ISSUES IN DISCRETIONARY ACCESS CONTROL

Deborah D. Downs, Jerzy R. Rub, Kenneth C. Kung, Carole S. Jordan

Abstract

This paper discusses the types of mechanisms that can be used to implement Discretionary Access Control (DAC). It also covers the access types that can be controlled by a DAC mechanism. and includes a brief discussion of related topics including protected subsystems; administering, auditing, and verifying DAC; and DAC implemented as an add-on to an operating system. Finally this paper discusses how the DAC information presented in this paper will be used by the Department of Defense Computer Security Center in the preparation of a DAC Guideline to aid system designers and developers in their selection of DAC mechanisms.

1. Introduction

One of the features required of a secure operating system is discretionary access control (DAC) which is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access authorization may, at its discretion, be capable of passing (perhaps indirectly) that access type or a subset on to any other subject. The objective of this paper is to provide information on the various DAC issues that are of concern to computer vendors, system designers and developers.

2. Terminology

Subjects are those entities that initiate activities which cause information to flow between objects. Usually these entities are persons, but processes or devices can also be subjects. Hence, a user who performs any actions on files is a subject; a job, which the user schedules to be run later, is a subject when it is running; and a device such as a power sensing unit which initiates backup routines upon detecting power failures can be a subject. In most interactive systems a user logs on, and a process starts and does work on behalf of the user. The process takes on the attributes of the user, such as access rights, and the process associated with a user is the subject. Generally, subjects are held accountable for the actions that they have initiated, and the audit trail associates with one subject any security relevant action performed on an object.

Objects are those entities that contain or receive information. Depending on the system, objects may include, but are not limited to, records, blocks, pages, segments, files, directories, directory trees, mailboxes, messages, and programs, as well as bits, bytes, words, fields, processors, communication lines, clocks, and network nodes. Subjects may also be treated as objects. For example, if a process may spawn child processes, those processes may be treated as objects.

In systems where the smallest amount of information that is normally handled as a unit is a file, each file is an object. But if each file can be broken into smaller pieces so that each piece can be individually manipulated (such as segments or pages), then each of these smaller pieces is an object. In addition, if the files are organized into a tree structure, then the directories for these files are also objects. In this guideline, "file" will be used as a generic term for files, segments, etc.

In some systems all the objects are treated logically as files, and DAC is handled by associating the access information with these files. Hence, each of the hardware devices (i.e. disks, terminals, printers) is treated as a file, with access control information associated with it. In order to access any of these devices a subject must have the proper access rights, and the security checking mechanism for the device is the same as the mechanism used for standard files. For example, writing to a terminal involves moving information to the file associated with this terminal. The access control information associated with this file determines which subjects are or are not allowed to write to this terminal. Consistency in handling DAC objects leads to less complexity in the DAC implementation and, therefore, to more assurance in its correct operation.

The number of objects to be protected by the DAC mechanism depends on the environment for which the system is intended. Almost all systems include in their DAC mechanisms files, directories (if the file system is tree structured), communication channels, and devices. General purpose operating systems trying to provide a more complete and user friendly DAC interface also include objects such as mailboxes, messages, and bulletin boards and their entries as objects protected by DAC. Again, the tradeoff is user friendliness and broader security versus the complexity of the DAC mechanism and the difficulty of assuring its correctness.

3. An Inherent Deficiency in Discretionary Access Control

DAC controls restrict a subject's access to a subset of the

protected objects on the system. The subject is also restricted to a subset of the possible access types available for those protected objects. The set of objects and access types can change dynamically based on whatever criteria the subject and/or other subjects wish to employ. Criteria such as "need to know" and "who do I like" are equally possible. Access is based entirely on the subject's identity and the mechanism has no knowledge of, and bases no decisions on, the semantics of the data.

Therefore the identity of the subject is crucial, and if actions can be performed using another person's identity, then DAC can be subverted. Thus the basic definition of DAC makes it vulnerable to Trojan Horses[1]. On most systems, any program which runs on behalf of the subject acts with the subject's identity and therefore has all of the DAC access rights of the subject's process.

The software produced by the computer system manufacturer, especially if the system has a high EPL[2] rating, should not contain Trojan Horses. Configuration management, testing, and trusted distribution should assure this. If a trusted user-created source and/or object module has been properly protected by DAC and has not executed in an environment with a Trojan Horse, it will also be free of Trojan Horses. But software written by software houses or by other untrusted users could easily contain Trojan Horses. Software with a Trojan Horse, running on behalf of a subject, not only could access the subject's protected objects, but also could copy the subject's objects (those which have read access) to a data space accessible to the subverter. The Trojan horse could also change the subject's DAC so that the subverter could have continuing access to the subject's protected objects. A Trojan Horse could also append code to all of the subject's executable objects so that, when those objects were executed by another user, the code would give the subverter access to all newly accessible objects and would attach the same code to them. This has been termed a "virus."[3]

The DAC Trojan Horse problem could be restricted in a system that implemented many domains (the set of objects that a subject has the ability to access) or dynamic small domains for each process, such as a capability based system or protected subsystems that supply a domain per process. In most systems today, with only user and supervisor domains, all of a subject's objects are available to a process running on behalf of a subject. If domains were created dynamically for a process, with only the necessary objects available in that domain (implementing the least privilege principle[4]), then a Trojan Horse would be limited to accessing only those objects. In most current general purpose computing environments, DAC cannot protect objects from users who are determined to gain access to them.

4. DAC Mechanisms

In order to implement a complete DAC system the information that is represented by the access control matrix model[5] must be retained in some form. An access control matrix has subjects represented on the rows and protected objects on the columns. The entries in the matrix describe what type of access each subject has to each object. Current operating systems have attempted to retain the access control matrix using either row or column based representations since storing the entire matrix is inefficient because it is sparsely populated. The implementations of row-based representations are, in some form, attaching to the subject a list of accessible objects and include:

- Capabilities
- Profiles
- Passwords.

