NCSC-TG-003
VERSION-1

NATIONAL COMPUTER SECURITY CENTER

A GUIDE TO UNDERSTANDING DISCRETIONARY ACCESS CONTROL IN TRUSTED SYSTEMS

30 September 1987

Approved for Public Release:
Distribution Unlimited.

NATIONAL COMPUTER SECURITY CENTER
FORT GEORGE G. MEADE, MARYLAND 20755-6000

NCSC-TG-003-87
Library No. S-228,576

FOREWORD

This publication, "A Guide to Understanding Discretionary Access Control In Trusted Systems," is issued by the National Computer Security Center (NCSC) under the authority of and in accordance with Department of Defense (DoD) Directive 5215.1, "Computer Security Evaluation Center." The guidelines defined in this document are intended to be used by computer hardware and software designers who are building systems with the intent of meeting the requirements of the Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD.

Recommendations for revision to this publication are encouraged and will be reviewed biannually by the NCSC through a formal review process. Address all proposals for revision through appropriate channels to:

National Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755-6000

Attention: Chief, Technical Guidelines Division

Patrick R. Gallagher, Jr. 30 September 1987
Director
National Computer Security Center

ACKNOWLEDGEMENTS

Special recognition and acknowledgement for their contributions to this document are extended to the following:

Carole S. Jordan, National Computer Security Center (NCSC), as primary author and preparer of this document. Dr. Deborah Downs, the Aerospace Corporation, who prepared an in-depth technical report on DAC mechanisms that became the major input to this document. Grant Wagner and Steve LaFountain, NCSC, who contributed their technical expertise and assistance throughout this effort. Dr. Dixie B. Baker, the Aerospace Corporation, who meticulously reviewed the document and suggested many changes.

Special thanks are extended to the many computer vendor representatives who enthusiastically gave of their time and technical expertise in reviewing the material and providing valuable comments and suggested changes. Among the vendor representatives who were so helpful were: Steven B. Lipner, Digital Equipment Corp., Dr. Roger Schell, Gemini Computers, Earl Boebert, Honeywell, Inc., and C.T. Bougas, Harris Corporation.

Contents

1. INTRODUCTION

The main goal of the National Computer Security Center is to encourage the widespread availability of trusted computer systems. In support of that goal a metric was created, the Department of Defense Trusted Computer System Evaluation Criteria (the Criteria) Il], against which computer systems could be evaluated for security.

2. PURPOSE

One of the features of the Criteria that is required of a secure system is the enforcement of discretionary access control (DAC). DAC 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 user or process given discretionary access to information is capable of passing that information along to another subject. This guide discusses issues involved in designing, implementing and evaluating DAC mechanisms. Its primary purpose is to provide guidance to manufacturers on how to select and build effective DAC mechanisms. Any examples of DAC mechanisms in this document are not to be construed as the only implementations that will satisfy the Criteria requirement. The examples are merely suggestions of appropriate implementations. The Criteria is the only metric against which systems are to be evaluated. In addition to showing examples of DAC mechanisms, this guide will restate and elaborate on the Criteria requirements for DAC. This guide is part of an on-going program to augment the Criteria on the issues and features it addresses.

3. SCOPE

The guidelines and interpretations established in this document apply to the discretionary access control requirements as expressed in the Criteria. The recommendations apply to computer systems and products that are being built or modified with the intention of meeting the requirements of the Criteria.

4. CONTROL OBJECTIVES

The term ``control objective'' refers to a statement of intent with respect to control over some aspect of an organization's resources or processes. In terms of a computer system, control objectives provide a framework for developing a strategy for fulfilling a set of security requirements. In particular, a given system can only be said to be secure with respect to its enforcement of some specific policy [2]. The control objective for security policy is as follows:

> "The security policy is a statement of intent with regard to control over access to, dissemination of, and modification of information. The security policy must be precisely defined and implemented for each system that is used to process sensitive information. The security policy must accurately reflect the laws, regulations, and general policies from which it is derived.''

Discretionary control is the most common type of access control mechanism implemented in computer systems today. The basis of this kind of security is that an individual user, or program operating on the user's behalf, is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control. Discretionary security differs from mandatory security in that it implements the access control decisions of the user. Mandatory controls are driven by the results of a comparison between the user's trust level or clearance and the sensitivity designation of the information.

