting that even the writing of the descriptor is done with use of a descriptor for the descriptor itself. Thus there is little danger of accidentally modifying a descriptor segment belonging to some other user,

---

* A more general approach, not yet implemented, but which removes the restriction that the protected subsystem be hierarchical, is described by Schroeder in his doctoral thesis[21).

since the only descriptor segment routinely appearing in the virtual memory of this process is its own.

Entries to the supervisor which implement "special privileges" (e.g., the operator may have the privilege of shutting the system down) are generally controlled by ordinary access control lists, either on the gates of supervisor entries, or in some cases by having the supervisor procedure access some data segment before proceeding with the privileged operation. If the user attempting to invoke the privilege does not appear on the access control list of the data segment, an access violation fault will occur, rather than an unauthorized use of the privilege.

The final step of "locking up" the supervisor lies in management of source-sink input-output. Recall first that all access to on-line catalogued information of the storage system is handled by direct mapping into the virtual memory. Thus, input and output operations in Muitics consist only of true source-sink operations, that is of streams of information which enter or leave the system. Such operations are performed by hardware I/O channels, following channel programs constructed by the I/O system in response to I/O requests of the calling program. These I/O channel programs are placed in a part of the virtual memory accessible only to the supervisor*. Similarly, all input data is read into a protected buffer area, accessible only to the supervisor. Only after the input has arrived and the supervisor has had a chance to check it is it turned over to the user, either by copying it, or by modifying a descriptor to make it accessible to the user. A similar, inverse pattern is used on output. Since during I/O neither the data nor the channel program is accessible to the user, there is no hesitation about permitting him to continue his computation in parallel with the I/O operation. Thus, fully asynchronous operations are possible.

The system is initialized from a magnetic tape which contains copies of every program residing in the most protected area. In this way, the integrity of the protection mechanisms depends on protecting only one magnetic tape, and is independent of the contents of the secondary storage system (disk and drums) which are more exposed to compromise by maintenance staff. On the other hand, since the system is designed for continuous operation, there appears to be no need for a separate package consisting of passwords and clearance information as suggested by Weissman[28].

To round out the discussion of primary and virtual memory protection, we should consider storage residues. A storage residue is the data copy left in a physical storage device after the previous user has finished with it. Storage residues must be carefully controlled to avoid accidental release of information. In a virtual memory system, the only way a storage residue could be examined would be to read from a previously unused part of the virtual memory. By convention, in Multics, the supervisor provides pages of zeros in response to such attempts. Since all access to on-line storage is via the virtual memory, no additional mechanism is required to insure that a user never sees a residue from the storage system.

Weaknesses of the Multics protection Mechanisms

One is always hesitant to list the weaknesses in his system, for a variety of reasons. Often, they represent mistakes or errors of judgement, which are embarrassing to admit. Such a list provides an easy target for detractors of a design, and in the protection area provides an invitation for potential attackers at production installations which happen to be using the system. In the case of a system still evolving, such as Multics, known weaknesses are being corrected as rapidly as feasible, so any list of weaknesses is rapidly obsolete. And finally, any list of weaknesses is almost certainly incomplete, being subject to all of the built-in blindnesses of its authors. Nevertheless, such a list is quite useful, both to look for specific interesting unsolved problems, and also to establish what level of considerations are still considered relevant by the designers of the system. The weaknesses described here begin with two major areas, followed by several smaller problems.

Probably the most important weakness in the current Multics design lies in the large number of different program modules which have the ability, in principle, to compromise the protection system. Of the 2000 program modules which comprise Multics, some 400, or 20%, are in the "most protected area", consisting of system initialization, the storage system, miscellaneous supervisor functions, and system shutdown. Although all of these 400 modules operate using the descriptor-based virtual memory described earlier, the descriptors serve for them only as protection against accidentally generated illegal address references; these modules are not constrained by the inability to construct suitable descriptors in the same way as the remaining 1600 modules and user programs. Thus any of these 400 modules (averaging perhaps 200 lines of source code each) might contain an error which compromises the security mechanisms, or even a security violation intentionally inserted by a system programmer. The large number of programs and the very high internal intricacy level frustrates line by line auditing for errors, misimplementation, or intentially planted trapdoors. This weakness is not surprising for the first implementation of a sophisticated system, and upon review it is now apparent that with mild software restructuring plus help from specialized hardware the number of lines of code in the most protected area can be greatly reduced --

---

* And to the I/O channels, which use absolute addresses. If separate I/O channels were available to each physical device and the I/O channels used the addressing descriptors, protected supervisor procedures would not be required for I/O operations after device assignment (which requires a descriptor to be constructed.)