The column-based representations that in some form attach a list of accessing subjects to the object include:

- Protection Bits
- Access Control Lists.

The following discussion describes each mechanism and presents its pros and cons.

4.1. Capabilities

Capability[6] based systems provide dynamically changeable domains (name spaces) for processes. Ability to access an object is demonstrated when a subject has a capability or "ticket" to the object. This capability contains allowable access rights (e.g., read, write, execute). Capabilities can be added to and deleted from a process during its execution, thereby changing the size of its name space. In some implementations, programs can contain capabilities and capabilities can also be stored in data files. Hardware and software mechanisms or encryption provide protection from alteration. Capabilities can be passed along to other processes and can sometimes be either increased or decreased in scope depending upon the access characteristics they contain. Because capabilities implement dynamic domains, ideally, they can limit the objects accessible to any program to the minimal set necessary for it to accomplish its task. This would limit, to some extent, a Trojan Horse's access to a subject's protected objects and to objects to which the stolen data could be output.

Because the ability to pass capabilities is not controlled by any policy, and because capabilities can be stored in files so that giving access to a file may give access to many other objects, in general one cannot determine for a particular object all the subjects that have access to it. Thus a complete DAC implementation, including revocation and access review, is impossible with capabilities. At this time, few systems have been implemented with capabilities and very few, if any, have attempted to implement a complete DAC mechanism. Some research has been conducted in restricting capabilities by overlaying a DAC mechanism.[7]

4.2. Profiles

Profiles associate a list of protected objects and access rights with each subject. When a subject attempts to access an object, the subject's profile is checked for the required access rights. Three main problems with this mechanism exist: restricting the size of the profiles, distributing access when an object is created or when the access is changed, and determining all the subjects that have access to an object. Since object names are usually not consistent or amenable to grouping, the profile for a subject that has access to many protected objects can get very large and difficult to manage. Also all protected object names must be unique, so fully qualified object names must be used.

Creating, deleting, and changing access to protected objects may require many operations since multiple subjects' profiles may have to be updated. When users create objects and want to give themselves and others access to the object the profiles must be updated in a secure manner. Users cannot be allowed to update their own or other user's profiles directly. CA-Sentinel's DOS/VSE[8] security add-on uses profiles and only allows the security administrator to change the profile. With such an implementation no user, including the creator, would have access to a new object until the security administrator updated the appropriate profiles, since as a general security principle access to an object by a subject must be null unless specific access is granted. Security administrator controlled profiles are extremely restrictive and would not be usable in environments where objects are created and/or access rights are changed frequently.

Timely revocation of access to an object is very difficult unless subjects' profiles are automatically checked on each access to an object. Deleting an object requires some method of determining which subjects have the object in their profile. In general, with profiles as with capabilities, answering the question of who has access to this protected object is very difficult. Since this question is usually important in a secure system and management of profiles is difficult, profiles are deficient as a DAC mechanism.

4.3. Passwords

Password protection of objects attempts to represent the access control matrix by row and involves associating with an object a password that must be presented to the operating system before access is granted. If each subject possesses its own password to each object, then the password is like a ticket to the object, similar to a capability system (except, of course, no dynamic domains exist). Most DAC implementations using passwords allow only one password per object or one password per object per access mode. Passwords on protected objects have been used in IBM's MVS[9] and with other mechanisms in CDC's NOS[10] to implement DAC.

Using a password protected DAC system poses many problems. It is virtually impossible for a user to remember a password for each protected object, especially if they have access to many objects, and when the passwords are stored in programs or files they are vulnerable since no hardware protection is provided. To restrict access to certain access modes requires a password for each combination of access modes. But in most systems that use passwords, access to a protected object is either total or nonexistent. In such implementations revoking a subject's access requires revoking access from all other subjects with similar access and then distributing a new password to those who are to retain access. This procedure becomes almost impossible when passwords are stored in programs or files. To be secure, passwords should be changed periodically, which is very difficult to do in such password protected DAC systems. In systems such as MVS, the default access to a file is unrestricted access. A file is protected only when the password protection is initiated for that file.

If passwords are used as in CDC's NOS to supplement another DAC mechanism they do have one positive aspect. If all objects are protected with different passwords, a Trojan Horse can be restricted to only the objects that are handed to it. An alternative to passwords that has the same problems, but adds extra protection when a DAC system is not trusted, is the use of encryption to protect objects.

4.4. Protection Bits

Protection bits are an incomplete attempt to represent the access control matrix by column. Implementations include systems such as Unix[11] that use protection bits associated with objects instead of a list of subjects that may access an object. In the Unix case the protection bits indicate a set of access modes for all subjects a single group and the owner of the protected object. The subject that creates the object is the owner, and ownership can only be changed through superuser privileges. The owner is the only one (besides a superuser) who can change the protection bits. Subjects can belong to more than one group but they can only belong to one group at a time (there is a current active group). The group name (the group in which the owner was active at creation of the object) and the owner name are listed with the protection bits.

The problem with any implementation similar to protection bits is that implementing the access control matrix model is virtually impossible. The system cannot allow or disallow access to a protected object on any single subject basis. Groups set up to specify any needed combination of subjects has been suggested, but the combinatorics of such a solution are impractical. Since groups are controlled by system administrators, such a scheme would certainly require their full time attention. Also, since only one group can be specified per object, different, non-owner subjects cannot be given different access types to an object.

4.5. Access Control Lists

Access Control Lists (ACLs) permit any particular subject to be allowed or disallowed access to a particular protected object. They implement the access control matrix by representing the columns as lists of subjects attached to the protected objects. Each entry in the list is an identification of a subject(s) and its authorized access to the object . The lists do not have to be excessively long if groups and wild cards are used.

