Honeywell

RECEIVED
APR 16 1976
J. H. SALTZER

PROJECT GUARDIAN
TECHNICAL COORDINATION LETTER

Date: 26 March 1976

TO: Contracting Officer
HQ/ESD/MCP
Hanscom AFB
Bedford, Mass. 01731

TCL No: 17

Contract No: F19628-74-C-0193

Attention: C. E. Fenton, Captain, USAF

Subject: A Technical Note on Discretionary Access Control

The attached technical note discusses "Discretionary Access Control" as it relates to the Multics Security Kernel. Also included, as an attachment, is a draft Honeywell working paper on the "Removal of Directory Control from the Multics Security Kernel." These technical papers are a follow-up to the technical interchange meeting which was held at Honeywell's Cambridge, Mass. offices on 12 February 1976.

If there are any questions on these papers, please contact the undersigned or Mr. N. Adleman at our Cambridge office.

Very truly yours,

HONEYWELL INFORMATION SYSTEMS, INC.

R. L. Carlson
Contract Specialist

Attachment

cc: ESD/MCI (5)
MITRE/D73, Mr. E. Burke (5)
RADC/ISM (3)
NSA/R14 (3)
AFDSC/XMS (2)
CCTC (5)

Removing Directory Control from the Multics Security Kernel

A. Bensoussan

INTRODUCTION.

This document is a proposal to remove directory control from the Air Force security kernel in the Multics System. It is intended to serve as a basis for further discussions with the various groups involved in project Guardian at the Air Force, Mitre, MIT and Honeywell, in order to evaluate its feasibility.

The long range objective is to show that the mathematical model for security, developed by Mitre, can be met by a system for which the top level formal specifications do not refer to directories at all.

The short range objective is to make minor changes to the Multics version known as the "new storage system" in order to obtain a system where the security access rules (i.e., no read up and no write down) are enforced regardless of how directories are manipulated.

This document addresses the short range objective only. The term "security kernel" is used to refer to the set of supervisor procedures and data bases which are necessary to enforce the security access rules.

Removing the management of directories from the security kernel would require restructuring the current ring 0 supervisor into two hierarchical layers. The security kernel would operate in ring 0; it would provide the segment and the process entities and would be responsible for enforcing the security access rules. Directory control would operate in ring 1, under the rules imposed by the kernel, and would use the abstract machine made available by the kernel in the form of a set of ring 0 kernel primitives.

The design of the new storage system has, to a large extent, achieved the separation of directory control and segment control, and would provide a very good basis for implementing this proposal without a major rewriting of the current ring zero supervisor.

THE KERNEL AND THE DIRECTORY TREE STRUCTURE.

One would expect the new kernel to be ignorant of directories. The new storage system has the appropriate modularity to cope with this situation, except for the quota implementation. The quota facility is defined in terms of the directory tree structure, and the procedures that implement this facility must know about the tree structure. Since the implementation of quota is distributed between directory control, segment control and page control procedures, the knowledge of the tree structure, instead of being encapsulated in directory control, has penetrated deeper into segment control and page control. To eliminate the knowledge of the tree structure from the new kernel, one has to reimplement the quota facility in such a way as to perform in directory control those quota manipulations that take place in segment control and page control of the current system (as well as the new storage system). Such an implementation of the quota facility poses no logical problem: the quota information would be stored in directories and managed by directory control; a "page fault" for a page with no disk address would be turned into a "quota fault" handled by directory control; the quota fault handler would check and update the quota information and would call the page fault handler in page control after having authorized this page to use a disk record.

Practically, however, it would require a large effort since the data structures used in the new storage system would have to be changed, a large number of segment control and page control procedures would have to be modified, and new procedures would have to be written. In addition, it would introduce a high overhead since the "quota fault" handler, invoked quite often, would induce new page faults to do its job while, in the current system, all data needed to do the quota checking at page fault time is carefully kept in core memory by page control and segment control.

For a long range project, or for an experimental system, one certainly should consider reimplementing the quota facility in order to simplify the kernel specification as well as its implementation and certification.

For a short range project, what I am proposing instead, is to retain the present implementation of quota, as in the new storage system, and to retain in the security kernel just enough knowledge about the directory tree structure to guarantee that the kernel would never violate the star property when performing a quota operation.

THE BASIC SYSTEM REQUIREMENTS.