Here is an example of a place where building a new system, rather than modifying an old one, has simplified matters. On some computer systems, the user constructs his own channel programs, and may even expect to modify them dynamically during channel operation. It is quite hard to invent a satisfactory scheme for protecting other users against such I/O operations without placing restrictions on their scope, or inhibiting parallel operation of the user with his I/O channel programs.

perhaps by as much as an order of magnitude. In examining many specific examples, there seem to have been three common, interrelated reasons for the extra bulk currently found in the protected area:

- economics: at the time of design, a function could be implemented more cheaply in the most protected region. Since the protection ring mechanism was originally simulated by software, there were design decisions based on the assumption that calls across ring boundaries were expensive.
- rush to get on the air: in the hurry to get an initial version of the system going, a shortcut was found, which required unnecessarily placing a module in the most protected region.
- lack of understanding: a complex subsystem was not carefully enough analyzed to separate the parts requiring protection; the entire subsystem was therefore protected.

With hardware-supported protection rings, hindsight, and the experience of a complete working implementation, it is apparent that a smaller "most protected area" can be constructed. It now appears possible to make complete auditing a feasible task. A project is now underway to test this hypothesis by attempting to develop an auditable version of the most protected region of Multics.

The second serious weakness in the current Multics design is in the complexity of the user interface. In creating a new segment, a user should specify permitted lists of users and projects, specify allowed modes of access for each, decide whether or not backup copies should be allowed and whether or not bulk I/O should be permitted for the segment, and whether or not the segment should be part of a protected subsystem. He should check that permissions he has given to modify higher-level directories interact in the desired way with his current intent. A variety of defaults have been devised to reduce the number of explicit choices which need be made in common cases: as already mentioned, a per-directory "initial access control list" is by default assigned to any new segment created in that directory. The defaults merely hide the complex underlying structure, however, and do not help the user with an unusual protection requirement, who must figure out for himself how to accomplish his intentions amid a myriad of possiblities, not all of which he understands. The situation for a project administrator, who can control the initial program his users get, and may perhaps force all of his users to interact via a limited, protected subsystem is similar, but with fewer defaults and more possibilities available.

The solution to this problem lies in better understanding the nature of the typical user's mental description of protection intent, and then devising interfaces which permit more direct specification of that protection intent. As an example, a graduate student devised a simple Multics program which prints a list of all users which may force access to a segment (by virtue of having modify access to some higher level directory.) This list does not correspond to any single access control list found anywhere in the system, yet it is clearly relevant to one's image of how the segment is protected. Setting up the mechanisms of access control lists, accessibility modes, and rings of protection perhaps should be viewed as a problem of programming in which, as usual, the structures available in initial designs do not correspond directly with the user's way of thinking, even though there may be some way of programming the structure to accomplish any intent. In the area of protection, the problem has a special edge, since if a user, through confusion, devises an overly permissive protection specification, he may not discover his mistake until too late.

At a level of significance well below the two major points of system size and user interface complexity are several other kinds of problems. These problems are felt to be less significant not because they cannot be exploited as easily, but rather because the changes required to strengthen these areas are straightforward and relatively easy to implement. These problems include:

1. Communication links are weak. Of course, any use of switched telephone lines leads to vulnerability, but provision for integration of a Lucifer-like system[23] for end-to-end encryption of messages sent over public lines or through a communication network would probably be a desirable (and simple) addition. As an example of a typical problem in this area, the Bell System 202C6 DATAPHONE dataset, which is used for 1200 bps terminals, does not include provision for reporting telephone line disconnection to the computer system during data output transmission. If a user accidentally hangs up his telephone line during output, another user dialing to the same port on the computer may receive the output, and capture control of the process. Although remedial measures such as requiring reauthentication every few minutes could be used, automatic detection of the line disconnection would be far more reassuring. (Note that for the more commonly used 103A DATAPHONE dataset, which does report telephone line disconnections, this problem does not exist; upon observing the dropping of the carrier detect line from the dataset, Multics immediately logs the user out.)

2. The operator interface is weak. The primary interface of the operator is as a logged-in user, where his interactions can be logged, verified, and suitably restricted. However, he has a secondary interface: the switches and lights of the hardware itself. It would appear that the potential for error or sabotage via this route is far higher than necessary. If every hardware switch in the system were both readable and settable by (protected supervisor) programs, then all such switches could be declared off limits to the operator, and perhaps placed behind locked panels. Since all operator interaction would then be forced to take place via his terminal, his requests can be checked for plausibility by a program. What has really gone wrong here is a failure to completely reconsider the role of the operator in a computer system operating as a utility. Functions such as operation of card readers and printers do not require access to switches on the side of the processor -- or even physical presence in the same room as the computer, for that matter. The decision that a system failure has occurred and the

appropriate level of recovery action to Cake are probably the operator functions which are hardest to automate or decouple from the physical machine room, but certainly much movement in this direction would be easy to accomplish.