Within the current technology, ACLs are the best way to implement a DAC mechanism. ACLs are the only mechanism that allows inclusion and exclusion of an individual subject. Therefore the following discussion of ACLs is more detailed concerning options than the previous sections. Also the discussion of DAC access types will be couched in terms of ACLs.

4.5.1. Groups

Groups are a mechanism for representing sets of subjects. In the ACL, multiple subjects can be identified by a group name. The ACL is sorted so that access identified by specific subject-id is ordered before access identified only by group. If more specific subject identification exists, it is used to compute access. Before implementing a group mechanism, the designers must decide what the groups are to represent. A group can be a shorthand way of referring to a set of subjects, an accounting mechanism, a structuring mechanism for the file system, a method of grouping project work, a means of identifying and restricting the accessible objects (implementing the least privilege principle), or a combination of these and other concepts.

Honeywell's Multics[12] is an example of a system that uses groups

to represent projects, for structuring the file system, to implement the least privilege principle and for accounting. Multics implements a group mechanism (called projects) that allows subjects to be active in only one group at a time, although they can be members of many groups. Several reasons exist for this requirement. Since Multics uses a group mechanism for accounting and defining the storage hierarchy, allowing only one active group simplifies the algorithm for determining whom to charge for computer time and where in the storage hierarchy segments are to be searched for and stored. Allowing a subject only one active group also simplifies access decisions. A subject could belong to two groups that have different access rights to the same object. In the worst case, one group could have null access to the object, while the other had full access to the object. If the subject has only one active group at a time, decisions are not required. Changing a subject's active group on Multics requires a log out and a new log on because the process' environment must be reset. Maybe the most important reason to have only one active group is to implement the least privilege principle. When a subject logs on to a particular project, the subject is restricted to only the objects accessible to the user by subject-id or through the project group-id. At best, only the objects necessary to do that project's work are accessible. To work on another project requires essentially a new log on and a change of reference. Unfortunately in Multics, since projects are also associated with the accounting and file system, they may tend to be of larger granularity than desired for least privilege.

If groups are used only as a shorthand way of referring to a set of subjects, an efficient implementation of groups allows subjects to always be active in all groups in which they have membership. In other words, to specify an active group or to default to an active group at log on is not necessary, and if the subject has membership in any group on an ACL, it always has that access right. The decision as to what access should be given when the subject has current membership in multiple groups which have different access rights should be based on the rights available by changing active groups on a Multics-like system. The union of the access from all group memberships, with null access treated as the empty set, should be given to the subject. Therefore if one group has null access and another has read, the subject should be given read access to the object. If the system supports other negative access modes. the decision process becomes more complicated. Implementing groups with this mechanism does result in some access rights being different from a Multics-like system. If a subject is a member of both group A, which has read access to object 1, and group B, which has write access to object 1, under the Multics-like system the subject could never read and write the object at the same time. However the subject could copy the object and then give both groups read and write access to the copy. On the other hand, in a system with multiple groups active for access, the subject would be given read and write access to the object. Using this method also means that after unsuccessfully checking for the subject-id on the ACL, all of the group entries must be compared to the list of groups of which the subject is a member.

In a DAC implementation where subjects can have active membership in many groups at once, the group mechanism could still be used for accounting and/or storage hierarchy management by specifying a specific group for only those purposes. Such a change in a Multics-like mechanism (still restricting a subject to only one active group at a time for the least privilege principle) would allow for a more flexible use of groups with a finer granularity.

The Apollo's Domain[13] system has a multiple, hierarchical group mechanism. The ACL entry has the form "subject-id.project.organization.node." As in Multics, if the ACL specifies access rights for the subject only (subject-id.*.*.*), then all other group access rights are ignored. This allows inclusion and exclusion of any single subject. In the Apollo's Domain system, as with Multics, if a subject is not on the ACL by subject-id, but is a member of a group that is specified on the ACL with no subject-id (*.project-id.*.*), the group rights are used, and organization and node memberships are not examined. But with Domain, if the subject is not identified by subject-id or group-id and the ACL contains an entry that specifies the subject's organization or organization and node (*.*.organization.* or *.*.organization.node), the subject will be allowed the specified access. The same process follows for the node entry in the ACL. Multiple group mechanisms add more complexity that may facilitate administrative control of a system, but do not affect the basic utility of a DAC mechanism.

Controlling the creation of groups is important, since becoming a member of a group can change the ability to access many objects. In many systems, e.g., Multics, a subject must be a member of at least one group. One detriment of any group mechanism is that changing the members of a group results in changing the effective access granted by every ACL containing that group name (usually an unknown set). Creation of groups should be only a Systems Administrator function or be distributed as a Project Administrator type function. Problems result from allowing all subjects to create groups and then be "owner" of that group. Subjects could choose group names that would mislead other subjects that wish to use that group on their ACL (e.g., "Top Secret"). Subjects should not be allowed to list the members of groups because of possible covert channels and privacy; therefore it is difficult to determine which group is the correct one to use. Group names and subject-ids must be controlled to prevent their re-use. Re-use of a subject-id or group name could result in unexpected access to objects.

Performance considerations exist in the design of the group mechanism. If groups proliferate and all groups on the ACL must be searched for the subject's membership, keeping a list of group memberships with the subject will probably be necessary. Otherwise a performance price is paid for searching the ACL for the subject's membership whenever the file is opened.