In order to remove directory control from the security kernel the system should be endowed with the following properties:

1. Directories must no longer contain any item needed to implement the segment entity or to enforce the security access rules.

2. Access to directories must be subject to the security access rules as if they were user segments.

3. Directory control procedures must execute in a less privileged mode than the kernel.

4. The kernel must guarantee its integrity without using directories and, in particular, without using the ACL protection mechanism provided by directory control

These four requirements are discussed separately below.

SEGMENT ATTRIBUTES REORGANIZATION.

In the old storage system, the first requirement was far from being satisfied. All segment attributes, regardless of their nature, were stored in directories. For example, the file map of a segment, needed to implement the segment entity, was stored in a directory, making segment control vulnerable to directory control since any directory control procedure could modify the file map.

In the new storage system, segment attributes have been reorganized into two groups. The first group consists of those attributes that directory control is responsible for, such as the symbolic names, the access control list, the ring brackets; these attributes are still stored in the branch. The other group consists of those attributes which are needed by segment control and page control to implement the segment entity and the quota facility, as well as the security attributes; these attributes are stored in a data base manipulated exclusively by segment control procedures. This data base is called the VTOC (Volume Table of Contents). There is one VTOC for each disk, describing all segments stored on the disk. The VTOC for a disk is stored in the disk itself, at a conventional location, and consists of an array with one entry for each segment residing on the disk. Each VTOC entry contains the following segment attributes:

- the unique identifier
- the security access class
- the file map
- the current and maximum segment length

- the dates the segment was last used and modified
- the directory switch (*)
- the unique identifier of the parent (*)
- the number of records used (*)
- the quota information (if directory) (*)
- a few other items needed by the salvager

All items marked with an (*) are used by segment control or page control procedures that deal with quota. If the quota facility was entirely implemented by directory control procedures, these items would be moved to the branch (or to some other non-kernel data base), and segment control would totally ignore the existence of directories.

The basic system requirement 1 is entirely satisfied by the new storage system, due to the way segment attributes are split between the directory branch and the VTOC entry.

ACCESS TO DIRECTORIES.

The second requirement states that access to directories must be subject to the security access rules. What is meant by "access" is direct access, hardware access through a segment descriptor word (SDW). A process should never be permitted to modify even a single bit of a directory if the classiciation of the process is not equal to the classification of the directory. A process should never be permitted to read even a single bit of a directory if the classification of the process is not greater than or equal to the classification of the directory.

In the old storage system, there were many instances where a process could not perform under these restrictions:

A process of any classification had to be able to write in a directory of any classification in order to deactivate a segment. The new storage system handles the deactivation of a segment without reading or modifying or locking or having to know anything about the parent.

A high classification process had to be able to modify lower classification directories in order to activate a high classification segment. The new storage system handles the activation of a segment without modifying any bit of any directory.

A high classification process had to be able to modify a lower classification directory in order to lock it since the lock was a word of the directory. In the new storage system, the lock of a directory does not reside in the directory therefore locking and unlocking a directory does not require modifying the directory.

One could probably find other examples where the security access rules had to be violated when accessing a directory. In the old storage system, a simple analysis of the various items stored in a directory would show that they can be classed into three categories, with respect to security: first, those with the same classification as the directory itself, such as the header, the names, the ACLs; second, those with the same classifications as the segments they refer to, such as the size of the segment, the time it was modified, the quota information; and third, those with no classification at all such as the file map, the AST entry pointer. Because of the heterogeneousness of the information recored in a directory, some of the directory manipulations could not be done without reading and modifying the directory regardless of its classification. In the new storage system all items stored in directory are exclusively of the same classification as the directory itself. All items that were not of the same classification as the directory have been eliminated and, as a result, all directory operations that required violating the security rules have been eliminated. Therefore, subjecting directory control to the security access rules would still give all processes enough access to perform all operations they are supposed to perform, and would guarantee that security could not be compromised by a malicious or erroneous directory control procedure.

The basic system requirement 2 is also satisfied by the new storage system.

DIRECTORY CONTROL IN RING 1.

It is clear that, for this proposal to make any sense at all, directory control should not be able to change the kernel. The most natural way in Multics to protect the kernel from directory control is to use the ring mechanism. The kernel would execute in ring zero and directory control in ring 1. This means that all directories would reside in ring 1 and all directory control procedures would execute in ring 1. (The current ring 1 would be moved into ring 2, curently empty).