3. Users are permitted to specify their own passwords, leading to easy-to-guess passwords. The resulting loss of security has already been well documented in the literature[25], and this method has been used at least once to improperly obtain access to Multics at M.I.T., when a programmer chose as his Multics password the same password he used on another, unsecured time-sharing system. A better strategy here would be to force the use of system-generated randomly chosen passwords, and also to place an expiration date on them, to force periodic password changes. For sensitive applications, or situations where the password must be exposed to unknown observers (as in using a system via the ARPA network), the system should provide lists of one-time passwords.

4. The supervisor interface is vulnerable to misimplementation. Although this difficulty could be described as a specific example of a supervisor too large and complex to audit, it is worth identifying in its own right. The problem has to do with checking the range of arguments passed to the supervisor. The hardware automatically checks that argument addresses are legitimately accessible to the caller, and completely checks all use of pointer variables as indirect addresses. However, it provides no help in determining whether the ultimate argument values are "reasonable" for the supervisor entry in question. Each entry must be prepared to operate correctly (or at least safely) no matter what combination of argument values is supplied by the caller. Certain kinds of interfaces make for difficulty in auditing a program to see if it properly checks range of arguments. For example, if the allowed range of one argument depends on the result of computation which is based in part on another argument, then it may be hard to enforce a programming standard which requires that all supervisor entries check the range of all their arguments before performing any other computation. The current Multics interface has examples of situations in which, to verify that a supervisor entry is correctly programmed so that it does not blow up when presented with an illegal argument, one must trace hundreds of lines of code and many subroutine calls. Such interfaces discourage routine auditing of the supervisor interface, and probably result in some undetected implementation errors. It would be interesting to explore the design of argument range-checking hardware, which would force the system programmer to declare the allowed range of arguments for his entries, and thereby force out into the open the existence of arguments whose range is not trivially testable, for interface design revision.

5. Secondary storage residues are not cleared until they are reassigned. When a segment is deleted, all descriptors for the physical storage area are destroyed, and the area is marked as reusable. No further descriptors for the storage area will ever be constructed without first clearing the storage area, but meanwhile the residue remains intact. In principle, there is no way to exploit these residues using the system itself, but automatic overwriting of the residues at the time of deletion would provide an additional safeguard against accidents, and guarantee that a segment, once deleted, is not accessible even to a hardware maintenance engineer. A similar problem exists for the magnetic tapes containing backup copies of segments. In at least one case on another time-sharing system, the persistence of backup copies has proved embarrassing: a government agency requested that a file containing a list of special telephone access codes be completely deleted; the installation administrator found himself with no convenient way to purge the residues on the backup tapes. These tapes should probably be encrypted, using per-segment keys known only by the operating system. It is an interesting problem to construct a strategy for safely encrypting backup copy tapes, while ensuring that encrypting keys do not get destroyed upon system failure, making the backup copies worthless.

6. Over-privileged system administrator. Some system functions have been organized in such a way that the administrators of the system require more privilege than really necessary. For example, measures of secondary storage usage are stored in the using directory rather than in an account file. As a result, the administrative accounting programs which prepare bills for secondary storage use must have access to read every directory in the storage system. For another example, the "locksmith" function, mentioned earlier, is currently implemented by giving the locksmith permission to modify the root directory of the storage system directory hierarchy. Thus the locksmith has the unaudited ability to grant himself access to every file in the storage system. Such a design means that one of the easiest ways to attack is to attempt to influence the system administrator, possibly by surreptitiously inserting traps in some program he is likely to use* while running a process whose principal identifier needlessly permits extensive privileges. The counter measure, currently partially implemented, is to provide administrators with protected subsystems from which they cannot escape, which are certified to exercise a minimum of privilege, and which maintain audit trails.

7. Ponderous backup copy and retrieval scheme. It has been noticed that the general method currently used for indexing the contents of storage system backup copy tapes is weak, so that the only effective way to identify a desired copy of a damaged segment is to permit the user to manually scan printed journals of the names of the segments copied onto each tape. These journals contain the names of

---

* This technique has been described as the "Trojan Horse" attack[5].

other users' segments and directories, and were intended for use only for emergency situations and with proper clearance. Unfortunately, the number of retrieval requests which can be handled on other than an emergency basis is a sensitive function of the quality of the tools available for searching the journals automatically while maintaining privacy. A simple scheme based on a protected subsystem for searching journals has recently been proposed, but is not yet implemented.