4.5.2. Wild Cards

A simple wild card mechanism allows the substitution of one or more characters where the wild card is specified. In ACLs wild cards can be used in two basic ways. In the first, the entire subject-id or group is replaced when the wild card is specified. For example in Multics, an ACL entry of "*.DODCSC" gives access to any subject who is a member of the DODCSC group. An ACL entry of "Downs.*" gives Downs access, no matter in what group the subject Downs is currently active . An ACL entry of "*.*" gives access to any subject in any group. The exclusion capability is possible by specifying a subject or group with an access type of "no-access." The group and wild card mechanisms allow the ACL list to be kept to a reasonable size. In fact, studies on the Multics system on the ARPAnet at MIT showed that the average ACL length was about 3-4 entries, and only 2% of

the ACLs had over 10 entries, but some of the ACLs had as many as 200 entries. On a system where the use of groups was not restricted by requiring that only one group be active or by accounting and storage system considerations, as it is in Multics, the very large ACLs are not necessary.

The other use of wild cards is to allow the wild card to substitute for a portion of the subject-id or group name. For example, "Dwo*" could allow all subjects access whose subject-id began with Dwo (e.g. Dwons, Dwo, Dwo111111) In some systems, this use of the wild card substitutes for a group mechanism. Effective use of this wild card mechanism requires subject-ids to be chosen so that the group is part of the name. For example, all members of the security project could be given subject-id s that end with "sec" and the ACL could contain "*sec " Obviously choosing subject-ids, allowing membership in multiple groups, and revoking someone's membership from one group become very complicated operations. Sorting the ACL entries to place the most restrictive entries first requires a complicated algorithm and rigid subject-id specification rules If a group mechanism is available, allowing wild cards to substitute for particular characters does not add functionality and only complicates the use of the DAC mechanism. Therefore, the use of wild cards should be restricted to complete substitution for a subject-id or group name.

4.5.3. Default ACLs

Many side issues exist with regard to the implementation of ACLs. Default ACLs are usually necessary to ensure user friendliness in the DAC mechanism. At the very least, in an owner type system when an object is created by a subject, the subject should be placed on its ACL by default. Some of the other possible default mechanisms include a system-wide default, a subject-associated default, or, if the file structure is a tree, a default associated with the directory.

A system-wide default cannot be tuned to allow access judiciously, but can be used as the default in cases where no other default has been specified. A system-wide default might give access only to the creating subject. A subject-associated default works well on a system with a flat file structure. When a subject is first entered on the system, its default ACL is specified.

For tree-structured file systems, a default(s) associated with the directory is efficient. If the subject organizes its directory structure to represent project work or areas of interest, then the ACLs for all objects in a sub-tree are similar. One default ACL in the directory is for children that are files. For children that are directories, either a separate sub-directory default ACL is specified or the default ACL has to be stated explicitly by the subject. Otherwise, unless care is taken using this mechanism, subjects with access to the root sections of the storage hierarchy receives access by default to all of the storage hierarchy. The overriding principle of least privilege implies that the use of defaults should not inadvertently give away more access than the subject intended. In other words, it is better to err on the conservative side. In all implementations some subject(s) must have permission to change the ACLs after they have been set by default, and a way to change the defaults must exist.

4.5.4. Named ACLs

An implementation of ACLs that is sometimes proposed is "named" ACLs. This mechanism is implemented by attaching to the protected object either the ACL name or a pointer to the named ACL. Thus a separate database of named ACLs is needed. The difficulty with this implementation is that when the named ACL gets changed, access to all of the objects pointing to that ACL also get changed. Determining all of the protected objects affected by the change is very difficult. The named ACLs proliferate since many objects need unique ACLs Determining whether the correct named ACL already exists may be difficult, and allowing subjects to search named ACLs may divulge sensitive information on access rights and provide covert channels The named ACLs also have to be protected in the same way as the real ACLs.

5. DAC Access Types

In this section, access permissions and access modes for objects are discussed. In the access control matrix model, access permissions and access modes are the entries that specify what kind of access rights the subject has to the object.

The access permissions define who has the ability to change access modes and/or the ability to pass to another subject that ability. In other words, subjects that have an access permission to an object may change the ACL of that object and, perhaps, pass this ability to other subjects. These two abilities, which should be implemented as separate access permissions, are used to specify the control of the DAC mechanism.

In contrast, access modes indicate a specific action that can be applied to the object. For example, the 'execute' mode allows the object to be executed. Access modes will be discussed in more detail later.

6. Access Permissions

In many current systems the concept of access permissions is not separated from access modes. For example, in Multics, modify access on the directory is actually the ability to change the ACLs of the children of the directory. Access modes and access permissions should be kept conceptually separate on any DAC system This separation allows control of the object to be separated from access to the object.

Since access permissions allow a subject to change the ACL of the object, they can be used to implement the control model for a DAC system. Three basic models exist for control in a DAC system; hierarchical, owner, and laissez-faire.

6.1. Hierarchical

Permissions to change the ACL of objects can be organized so that control is hierarchical, similar to the way most business organizations are formed. A simple example would have the systems administrator at the root of the hierarchical tree. He/she would have the ability to change the ACL and to pass on that ability. The system administrator (or the company structure) divides everyone else into subsets (e.g., departments). The default ACL for each department gives the department head permission to change the ACL and the permission to pass on the ability to change the ACL. The department heads divide their subordinate subjects into subsets (e.g.,projects) and set the defaults so that for each project, they give the project heads the permission to change the ACL. The subjects at the bottom of the

hierarchy have no access permissions on any object. Notice that in the hierarchy those who have the ability to change the ACL on an object can give themselves any access mode on the object.

The advantages of a hierarchical structure are that control can be placed in the most trusted hands and the control can mimic the organizational environment. The disadvantage of the example hierarchical structure is that multiple subjects have the ability to change the ACL on an object.

Other hierarchical organizations can be imagined and implemented using access permissions. Hierarchical organizations could also be programmed into the DAC system without using access permissions. However, such a restrictive implementation would result in a choice of a specific hierarchical structure which would hinder use of the system by organizations that did not fit that mold.