All supervisor ring 0 gates such as hcs$xxx entry points would become ring 1 gates which may, in turn, call upon a new set of kernel ring 0 gates.

The kernel must present an adequate interface to directory control in the form of a set of kernel primitives to create segments, to delete them, to truncate them, to manipulate those segment attributes which are relevant to directory control but stored in the VTOC entry (such as the quota), to assign segment numbers and to manipulate the access fields of segment descriptor words. The list of these kernel primitives is given in one of the next paragraphs.

The basic system requirement 3 is not satisfied by the new storage system, of course. However, the new storage system provides a very good framework for implementing it since segment control has been made functionally independent of directory control. As a result, the new storage system already exhibits the exact modularity one would expect the system to have for moving directory control into ring 1. In fact, each of the kernel primitives described later is already available as a separate ring 0 procedure.

KERNEL INTEGRITY.

In the current system, the integrity of the kernel is achieved by using the ring protection mechanism and the ACL mechanism. The ring information, as well as the ACL information are stored in directories and manipulated by directory control primitives.

If directory control is no longer part of the kernel, access to all kernel segments must be determined by using information recorded in a kernel data base and manipulated by kernel procedures. The protection mechanism needed to protect kernel segments does not have to be as flexible and sophisticated as the ACL mechanism used to protect user segments because, in most cases, all users are given the same access rights to a given kernel segment, and also because these access rights are not likely to change as often and as freely as the ACL for user segments.

The protection mechanism I am proposing is only one of many possible mechanisms; it is less flexible but simpler than the ACL mechanism and can be described as follows: A kernel segment always has the same ring brackets; it has a single standard access mode used for standard processes, and a single privileged access mode, associated with a single privileged process key, used for privileged processes that have requested and obtained this privilege key at login time. The privileged access mode would allow a trusted system process to call special kernel gates that are not available to normal users. These privileged processes would have to request that a given key be associated with them at login time. This request would be validated the same way the user name, project name, and access class are validated at login time.

This mechanism can easily be implemented in the new storage system. The VTOC entry of a segment could indicate whether or not the segment is a kernel segment. If it is a kernel segment, the access control information would be found in the VTOC entry and would be used by the kernel to manufacture the access field of the segment descriptor word for that segment.

KERNEL PRIMITIVES.

This paragraph provides the list of the primitives the kernel should make available to directory control, in order to make it possible for directory control to perform those functions it is responsible for. All these primitives already exist in the new storage system but they are not implemented as kernel gates and they do not perform any security checking. The purpose of this paragraph is to select from the set of segment control procedures those which need to be a gate, to give a short description of what their functions are, and to give a complete description of the security checking they are responsible for. Special gates available to only privileged processes, such as "declassify", are not relevant to this discussion and have been omitted.

The following notation is used in this paragraph:

```
ERR             = error
s               = segment number
uid             = unique identifier
cl (process)    = clearance of the current process
cl (uid)        = classification of the segment defined by uid
par (uid)       = parent of the segment defined by uid
```

1. create (parent_uid, dirsw, access_class) returns (uid)

   This procedure creates an empty segment (i.e., a VTOC entry) with the access_class defined by the input argument "access_class". The parent of the created segment is defined by its unique identifier "parent_uid", and the directory switch "dirsw" defines whether or not the created segment is a directory. A new unique identifier is assigned to the created segment and its value "uid" is returned to the caller.

   ERR if parent_uid does not denote a directory
   ERR if cl (process) ≠ cl (parent_uid)
   ERR if (cl (process) ≤ access_class) = false
   ERR if dirsw = 0 AND access_class ≠ cl (parent_uid)

2. delete (uid)

   This procedure deletes the segment (i.e., the VTOC entry) defined by its unique identifier "uid".

   ERR if uid not found or if it denotes a kernel segment
   ERR if cl (process) ≠ cl (par (uid))
   ERR if cl (par(uid)) < cl (uid) AND segment to be deleted is not empty

   This primitive provides a "write-down" channel when deleting an upgraded directory. This channel also exists in the current system and is not due to the fact that directory control is outside the kernel. It could be eliminated by

making deletion of upgraded directories a trusted process function.

3. truncate (uid, n)

This procedure truncates the segment defined by "uid", from the word number "n".

ERR if uid not found or if it denotes a kernel segment
ERR if cl (process) ≠ cl (uid)

4. give_quota (uid, q)

This procedure delegates an amount of quota equal to q (q>0) to the directory whose unique identifier is "uid", from its parent.

