Protection and the Control of Information Sharing in Multics

Jerome H. Saltzer
Massachusetts Institute of Technology

The design of mechanisms to control the sharing of information in the Multics system is described. Five 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.

Key Words and Phrases: Multics, protection, security, privacy, access control, authentication, computer utilities, time-sharing systems, proprietary programs, protected subsystems, virtual memory, descriptors

CR Categories: 3.70, 4.30, 6.2

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15–17, 1973.

Work reported herein was conducted by the Computer Systems Research Division of Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract number N00014-70-A-0362-0006. Author's address: Computer Systems Research Division, Massachusetts Institute of Technology, Project MAC, 545 Technology Square, Cambridge, MA 02139.

An essential part of a general-purpose computer utility is a set of protection mechanisms which control transfer of information among users of the utility. The Multics system,[1] a prototype computer utility, serves as a case study of protection mechanisms which can permit controlled sharing of information in an on-line, general-purpose, information-storing system. This paper provides a survey of the 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, 12], and has influenced every stage of system design, starting with 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 was 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.

In trying to identify the ideas related to protection which were 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 the ideas they found in Multics while constructing their own systems. Thus some of the ideas reported here have already appeared in the literature, and earlier versions of some ideas have been scattered in previous papers and books about Multics. However, Multics is unique in the extent to which information protection has been permitted to influence the entire system design. By describing in one place the range of protection ideas embedded in Multics, and their current design status, the extent of this influence should become apparent.

[1] A brief description of Multics and a more complete bibliography of Multics publications are given in the paper by Corbató, Saltzer, and Clingen [7].

Design Principles

One of the lessons learned during the development of Multics was the importance of formulating design principles and of carefully communicating these design principles to every project member. Although they were articulated only during the project rather than in advance, the following five principles, especially applicable to protection, are worthy of mention.

1. Base the protection mechanisms on permission rather than exclusion. This principle 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, seems to present a 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 permission is more fail-safe. Similarly, 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. 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.
2. Check every access to every object for current authority. In a system designed to operate continuously, this principle requires that, if access decisions are remembered for future use, careful consideration be given to how changes in authority are propagated into such local memories.
3. The design is not secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on possession of specific, more easily protected, protection keys or passwords. This strong decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers, without concern that such review itself may compromise the safeguards. This principle is not new—Peters [24] and Baran [2] discuss it in depth—but its violation sent a surprising number of design proposals back to the drawing boards.
4. 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. The purpose of this principle is to reduce the number of potential interactions among privileged programs to the minimum necessary to operate correctly, so that one may develop confidence that unintentional, unwanted, or improper uses of privilege do not occur. 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. The military security rule of "need-to-know" is an example of this principle.
5. 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.

In the design of Multics there were two specific functional objectives worth mention. The first of these was to provide for decentralization of the setting of protection specifications. If a system design forces too many administrative decisions (e.g. protection specifications) to be set by a single administrator, that administrator can quickly become a bottleneck and an impediment to effective use of the system, with the result that users begin adopting habits which bypass the administrator, perhaps compromising protection in the bargain. 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, on the other hand, centralization of authority must also be available as an option.

The second 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 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 in Multics 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 for 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 cataloging 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.[2] 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 cataloged by the storage system, the principal identifier of the process is compared with those appearing in the access control list of the object; if no match is found access is not 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 the 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. A project administrator decides which users are to be in his project.
3. The third partition places users in named groups called *compartments*. Any user may operate in any of the named compartments, by designating which compartment he wishes to use at the time he authenticates his identity. Compartments are useful when borrowing unaudited programs: a user may indicate that certain of his files can be accessed only by him, and further only when he is operating in compartment "a". He can then be careful to utilize the borrowed program only when he is operating in compartment "b"; the borrowed program cannot access those files restricted to compartment "a".[3]

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 cataloged 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 partition. (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 multicomponent 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 of 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 require a minor change in Multics, namely 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 would then be easily expressed.

We have used the term "object" to describe the entities cataloged 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 catalogs 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.

[2] The Multics access control list corresponds roughly to a column of Lampson's protection matrix [19].

[3] 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. The Multics protection ring scheme [28] provides an alternative method for safely executing borrowed programs.

Table I. Acceptable Combinations of Access Modes for a Segment

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

If we use the letters r, w, and e for these three modes of access, an access control list entry for a segment may specify any of the combinations of access in Table I. Certain access mode combinations are prohibited either because there is no widely useful interpretation (e.g. write access by itself) or correct implementation requires more sophisticated machinery than implied by the simple mode settings. (For example, granting execute access only, while appealing as a method of 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.)

In a similar way, message queues permit separate control of enqueuing and dequeuing 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 to 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, for example, some users read-only access while others can read and write is a powerful capability. 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.[4] 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 goal 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. As mentioned later, under the discussion of weaknesses, this goal has been achieved only partially.

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.[5] Only if the user wishes access to be handled differently than this does he have to take explicit action. Permission to modify the entries in a directory 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 nonexistent person. Such checks were initially implemented in Multics, 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

[4] This feature violates the design principle that selective exclusion is less desirable than selective permission (because of the risk of undetected errors), but has been provided nevertheless to avoid the clumsy alternative of listing every nonexcluded project member.

[5] 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 design principle that calls for naturalness and ease of use. 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.

entries called *links*, which named the original file, and typically contained further restrictions on allowable access modes. In modern terminology, these links were essentially a form of *capability* [11], and the CTSS scheme had several defects common to capability systems but 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 on which link he used. Allowing access rights to depend on the name which happened 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 in 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 Multics [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.[6] 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, in the manner of an interrupt handler, 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 be high. On the other hand, if the trap occurs only at the time the segment is mapped into a user's address space,[7] then the design principle 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 [27], is available.

Both backup copying of segments (for reliability) and bulk I/O to printers and other devices 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.[8] 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 design principle that 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 run a superprivileged program which examines every access control list of the storage system for instances of that name, and delete them. On the other hand, such a systematic search would not discover user programs which initialize access control lists and contain names of now-departed 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 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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.