6.2. Owner

Another control model associates with each object an owner (usually the creator of the object) who is the only subject that has the permission to change the ACL of this object. The owner is always in "full" control of the objects that he/she has created and has no ability to pass control of the original object to any other subject. Hence the owner may change the ACL at any time in order to grant and deny to other subjects the access modes to the objects under their control

This near absolute control by the owner can be implemented administratively with a DAC mechanism that supports access permissions. The system administrator could set up the system so that each subject would have a "home" directory. The default ACL for files and sub-directories would always give that subject the access permission to change the ACL, and the access permission to pass on the ability to change the ACL would never be used on the system. Of course the system administrator has the ability to alter all of the access control on the system. Owner control can be viewed as a limited hierarchical system of only two levels and could be implemented with a flexible hierarchical control model.

Another way to implement owner control is by programming it into the DAC system without implementing any access permissions. The DAC mechanism stores the identity of the creator of the object as the owner and the only subject that can change the ACL, with no method of passing the capability of changing the ACL to another subject. Such an implementation is restrictive but has been used on many systems. A strict ownership policy results in the owner being the only subject that can delete an object. If the owner leaves the organization or dies, it takes privileges such as Unix's *super user* to delete the subject's objects. Unix is an example of an operating system where only the owner of an object has the ability to change its ACL and no access permissions are implemented.

Another disadvantage of owner control is that for non-owner subjects to change their access modes for an object is difficult (i.e., to share the objects is harder). In order to gain or lose any access to an object, subjects must ask the owner of the object to change its ACL for them. However, in certain operating environments, this disadvantage is actually a desired system characteristic.

6.3. Laissez-faire

In a laissez-faire scheme, the creator of an object may pass to any subject they wish the ability to change the ACL and the ability to pass on that ability, and no ownership concept is present. Once a subject has passed on the permission to pass the ability to change an ACL to still other subjects, they may pass this permission to other subjects without the consent of the creator of the object. Hence, once the access permissions are given away, control of an object is difficult. The ACL will show all the subjects that could change the ACL of an object, but not which subject, if any, is in charge of the final decisions about the object.

Such a control mechanism could work very well on a egalitarian research system, but is not recommended as the only possible control mode on any system.

7. File Structure

Use of access permissions is also related to the structure of the storage system. In operating systems where the storage system is tree structured, two types of objects are involved: directories and files. Access permissions can be applied to both. Permission to change the ACL on a file just changes access to that particular file, but permission to change the ACL on a directory may be implemented as the ability to change not only the directory's ACL but also the ability to change the ACLs of all of the children of the directory (this will be called extended dir ACL change). This implements a hierarchical control structure since permission to change the directory ACL means that the subject can change any access on the whole subtree below that directory.

The **hierarchical** control scheme described above would use the extended dir ACL change and the permission to pass on the ability to change an ACL. Giving the system administrator, the department heads, and the project leaders extended dir ACL change to the correct directory in the storage structure (with the tree organized to represent the hierarchical structure) would give them control over the correct portion of the file system. A pure **owner** control policy could be implemented without using the extended dir ACL change. The subject who creates a file or a directory would receive the ability to change the ACL of either. **Laissez-faire** could use the extended dir ACL change together with the ability to pass on the ability to change an ACL on both the directories and the files. This would only produce a slightly more complicated state of anarchy.

Depending on the situation, one or a combination, of the above schemes could be used to implement the control model of the DAC mechanism. In general, the use of access permissions could allow the organization to structure the DAC mechanism to best fit the particular environment. Of course, the tradeoff for this flexibility is a more complex DAC mechanism and a more difficult training process in learning to set up a DAC that is correct for a particular environment.

8. Access Modes

A fairly wide range of access modes are available on various computer systems implementing DAC mechanisms. This section discusses various modes and describes a minimal set. The discussion begins with the most basic protected objects supported by the system: files.

- **Read-copy** allows an object to be read and copied.

On most, if not all, systems, the "read" mode is actually implemented as read-copy. Conceptually a read mode that allowed only display of the object would be valuable. However, the implementation of a display-only mode as a basic access type would be very difficult since it would involve displaying the file only on media with no storage capacity. Read-copy only restricts the subject to reading or copying the "original" object. If subjects copy the object, they may set any access rights they want to the copy.

- **Write-Delete** allows a subject to modify the contents of an object in any manner they choose including expanding, shrinking or deleting.

Many additional types of write access modes have been used in different systems for controlling what changes can be made to an object. Examples of such write access modes include write-append, delete, and write-modify. These access modes apply only when the system understands something about the characteristics of the object. For example, a hardware and/or operating system may have several more specific write modes that apply to the support of index sequential files. Computer systems that support several different file types with differing write access modes may either map those modes into a single, or minimal, set of modes to be supported by DAC, or all of the possible write modes can be specified and only a subset would apply to a particular file type. The former simplifies the DAC mechanism and the user interface but the latter gives a finer granularity of access control. In contrast, systems with virtual memory file systems like Multics, and, to some extent, stream files like Unix, cannot enforce varieties of write access on the basic file or segment objects.

Of course, the basic write-delete access is not really useful without read-copy access. But giving a subject read-copy access without giving write-delete access will often be useful.

- **Execute** allows a subject to run the object as an executable file.

On many systems execute access requires read access. For example, in Multics, operations involving constants and finding entry points, etc. in the linkage section are seen as reading the object text, and therefore read access is necessary to execute a segment. But more basic problems exist in Multics-like environments where the address space is created for the lifetime of the process and each program is executed in that address space. The subject has the ability to manipulate the address space (change descriptors or registers) before or while a program is running. Almost any program can be manipulated to copy itself if its environment can be manipulated (e.g., find a move-long instruction, change the pointers and execute). So even if the DAC allowed an execute only access, in a Multics-like environment that access cannot be enforced. The solution is to create a new process (and therefore address space) to run each program; for Multics this is too expensive. On any system the execute access should also control where execution can begin and where calls to other programs return; execute access should enforce defined entry points. A correct implementation of execute without read allows proprietary programs to be protected from copying.