8. Counter-intelligence techniques have not been exploited. Although logs of suspicious events (such as incorrectly supplied passwords) are maintained no true counter-intelligence strategies are employed. For example Turn, et al. [26] have suggested inserting carefully monitored apparent flaws in the system. These flaws would be intended to attract a would-be attacker, any attempt to exploit them would result in an early warning of attack and an opportunity to apprehend the attacker.

9. Some areas of potentially vulnerability have not been examined. These include vulnerability to undetected failures of the hardware protection apparatus[17],* electromagnetic radiation from the physical hardware machine[3], and traffic analysis possibilities, using performance measurement tools available to any user.

It is interesting to note that none of these nine specific weaknesses represent intrinsic difficulties of full-scale computer utility systems -- relatively straightforward modification can easily strengthen any of these areas. In fact, neither the two major weaknesses nor the nine specific ones represent "holes" in the sense of being immediately exploitable by an attacker. Rather, they are areas in which an attacker is more likely to discover a method of entry caused by misimplementation, misunderstanding, or mismanagement of an otherwise securable system. Thus we might describe the protection system as usable, though with known areas of weakness.

Conclusions

This paper has surveyed the complete range of information protection techniques which have been applied to a specific example of a system designed for production use as a computer utility. Over three years of experience in a production environment at M.I.T. has demonstrated that the mechanisms are generally useful. A commonly asked question (especially in the light of recent experiences with attempts to add security to other commercially available computer systems) is "how much performance is lost?" This question is difficult to answer since, as is evident, the protection structure is deeply integrated into the system and cannot by simply "turned off" for an experiment.* However, one significant observation may be made. in general, the protection mechanisms are closely related to naming mechanisms, and can be implemented with a minimum of extra fuss in a system which provides a highly structured naming environment. Thus, the users of Multics apparently have found that the overall package of a structured virtual memory with protection comes at an acceptable price.

The Multics protection mechanisms were designed to be basic and extendable, rather than a complete implementation of some specialized security model, Thus there are mechanisms which may be used to provide the multilevel security classification (top secret, secret, confidential, unclassified) and the access compartments of the U.S. governmental security system[27]. If one wished to precisely imitate the government security system, he could do so without altering the operating system. In this sense, Multics differs with, say, SDC's ADEPT[28] and IBM'S Resource Security System[14], both of which specifically implement models of the government security system, but which do not permit, for example, user-written program-protected data bases.

We should also note that the Multics system was designed to be securable, which is different than stating that any particular site is actually operated in a completely secured fashion. Such matters as machine room security, certification of of hardware maintenance engineers and system operators, and telephone wire tapping are largely outside of the scope of operating system design. In addition, correct administration can be encouraged by the design of an operating system, but not enforced. Further we have reported the design of the system, realizing that its implementation has not yet been completely audited and therefore may contain trivial programming errors which affect protection.

Acknowledgements

As is usual in any large system design, many individuals have contributed ideas and suggestions, and a complete acknowledgement is very hard to compose. Professor E.L. Glaser provided the firm conviction that information protection was a reasonable goal during the critical initial design period of the Multics system. He also suggested several of the design principles and many of the specific protection mechanisms which were ultimately included. Professor R.M. Graham worked out the initial design of the protection ring mechanism, and Professor M.D. Schroeder expanded that design to include automatic argument validation and complete hardware support. Integration of protection into the storage system was accomplished by R.C. Daley. More recent upgradings of the user interface have been designed by V.L. Voydock, R.J. Feiertag, and T.H. VanVleck. P.A. Belmont,

---

* Although the 6180 hardware is less vulnerable than some. An asynchronous processor-memory interface tends to stop when an error occurs rather than proceeding with wrong data; complete instruction decoding explicitly traps all but legal operation codes and addressing modifiers; and the multiprocessor organization helps obviate the need for pipelines and other accident-prone highly-tuned logic tricks.

* In analogy, we may consider a mouse. The mouse has an elaborate system which maintains a constant body temperature, where, for example, a lizard does not. There is a sense in which the mouse is thereby less efficient, but one may also credibly argue that the question of efficiency is incorrectly posed. In a similar way, comparison of systems with and without protection may also be incorrect. (Analogy thanks to Caria M. Vogt.)

D.A. Stone, and M.A. Meer developed an early internal memorandum which helped articulate the design issues. Others offering significant help include Professor F.J. Corbató, C.T. Clingen, D.D. Clark, M.A. Padlipsky, and P.G. Neumann. Of course, every system programmer who worked in the most protected region of Multics has also contributed by his extra care and understanding of the protection objective.