ERR if uid does not denote a directory
ERR if cl (process) ≠ cl (par(uid))

5. return_quota (uid, q)

This procedure returns an amount of quota equal to q (q>0) from the directory whose unique identifier is "uid" to its parent.

ERR if uid does not denote a directory
ERR if cl (process) ≠ cl (uid)
ERR if cl (process) ≠ cl (par(uid))

6. read_vtoce_item$XXX (uid) return (v)

This procedure has one entry point for each item XXX located in the VTOC entry, that directory control may have to know the value of. The entry point XXX reads the item XXX from the VTOC entry defined by its unique identifier "uid", and returns its value "v" to the caller.

ERR if uid not found
ERR if (cl(process) ≥ cl (uid)) = false

7. write_vtoce_item$XXX (uid, v)

This procedure has one entry point for each item XXX located in the VTOC entry, that directory control may have to change the value of. The entry point XXX selects the VTOC entry defined by its unique identifier "uid", and assigns the value "v" to its item XXX.

ERR if uid not found or denotes a kernel segment
ERR if cl (process) ≠ cl (uid)

8. assign_segno (uid) returns (s)

Assigns a new segment number "s" to the segment whose unique identifier is "uid", and returns the value "s" to the caller.

ERR if uid not found
ERR if (cl (process) ≥ cl (uid)) = false

9. release_segno (s)

Makes segment number "s" invalid in the current process.

ERR if segment number "s" has not been assigned by the assign_segno primitive in this process

10. give_access (s, mode, rings)

Sets, in the segment descriptor word for segment number "s", the ring brackets to the values specified by "rings", and the access field to the mode specified by "mode", adjusted according to the star property.

ERR if segment number "s" has not been assigned
ERR if any ring numbers specified by "rings" is zero
ERR if segment defined by "s" is a kernel segment

11. revoke_access (uid)

Revokes any prior access that has been granted to that segment by the "give_access" primitive in any process.

ERR if uid not found or denotes a kernel segment

The first time a process references a segment to which access has been revoked, an "access_must_be_recomputed" fault occurs, transferring control to the "recompute_access" procedure in directory control. This procedure recomputes the access and calls the "give_access" kernel primitive to set the access bits in the segment descriptor word.

12. lock_directory (uid)

Performs a P operation on a binary semaphore associated with the directory defined by uid.

ERR if uid does not denote a directory
ERR if (cl (process) ≥ cl (uid)) = false

13. unlock_directory (uid)

Performs a V operation on a binary semaphore associated with the directory defined by uid.

ERR if uid does not denote a directory
ERR if (cl (process) ≥ cl (uid)) = false

CONCLUSION.

If the decision is made to implement this proposal, the short range project would consist of (a) giving informal (but precise) specifications of the kernel functions, (b) using these specifications to get a good level of confidence that they represent the mathematical model for security, and (c) modifying the Multics new storage system as proposed in this document.

The long range project would consist of (a) giving formal specifications of the kernel, without refering to directories at all, not even to the tree structure, (b) proving that these formal specifications represent the mathematical model and (c) implement a kernel that meets the formal specifications.

I am very thankful to Jerry Stern for the long discussions he had with me and for his valuable comments and criticisms.

To: Project Guardian Distribution

From: J. Stern

Subject: Discretionary Access Control

Date: 3/15/76

The question of whether or not discretionary access controls should be supported by the Multics security kernel was discussed in a recent meeting attended by representatives from the Air Force, MITRE, and Honeywell. We, at Honeywell, suggested that due to certain inherent vulnerabilities of discretionary access controls and the as yet undemonstrated ability to prove any meaningful assertions about discretionary access controls, there is little or no technical justification for including this mechanism in the kernel. In the course of the discussion, it became apparent that there was considerable confusion concerning the basic issues, i.e., the presumed benefits of including discretionary controls in the kernel and the implied costs. This note attempts to clarify some of these issues.

To begin with, it should be recognized that there is no necessity to justify that discretionary controls be left out of the kernel. This is the proper default assumption for all mechanisms. What must be justified is the desire to put discretionary controls within the kernel. It must be shown that some real objective is achieved by this action and that the cost, in terms of increased kernel size, is warranted.