- **Null** grants no access permissions and is used to allow exclusion of a particular subject in an ACL.

Often a null access mode does not exist but is implied in the ACL by specifying no access modes for a particular entry.

The minimal set of access modes for an object that is a **file** is the set of access modes used on many existing systems (e.g., Unix, Multics); read-copy, write-delete, and execute, with null implied by no access modes. These access types supply minimal, but sufficient, granularity in limiting the access to a file. With any smaller set a file cannot be controlled independently with respect to read, write, or execute.

Most operating systems apply DAC to objects other than files. Many times these other objects are really structured files, and the system understands the semantics of the object. These objects usually have "extended" access modes which are relevant to the particular structure of the object. They are usually implemented in a manner similar to data abstractions in that the operating system maps the "extended" access modes down to the basic access modes.

8.1. Directories

If the files are organized in a tree structure, then the directories are used to represent the non-leaf nodes of the tree. Directories are usually implemented as structured files or segments. Whether access modes are associated with directories depends on how the tree structure is used to control access.

Three methods can be used to control the access to the directories and their associated files:

1. access control on the directories but not on the files
2. access control on the files but not on the directories
3. access control on both the files and the directories.

If access mode controls are placed only on the directories, then once a subject is given any access to a directory it has that access to all files under the directory. Of course if one of the objects under this directory is another directory (a sub-directory), then the subject needs access mode permission to the sub-directory before it can access the objects under the sub-directory. Placing access control only on directories requires subjects to group files according to access types. This requirement could be too restrictive and could conflict with other reasons for grouping files.

If access mode controls are placed only on files, the controls are more granular. Access to individual files are controlled independently of other files under the same directory. But if no access controls are placed on directories, subjects could browse through the storage structure looking at the names of other subjects' files. Moreover, file placement would be uncontrolled, thereby defeating the main purpose of a tree structure.

The most efficient way to implement DAC with ACLs on a tree structured file system is to control access at both the directory and file levels. However, the designer must then make a decision as to whether the subject must have access to the whole path to access an object, or whether access to only the object itself is

sufficient. For example, Multics designers made the decision that if the subject knew the correct pathname to an object, and had non-null access to the object, then non-null access to the intervening directories was not necessary. This decision makes checking for legal access much easier and allows a user to grant another user access to an object by merely changing the ACL on that one object. If the user does not know the correct pathname to an object and does not have access to the intervening directories, no way exists to determine the pathname of the object, and therefore no way exists to access it. Allowing access to an object without access to the parent directories complicates decisions about when access should be given to the other attributes of a file (e.g., length, date contents modified). Such decisions depend on the particular implementation.

In Unix the lack of access to the directory implies that no access is available to the entire sub-tree controlled by that directory. A user cannot give another user access to a file without having given that user access to the parent directories.

The minimal set of access modes for an object that is a directory should include read and write-expand.

- Read allows a subject to see the directory entries (i.e., the names, the ACLs, and the associated information about files and sub-directories in this directory).

Read access implies the ability to access the children of the directory, depending on their own ACLs

- Write-expand-delete allows a subject to add new objects to the directory (i.e., to create and delete files and sub-directories under the directory).

Since directory access modes are extended access modes and depend upon how the directory is structured, the actual access modes that would be implemented for a directory are very system dependent. For example, Multics implements three access modes for directories: status allows a subject to see the attributes of the directory and its children; modify allows the subject to modify those same attributes including deletion; and append allows new children to be created.

The tradeoff in determining what objects should be included in the system DAC mechanism and how many access types should be implemented for each object is between the user friendliness of the operating system and the complexity of the DAC mechanism. The DAC mechanism is part of the Trusted Computing Base and therefore must be included in the assurances necessary in the Criteria. At the A1 level, the mechanism to support each new protected object and its access mode will have to be verified. Also, the final implementation of the access modes should not be so complicated that subjects cannot easily remember the implication of each mode. If subjects cannot distinguish the functions of each access mode, they may just grant to other subjects either the full access rights or no access rights to an object.

Other objects that have been protected with a computer system's DAC mechanism include mailboxes (message queues in general), communication channels, forums (a type of bulletin board), and devices. Their access types are dependent on the implementation and will not be discussed here.

9. Protected Subsystems

In order to provide users with access types finer than the standard *read, write, execute,* etc., a rather limited number of systems support protected subsystems. Saltzer and Schroeder[14] refer to the need for "protected subsystems", which they define as user-provided programs which control access to files.

By extending the access control mechanism to allow objects to be accessed in ways specified by subjects (programs as well as users), a system may allow a specific user-written program to be designated "the only subject" allowed any access or a particular type of access to specific files. A protected subsystem can be defined as a collection of procedures and data objects that is encapsulated in a domain of its own so that the internal structure of a data object is accessible only to the procedures of the protected subsystem, and the procedures may be called only at designated domain entry points and only by designated subjects.

The encapsulated data files are the protected objects. Programs in a protected subsystem act as managers for the protected objects and enforce user-written access controls on them. Subjects outside the protected subsystem are allowed to manipulate the protected objects only by invoking the manager programs. Any access constraints that can be specified in an algorithm can be implemented in a protected subsystem. Giving users the ability to construct protected subsystems out of their own program and data files allows users to provide arbitrary controls on sharing.

By allowing programs inside a protected subsystem to invoke programs in another protected subsystem without compromising the security of either, multiple protected subsystems can be used to perform tasks. This limits the extent of damage a malicious or malfunctioning borrowed program might have to objects protected by the subsystem. Likewise, the lending user could also encapsulate the loaned program in a protected subsystem of its own to protect it from the programs of the borrower.