References

1. Ackerman, W.B., and W.W. Plummer, "An Implementation of a Multiprocessing Computer System." ACM Symposium on Operating Systems Principles, October, 1967, Gatlinburg, Tennessee.

2. Baran, P., "Security, Secrecy, and Tamper-Free Considerations," On Distributed Communications 9, Rand Corp. Technical Report RM-3765-PR.

3. Beardsley, C.W., "Is your computer insecure';" IEEE Spectrum 9, 1 (January, 1972), pp. 67-78.

4. Bensoussan, A., C.T. Clingen, and R.C. Daley, "The Multics Virtual Memory: Concepts and Design," Comm. ACM 15, 5 (May, 1972), pp. 308-318.

5. Branstad, D.K., "Privacy and Protection in Operating Systems," Computer 6, 1, 1973, pp. 43-47.

6. The Compatible Time-Sharing System: A Progrsmmer's Guide, M.I.T. Press, 1966.

7. Corbató, F.J., J.H. Saltzer, and C.T. Clingen, "Multics: The First Seven Years," AFIPS Conf. Proc. 40, (1972 SJCC), pp. 571-583.

8. Daley, R.C., and P.G. Neumann, "A General-Purpose File System for Secondary Storage," AFIPS Conf. Proc. 27, (1965 FJCC), pp. 213-229.

9. The Descriptor -- A Definition of the B5000 Information Processing System. Burroughs Corporation, Business Machines Group, Sales Technical Services, Systems Documentation, Detroit, Michigan, 1961.

10. Evans, D.C., and J.Y. LeClerc, "Address Mapping and the Control of Access in an Interactive Computer," AFIPS Conf. Proc. 30, (1967 SJCC), pp. 23-30.

11. Glaser, E.L., "A Brief Description of Privacy Measures in the Multics Operating System," AFIPS Conf. Proc. 30, (1967 SJCC), pp. 303-304.

12. Graham, P.M., "Protection in an Information Processing Utility," Comm. ACM 11, 5 (May, 1968), pp. 365-369.

13. Holland, S.A., and C.J. Purcell, "The CDC Star-100 -- A Large Scale Network Oriented Computer System," IEEE International Computer Society Conf., (September, 1971), pp. 55-56.

14. IBM Application Program Manual "OS/MVT with Resource Security. General Information and Planning Manual," File no. GH20-1058-0, IBM Corporation, December, 1971.

15. Lampson, B.W., "An Overview of the CAL Time-sharing System," Computer Center, University of California, Berkeley, (September 5, 1969).

16. Lampson, B.W., "Protection," Proc. 5th Princeton Conf. on Information Sciences and Systems, (March, 1971), pp. 437-443.

17. Molho, L.M., "Hardware aspects of secure computing," AFIPS Conf. Proc. 36, (1970 SJCC) pp. 135-141.

18. Needham, P.M., "Protection Systems and Protection Implementations." AFIPS Conf. Proc. 41, Vol I. (1972 FJCC), pp. 572-578.

19. Peters, B., "Security considerations in a multi-programmed computer system," AFIPS Conf. Proc. 30, (1967 SJCC), pp. 283-286.

20. Rotenberg, L;, "Making Computers Keep Secrets," Ph.D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, September, 1973. (Also available as M.I.T. Project MAC Technical Report TR-116.)

21. Schroeder, M.D., "Cooperation of Mutually Suspicious Subsystems in a Computer Utility," Ph.D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, September, 1972. (Also available as M.I.T. Project MAC Technical Report TR-104.)

22. Schroeder, M.D., and J.H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Comm. ACM 15, 3 (March, 1972), pp. 157-170.

23. Smith, J.L., W.A. Notz, and P.R. Osseck, "An Experimental Application of Cryptography to a Remotely Accessed Data System," Proc. ACM 1972 Conf., pp. 282-297.

24. System 370 Principles of Operation. IBM Systems Reference Library File no. GA22-7000.

25. "Third Party ID Aided Program Theft," ComputerWorld V, 14, April 7, 1971.

26. Turn, R., R. Fredrickson. and D. Hollingworth, Data Security at the Rand Corporation, Rand Corp. Technical Report P-4914, October, 1972.

27. Ware, W.. et al., "Security Controls for Computer Systems, Rand Corp. Technical Report R-609, 1970. (Classified Confidential).

28. Weissman, C., "Security Controls in the ADEPT-50 Time-Sharing System," AFIPS Conf. Proc. 35, (1969 FJCC), pp. 119-133.

29. Wilkes, M.V., Time-Sharing Computer Systems. American Elsevier Publishing Co., 1968.