The original motivation for having discretionary controls was, of course, to enforce the need-to-know policy. Unfortunately, it is clear that the need-to-know policy cannot be fully enforced by discretionary access controls. This is because discretionary controls, even when implemented in the kernel, are inherently vulnerable to casual mistakes, program bugs, and Trojan horse programs. The danger of a casual mistake, of course, exists in the paper system as well as the computer system. However, the consequences of a single mistake, e.g., a wrong keystroke, can be much greater in the computer system. The more subtle dangers of a program bug or a Trojan horse program have no counterparts in the paper system.

If we simply accept the above threats, then clearly our objective in providing discretionary controls cannot be enforcement of the need-to-know policy in any strict sense. Thus, the purpose of discretionary access controls must be to satisfy some lesser objective. Notice that in the case of the security and integrity policies, we do not accept any of these threats. The non-discretionary controls which enforce these policies cannot be

circumvented by user mistakes, program bugs, or Trojan horse programs.

We must define precisely what we intend to prove about discretionary controls. It has been suggested that we wish to prove that the ACL mechanism works correctly. One can only prove the correctness of a mechanism relative to some specific assertions about its operation. At present, we do not have an appropriate set of assertions for discretionary controls. The "discretionary security property" defined by Bell and LaPadula [1] is one assertion that could be proven, but in itself, does not appear sufficient to guarantee the desired operation of discretionary controls. This is because the discretionary security property ignores the hierarchical control of ACLs. Thus, a kernel that permits anyone to arbitrarily change an ACL could be proven to satisfy the discretionary security property. This would not be an interesting result.

Let us assume, however, that appropriate assertions could be developed and that the correctness of discretionary controls could be proven. What claims could we then make about discretionary controls? Unfortunately, errors contained in non-kernel programs could still cause unintended release or modification of information. One could argue that by providing a correct kernel, we at least make it possible for some user to produce his own closed subsystem of certified programs. This argument is neither interesting nor realistic. It is not interesting because we know that the vast majority of users will use uncertified programs, both system-supplied and private. In fact, if an uncertified supervisor is provided which must be invoked to reach the kernel, then it becomes impossible to construct a certified subsystem. It is not realistic because the effort required to certify even a modest replacement for the Multics operating system is prodigious. That is precisely why we have adopted the security kernel approach.

The most important claim that a correct ACL mechanism supports is that it cannot be directly circumvented. If a user is denied access by the ACL mechanism, then he cannot bypass this restriction without the willful or unwilliful cooperation of an authorized user. A malicious user can only wait and hope for another user to fall victim to a mistake, program bug, or Trojan horse program; but the malicious user cannot force any of these incidents to occur.

The confidence this claim can give us in the use of discretionary controls is difficult to assess. For example, let us consider the operation of granting access. A correct ACL mechanism can ensure that the only users able to grant access to a given segment are those users having modify permission to the parent directory. We cannot be certain, however, that access to the segment will be granted as intended. Neither can we be sure that the ACL of the parent directory is correct. Essentially, errors

in non-kernel programs can cause any ACL to be incorrect relative to our intentions. However, the ability to change an ACL is correctly controlled by another possibly incorrect ACL.

To a large extent we can expect that errors in uncertified programs will gradually be discovered over time and repaired. However, in a constantly changing system like Multics, where new errors can be introduced, it is not clear that the number of errors will decrease in time. More importantly, it is likely that errors of the type about which we are most concerned will go undetected for long periods of time. This is because errors that permit unintended access will not be noticed under normal circumstances.

In the final analysis, our confidence in discretionary controls hinges on our belief that errors in the uncertified programs we use are both rare and of limited harm. Given these conditions, errors in uncertified programs can be considered sufficiently random that their exploitation would be unlikely. Unfortunately, we simply do not know what conditions are given. We can only speculate as to the quantity and quality of errors contained in non-kernel programs. Therefore, even if the ACL mechanism is certified, the risk of need-to-know violations still remains unknown. If, indeed, errors in non-kernel programs are inconsequential, then certifying the ACL mechanism may substantially reduce this risk. On the other hand, if more serious errors exist in non-kernel programs, then certifying the ACL mechanism may not significantly reduce this risk. The point is that we have no way of measuring how much good it will do to certify the ACL mechanism. We know that no matter how much effort is applied, discretionary controls will still be vulnerable. If the certification of discretionary controls were a trivial task, we could ignore these points. But this does not appear to be the case. The prospect of devoting a substantial portion of our time and energy to a task with uncertain benefits is cause for concern.