9.1. Domain, MTS, and Others

Apollo's Domain and University of Michigan's Michigan Terminal System (MTS) implement limited protected subsystems. Both systems allow users to write their own subsystems, but they do not provide mechanisms for protected subsystems to interact with each other. The general claim is that implementation of multiple protected subsystems would require additional hardware and/or extensive software assistance.

The two systems use different approaches in implementing protected subsystems. In Domain, a user has to have an ability to add a subsystem name to a system-wide list of subsystems, thereby creating the subsystem. The user then assigns a certain file to be a manager of the subsystem and another file to be an object. The user then removes all other access to the object, making the manager an exclusive accessor. Access to the subsystem is controlled by the ACLs associated with the subsystem manager.

During execution, the system verifies that both the manager and the object are in the same subsystem and that the manager has raised its privilege level allowing it to access its own protected objects. Only then is the subsystem manager allowed to operate

on the object. The additional step of raising the manager's privilege level is designed to prevent any unintentional access to the object.

In MTS the connection between the manager and the object is done via user-named "program keys." A user attaches the key to the manager program and sets access on the object to that key MTS makes each key unique by prefixing it with the user name The user then removes all other access (if any) to the object and for additional protection sets the access to the manager to *execute-only*. A user does not need special privileges to set up a protected subsystem.

During the association of the program key with the object, the user is also required to specify a basic access mode that the manager will be using. That is, if the manager needs only a *read* access to the object, the object should be made only read-accessible to the manager. This further enhances the protection of the object.

Other operating systems provide features which can be used as limited protected subsystems. In Digital Equipment Corporation's TOPS-10[15] a "file daemon" mechanism allows a user to better fine-tune access to objects. Control Data Corporation's NOS is capable of object protection with access control lists as well as with passwords. If passwords on objects are kept secret from all but the managers, then protected subsystems can be implemented in NOS. Unix has a "setuid" privilege which allows the process running a program to have the access rights of the program's owner. Multics' ring structure can be viewed as a mechanism to support protected subsystems.

9.2. Features of a Good Protected Subsystem

If protected subsystems are to be implemented, the capability should be designed and built into an operating system during the design and implementation of the operating system itself. Adding it on later is difficult.

A good implementation of protected subsystems should provide a straight-forward means for creating the subsystem. The connection between the manager and the object should not be obscured with secret passwords. The operating system should protect the subsystem so nothing is revealed about the object or the manager other than their interface with the user.

Objects should be allowed to have more than one manager so that various groups of users could access the same objects through specific managers. Managers should not be allowed to manage more than one subsystem to avoid making the manager more powerful and difficult to control than originally intended. Managers should be further controlled with the basic access modes provided by the operating system for objects. Managers must be allowed to manage multiple objects in the same subsystem. Not only user-prepared programs, but system programs and utilities, should qualify as potential managers.

Protected subsystems could provide a capability for increasing the integrity of applications by providing the users a means of implementing data abstractions. The mechanism to ensure the ability to set up protected subsystems could be part of the Trusted Computing Base (TCB), but the subsystems themselves would not. The user could be assured that the manager programs were the only object manipulators, but no assurance of their correct operation, other than the basic access checks supported by the operating system, would be provided.

10. Administering DAC

To set up the DAC mechanism requires a Facilities User's Guide that explains how to properly administer a DAC system, and how to identify the system security administrator and any project administrators and their functions. The correct access must be set on the system administration database, which would include user registry, group definitions, etc. The guide must explain how access must be set to handle fixing system problems, including viewing dumps and repairing file systems, etc. Access must be set correctly to the standard libraries, including the system commands and subroutines, initialization modules, and the operating system source code. ACLs may need to be set on the supervisor entry points and the I/O daemons or device queues

In general, documentation should describe how to initialize the ACLs to best provide a protected environment If the DAC mechanism is flexible and provides the ability to set up different control structures, examples should be given for a range of control structures.

11. Auditing DAC

Auditing is an important part of a secure computer system In general an audit log message should include a time stamp, the subject's identity, the object's identity, the type of action and any other pertinent information Much of the auditing on any operating system is not specific to the DAC mechanism, but in many instances the DAC mechanism should place an entry in the audit log. Any operation that changes the control structure of the DAC should be audited. Any change in any ACL, including the default ACLs, should be audited. Any attempt to access a DAC protected object should be audited Any changes to group definitions should be audited.

12. Verifying DAC

Verifying the correctness of the DAC mechanism is difficult. As has been illustrated in this paper, DAC does not have well defined rules as do mandatory access control systems. In fact no formal model of DAC currently exists although some mention is made of the DAC mechanisms in the Bell and LaPadula[16] formal model for mandatory access control. In systems such as SCOMP,[17] whose design has been formally verified for security, the only property that was proven about the DAC mechanism was that it did not violate the mandatory access formal model. Research is needed in developing a formal model for DAC, and verification of systems where DAC is important should prove at least that the DAC mechanism is consistent with its descriptive top level specification.

13. DAC Add-ons

Systems such as IBM's MVS have had DAC mechanisms added on. The DAC checks are made before the operating system gets a request for an object, and the request is not forwarded to the operating system if it requests illegal access. Such add-ons are basing their assurance on the inability of a user to get access to the operating system through another interface or to cause the operating system to retrieve illegal data with a legal request. Most

add-on packages are applied to systems that provide little security on their own. Thus the assurance that the operating system could not be subverted is non-existent.