Discretionary controls are not a replacement for mandatory controls. In any environment in which information is protected, discretionary security provides for a finer granularity of control within the overall constraints of the mandatory policy. Both discretionary and mandatory controls can be used to implement an access control policy to handle multiple categories or types of information, such as proprietary, financial, personnel or classified information. Such information can be assigned different sensitivity designations and those designations enforced by the mandatory controls. Discretionary controls can give a user the discretion to specify the types of access other users may have to information under the user's control, consistent with the overriding mandatory policy restrictions. In a classified environment, no person may have access to classified information unless: (a) that person has been determined to be trustworthy, i.e., granted a personnel security clearance - MANDATORY, and (b) access is necessary for the performance of official duties, i.e., determined to have need-to-know - DISCRETIONARY. The discretionary security control objective is:

> Security policies defined for systems that are used to process classified or other sensitive information must include provisions for the enforcement of discretionary access control rules. That is, they must include a consistent set of rules for controlling and limiting access based on identified users who have been determined to have need-to-know for the information.

5. DEFINITIONS

Discretionary Access Control (DAC)-The Criteria defines discretionary access control as:

> ``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
> `permission is capable of passing that permission (perhaps indirectly) on
> `to any other subject."

DAC controls are used to restrict a user's access to protected objects on the system. The user may also be restricted to a subset of the possible access types available for those protected objects. Access types are the operations a user may perform on a particular object (e.g., read, write, execute). Typically, for each object, a particular user or set of users has the authority to distribute and revoke access to that object. Users may grant or rescind access to the objects they control based on "need to know" or "whom do I' like" or other rules. DAC mechanisms control access based entirely on the identities of users and objects.

The identity of the users and objects is the key to discretionary access control. This concept is relatively straightforward in that the access control matrix, defined below, contains the names of users on the rows and the names of objects on the columns. Regardless of how the matrix is represented in memory, whether by rows or by columns, the names of the users and objects must be used in the representation. For example, in a row-based representation an entry might read the equivalent of ``KIM can access KIMSFILE and DONSFILE". In a column-based representation, one might find the equivalent of "DONSFILE can be accessed by DON, JOE and KIM".

Other Definitions:

access control matrix-a two-dimensional matrix representing users on the rows and objects on the columns. Each entry in the matrix represents the access type held by that user to that object. Access control matrices are usually sparsely populated and are represented in memory by row or by column, eliminating storage requirements for empty entries. Sea figure 1 in Section 7 for an example of an access control matrix.

access mode-entries in the access control matrix that specify certain operations a user may perform on an object, (e.g., read, write, execute, delete).

access permission-permission of a user to access an object in some manner. Entries in the access control matrix specify access permission. No access or ``null'' access may also be specified if desired.

control permission-a certain access mode that allows users to grant/revoke access permission> and change access modes to objects. Sometimes this includes the ability to pass control permission to other users.

defined groups-groups which have been defined to the DAC mechanism before being used in access control decisions. Groups are normally defined by users with special privileges, such as the system administrator. A group should be defined by listing the identities of the members to be included in the group.

least privilege-this principle requires that each subject in a system be granted the most restrictive set of privileges needed for the performance of authorized tasks. The application of this principle limits the damage that can result from accident, error, or unauthorized use.

named users-users that are uniquely identified to the TCB. The unique identifier is to be used by the DAC mechanism to perform access control decisions.

object-a passive entity that contains or receives information. Access to an object potentially implies access to the information it contains. Examples of objects can be: records, blocks, pages, segments, files, directories, directory trees, and programs, as well as processors, video displays, printers, network interfaces, 1/0 ports, etc.

ownership-a concept in which one user has total control over access to an object. A subject operating on behalf of that user is normally the creator of the object and is totally responsible for controlling access to it.

subject-an active entity, generally in the form of a process or device operating on behalf of a user that causes information to flow among objects or changes the system state. Technically, a process/domain pair.

Trojan horse-a computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process. An example is a program that makes a "blind copy" of a sensitive file for the creator of the Trojan horse.

Trusted Computing Base (TCB)The totality of protection mechanisms within a computer system - including hardware, firmware, and software - the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system.

*-property--A Bell-LaPadula security model rule, called the ''star*property,'' that allows a subject write access to an object only if the security level of the subject is dominated by the security level of the object. Also known as the Confinement Property.