Let us now turn our attention to the implied costs of including the ACL mechanism in the kernel. The basic ACL mechanism in itself is fairly cumbersome when compared to security and integrity controls. Worse yet, however, is the fact that a decision to place ACLs within the kernel is, in effect, a decision to place all of directory control within the kernel. This is necessary because the ACL representation is stored in directories and because the directory hierarchy serves as an authority structure for controlling ACLs.

A recent proposal by Andre Bensoussan [2] describes a scenario for removing directory control (and discretionary controls) from the present Multics ring 0. This scheme alone could result in a significant reduction in kernel size. And, with somewhat more effort, it appears that all knowledge of the hierarchy could be removed from the kernel (including that currently retained for

the quota mechanism).

In order to obtain a rough estimate of the size of directory control relative to other kernel functions, Lee Scheffler investigated the composition of the present Multics ring 0 supervisor. His investigation showed that the addition of directory control functions to other anticipated kernel functions can result in a kernel size increase of 36 to 48 percent as measured in lines of code. These findings are reported in Appendix A. It is apparent from these figures that a decision to include the ACL mechanism within the kernel will be costly.

In addition to directory control, there is also the area of user authentication. Clearly, since the basis for discretionary access control is user identities, these identities must be authenticated by the kernel. If not, any requirement for placing discretionary controls within the kernel becomes ludicrous. We note with interest a recent design note [3] prepared by MITRE which asserts that user authentication will not be performed by the kernel. If this is, indeed, the case, then in our view, the issue is settled. There is no requirement for kernel-supported discretionary controls. If this is not the case, then inclusion of user authentication and related answering service functions within the kernel will impose a high penalty in terms of increased kernel size. Apparently, MITRE has recognized this penalty to be unjustifiable and has chosen to avoid it.

In summary, we see limited benefits and high costs for including discretionary controls within the kernel. Even if cost/benefit considerations are overlooked, there is still the undemonstrated ability to prove useful assertions about discretionary controls. Without such assertions and a technique for proving them, there should be no requirement for placing discretionary controls within the kernel.

In view of this conclusion, we are pursuing a kernel design having two distinct layers. The inner layer will implement non-discretionary controls and the outer layer will implement discretionary controls. Initially, only the inner layer will be specified and certified. A decision regarding the outer layer will be postponed until we have gained more experience and are better able to evaluate the additional cost of specifying and certifying the outer layer. This two-layer approach has the advantage of confining the certification of non-discretionary controls to the inner layer kernel. This is desirable whether or not the outer layer is later certified for discretionary controls.

References

1. D.E.Bell and L.J.LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation", MTR-2997, The MITRE Corporation, Bedford, Mass., July 1975

2. A.Bensoussan, "Removing Directory Control from the Air Force Security Kernel", to be published as an MTB

3. W.L.Schiller, "Preliminary Specification of the Answering Service", Multics Design Note #33, MITRE

Appendix A

Multics Kernel Size Estimates

This appendix presents size estimates of a Multics kernel with and without discretionary access control (ACL) and directory control functions. These estimates are based entirely on the sizes of programs currently implementing these functions in ring zero of Multics. The primary intent of these figures is to show the impact, measured in increased code to be certified, of making discretionary access control, and therefore directory control, part of the certified Multics kernel.

These figures were derived as follows. Each source program in the current Multics hardcore libraries was extracted. For PL/1 programs, the number of non-declaration PL/I statements (instances of ";") were counted. (Nested conditionals terminated with a single semicolon were counted as one statement.) For ALM (assembly language) and MEXP (macro assembler) programs, the number of non-comment, non-blank and non-include statement lines were counted. For programs and data bases in other languages (e.g. error_table_), the number of non-comment and non-blank lines were counted.

These figures were combined in a single segment. For each program, a character string was added to identify

1. Whether the program implemented a function that would be required for kernel, salvager, or backup functions (exclusive of system initialization); and

2. Whether the program's primary function was to implement discretionary access control (ACLs) or directory control, or whether a major part of its operation depended upon the existence of directory control (such as the pathname associative memory programs).

Finally, these figures were sorted according to classification 1, and sums taken (separately for PL/1, ALM, MEXP, and others) with and without modules in classification 2. The results are listed in table A-2.