I. PREPARATION OF A DAC GUIDELINE

The Department of Defense Computer Security Center (DoDCSC) was established in 1981 to provide uniform DoD policy for security requirements, controls and measures to reduce the threat of compromise of classified and sensitive information processed in computer systems. The main goal of the DoDCSC is to encourage the widespread availability of trusted computer systems. In support of that goal a metric was created, the DoD Trusted Computer System Evaluation Criteria,[18] against which computer systems could be evaluated for security. Since one of the features required of a secure system is a DAC mechanism, several vendors and system designers have expressed a need for guidance from DoDCSC on how to build and implement effective DAC mechanisms. In response to this need, the DoDCSC is preparing a DAC Guideline.

I.1. PURPOSE OF THE DAC GUIDELINE

The purpose of the DAC Guideline is to:

1. discuss the issues involved in implementing and evaluating various DAC mechanisms.

2. provide information to the evaluator which will aid in assessing the effectiveness of a DAC mechanism in meeting the requirements of a particular evaluation class defined in the Criteria, and

3. provide guidance to systems designers and developers on how to build DAC mechanisms.

I.2. PROPOSED CONTENTS OF THE DAC GUIDELINE

The contents of the DAC Guideline have not been finalized at this time. Most of the information presented in this paper will be included in the Guideline. The following additional topics are proposed and may be changed or added to as the Guideline is being developed.

- DEFINITION OF DISCRETIONARY ACCESS CONTROL. The DAC definition, as it appears in the Criteria, will be included along with a narrative interpretation of the definition.

- DESCRIPTIONS OF DAC MECHANISMS. Currently-used DAC mechanisms, such as Capabilities, Profiles, Passwords, Protection Bits, and ACLs, will be described, including their limitations, strengths, and weaknesses. The Guideline will include recommendations as to which mechanism or DAC features can be used to satisfy the requirements of a particular evaluation class of the Criteria. Emphasis will be placed on 'good practice', not just on the minimum requirements for a particular evaluation class. Good practice means that the DAC mechanism should be designed so that users can easily use the computer to enforce the same information access rules that would be enforced in an all-paper world.

- ENVIRONMENTAL CONTROLS. The Guideline will include information on the environmental security controls (physical, personnel, and administrative) on which various DAC mechanisms depend in order to be effective.

- FACILITIES USER'S GUIDE INFORMATION. DAC mechanisms can be degraded by improper use, therefore, the Guideline will include information on the preparation of a Facilities User's Guide that explains how to properly set up and administer a DAC system.

- MAPPING THE CRITERIA AGAINST DAC MECHANISMS. The Guideline will give an overview and interpretation of the DAC requirements as they appear for each evaluation class in the Criteria.

I.3. SCHEDULE FOR PUBLICATION OF THE DAC GUIDELINE

A first draft of the DAC Guideline will be written and distributed for review by the end of July 1985.

I.4. VENDOR PARTICIPATION IN THE PREPARATION OF THE DAC GUIDELINE

Several computer hardware and software vendors are in the process of installing, or have already installed, DAC mechanisms in their systems which are intended to meet the requirements of the DoD Trusted Computer System Criteria. Much of the technical information presented in this paper was collected at the request of the DoD Computer Security Center by the Aerospace Corporation in response to the need and the interest shown by vendors.

Representatives of the DoD Computer Security Center plan to visit computer hardware and software vendors to discuss the DAC technical paper developed by the Aerospace Corporation. Discussions with each vendor will be aimed at answering the following questions:

- Has the vendor installed any of the DAC mechanisms described? Does the vendor have any experience with the reliability, user-friendliness, or implementation problems of any of the mechanisms that they have installed?

- How difficult would it be for the vendor to change a current DAC mechanism or add a new mechanism if necessary to meet a particular class of the Criteria?

- Does the vendor know of other DAC mechanisms which were not covered in the DAC technical paper?

In order to produce a DAC Guideline that is relevant and helpful to the vendor community, vendor reaction to the DAC technical paper and vendor feedback to draft versions of the DAC Guideline will be solicited by the DoD Computer Security Center.

References

1. Schroeder, M.D., *Cooperation of Mutually Suspicious Subsystems*, PhD dissertation, M.I.T., 1972.

2. DoD Computer Security Center, *Evaluated Products List for Trusted Computer Systems*. DoD. CSC-EPL, 1984.

3. Cohen. F., "Computer Viruses - Theory and Experiments," *7th Security Conference*, DOD/NBS, September 1984, pp. 388-402.

4. Saltzer, Jerome H., "Protection and the Control of Information in Multics," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 388-402.

5. Lampson, B.W., "Protection," *Proc. Fifth Annual Princeton Conference on Information Sciences and Systems*, Princeton University, March 1971.

6. Fabry, R.S., "Capability Based Addressing," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 403-411.

7. Karger,P.A. and A.J. Herbert, "Lattice Security and Traceability of Access," *Symposium on Security and Privacy*, IEEE, April 1984, pp. 13-23.

8. Computer Associates, *CA-SENTINEL Reference Guide*, 1983.

9. IBM, *Access Method Services*, 1983.

10. Control Data Corporation, *NOS Version 2 Reference Set*, 3 ed., 1983.

11. UC Berkeley, *Unix Programmer's Manual*, 7 ed., 1981.

12. Honeywell Informations Systems, Inc., *Multics Programmer's Manual -- Reference Guide*, 7 ed., AG91.

13. APOLLO Computer Inc., *The DOMAIN System Administrator's Guide*, 3 ed., 1983.

14. Saltzer, Jerome H. and Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, Vol. 63, No. 9, September 1975, pp. 1278-1308.

15. Digital, *DECSYSTEM10 Users Handbook*, 1983.

16. Bell, D.E. and LaPadula, L.J., "Secure Computer Systems: Unified Exposition and Multics Interpretation," Tech. report MTR-2997 Rev. 1, MITRE Corp., March 1976.

17. Benzel Vickers, T., "Overview of the SCOMP Architecture and Security Mechanisms." Tech. report MTR-9071. MITRE Corp.. September 1983.

18. DoD Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD, CSC-STD-001-83, 1983.