The postulated Multics kernel of classification 1 includes a minimum of functions required for steady state system operation. These functional areas are listed in table A-1.

Table A-1

Functions of a Multics Kernel

- page control
- segment control
- volume management
- address space management
- disk I/O
- peripheral I/O (iol)
- terminal I/O (code conversion removed)
- kernel I/O (to operator's console)
- traffic control and interprocess communication (stripped down)
- fault and interrupt handling
- system error logging (syserr)
- security and integrity control
- discretionary access control
- directory control

The following functional areas were specifically excluded because their status relative to inclusion in the Multics kernel is not clear:

ARPA network

resource management (rcp_)

tape and printer I/O programs

dynamic linking

System initialization functions, including both pre-bootload activities (e.g., system header and boot tape generation) as well as bootloading, are excluded from these figures for several reasons. First, initialization functions do not operate in the steady state environments of the kernel, salvager or backup. They may not need the same type or degree of certification as that required for certification of the steady state. Secondly, functions such as initialization header generation involve humans, and the nature of the certification of such activities is not yet clear. Finally, the overall structure and scope of system initialization is not yet clear. For these reasons, estimates based on current Multics initialization would be misleading and would not be comparable to estimates of other functional areas for purposes of projecting programming and certification effort.

Notes for Table A-2

The column headed "#programs" lists the number of separate programs and data bases, separately by language type, and in total.

The column headed "#statements" lists the total statement counts for each language. To make these figures comparable for PL/I and non-PL/I programs, the raw statement counts for ALM programs (shown in parentheses) were reduced by a factor of 4. This represents an approximation to the number of PL/I-like statements that would be required to implement identical functions. This factor is lower than the normal PL/I-to-ALM expansion factor of 6 because these programs are originally written in ALM, where better code size optimization is possible. The statement counts for MEXP and other language programs were not so reduced because these languages are macro-languages of a comparable code-expansion ratio to PL/I.

(1) This line represents the programs and data bases in the current Multics hardcore libraries, excluding obsolete programs. These figures include many functions not necessary to kernel operation.

(2) This line represents programs currently implementing the kernel functions listed in table A-1, except for ACLs and directories. It shows the amount of code to be certified relative to steady state assertions, with the exceptions of backup and salvaging.

(3) This line represents programs currently implementing ACL and directory functions necessary in a kernel (not including backup and salvaging functions). It shows the increase over (2) in the total amount of code to be certified relative to steady state assertions due to the addition of ACLs and directories to the kernel.

(4) This line represents programs currently implementing the kernel functions (line (2)) with the addition of backup and salvaging functions. (This includes physical volume salvaging and segment backup functions.) Backup figures are based on the current (pre-NSS) backup system, and may change with NSS backup. (Backup represents about 40% of the increase over (2)). Figures for salvaging are based on the NSS salvager. This line shows the total amount of code to be certified if ACLs and directories are outside the kernel. exclusive of system initialization.

(5) This line represents programs currently implementing ACL and directory functions that would be required for kernel, backup, and salvaging functions. It shows the increase over (4) in total code to be certified due to the addition of ACLs and directories, exclusive of system initialization.

Table A-2

Multics kernel size estimates based on current Multics programs

(See detailed notes on previous page)

		#programs	(%increase)	#statements	(%increase)
(1) Current Multics hardcore	pl1	437		43976	
	alm	144		5275	(21098)
	mexp	22		6392	
	other	5		454	
	total	608		56097	
(2) Multics kernel without ACLs or directories	pl1	133		11550	
	alm	63		2366	(9465)
	mexp	6		545	
	other	2		454	
	total	204		14915	
(3) Increment to kernel for ACLs and directories	pl1	41	(30.8%)	5271	(45.6%)
	alm	1	(1.6%)	13	(53) (5.6%)
	mexp	2	(33.3%)	166	(30.5%)
	other	0		0	
	total	44	(21.6%)	5450	(36.5%)
(4) Multics kernel + salvager + backup without ACLs or directories	pl1	150		13326	
	alm	67		2430	(9719)
	mexp	6		545	
	other	2		454	
	total	225		16755	
(5) Increment to kernel + salvager + backup for ACLs and directories	pl1	54	(36.0%)	7908	(59.3%)
	alm	1	(1.5%)	13	(53) (5.5%)
	mexp	2	(33.3%)	166	(30.5%)
	other	0		0	
	total	57	(25.3%)	8087	(48.3%)