6. AN INHERENT DEFICIENCY IN DISCRETIONARY ACCESS CONTROL

6.1 A FUNDAMENTAL FLAW IN DISCRETIONARY ACCESS CONTROL

Discretionary access control mechanisms restrict access to objects based solely on the identity of subjects who are trying to access them. This basic principle of discretionary access control contains a fundamental flaw that makes it vulnerable to Trojan horses [3], [4]. On most systems, any program which runs on behalf of a user inherits the DAC access rights of that user [5]. An example of the workings of a Trojan horse will illustrate how most DAC mechanisms are vulnerable. Reference [6] contains such an example, which is similar to the following:

6.2 AN EXAMPLE OF A TROJAN HORSE

Consider a system where an access control list mechanism (as described in Section 7.3) is used to implement discretionary access control. There are two users on this particular system: an honest user, DOE; and a dishonest user, DRAKE. DOE has a data file which contains highly sensitive data; this file is known as DOESFILE. He has diligently set the ACL to allow only himself to read the file. No other users are authorized to access the file. DOE is confident that no one but himself will be able to access his data file.

DRAKE is determined to gain access to DOESFILE. He has legitimate access to the system which allows him to implement a useful utility program. In this utility DRAKE embeds a covert function to read DOESFILE and copy the `contents into a file in DRAKE's address space called DRAKESFILE. DRAKESFILE has an ACL associated with it that allows processes executing on DOE's behalf to write to it, while allowing DRAKE's processes to read it.

DRAKE induces DOE to execute his utility program by telling him how useful and efficient it is. DRAKE is careful not to tell DOE about the covert function (Trojan horse) that is resident in the utility program. DOE executes the corrupted program and it appears to perform perfectly. However, while it is operating on DOE's behalf, it assumes his identity and thus his access rights to DOESFILE. At this time it copies the contents of DOESFILE to DRAKESFILE. This copying takes place completely within the constraints of the DAC mechanism, and DOE is unaware of what is happening.

This example should make clear the danger of Trojan horse attacks and the inadequacy of most DAC mechanisms to protect against such attacks. It should be noted that an elaborate DAC mechanism may provide illusory security to users who are unaware of its vulnerability to Trojan horse attacks.

Configuration management, testing, and trusted distribution should ensure that software produced by the computer system manufacturer does not contain Trojan horses, especially if the system has a high EPL rating. However, software from other sources does not come with these assurances. In very high threat environments, it is wise to assume that unevaluated software does contain Trojan horses. This assumption dictates that discretionary access control not be used as the sole protection mechanism in high threat environments.

The Trojan horse threat can be reduced in systems that implement many domains or dynamic small domains for each process. In most systems today, with only user and supervisor domains, all of the user's objects are available to a process running on that user's behalf. If domains were created dynamically for each process, with only the necessary objects available in that domain (implementing the least privilege principle), then a Trojan horse would be limited to accessing only those objects within the domain [5], [7].

A reference monitor which implements a mandatory security policy which includes the *-property would provide robust protection against Trojan horse attacks. The mandatory access control implementation would prevent the Trojan horse from disclosing the information to a user who is not permitted access to the information under the mandatory access rules. Assume the same scenario as was described previously with the following changes. The computer system now implements a mandatory security policy with two hierarchical sensitivity levels. For the sake of simplicity, the levels are called sensitive and non-sensitive. DOE operates at the sensitive level, and DOESFILE is sensitive. DRAKE is not authorized to access sensitive data, so he operates at the non-sensitive level. DRAKE is only allowed to read non-sensitive files, so DRAKESFILE is non-sensitive. As before, DRAKE's Trojan horse program is executed by DOE. The program takes on the sensitivity level and the identity of DOE. Within the constraints of the mandatory and the discretionary security policies, the program reads DOESFILE. However, when the Trojan horse tries to write the sensitive data to DRAKESFILE, the reference monitor disallows the operation. Since the Trojan horse is now executing at the sensitive level, the program cannot be allowed to write to a non-sensitive file. That would be a violation of the *-property [1].

This example should show the reader that discretionary access control is only effective to restrict the ``honest'' user in browsing through other users' files and preventing accidental disclosure or destruction of information. The malicious user who is determined to gain unauthorized access to data on the computer system must be restricted by other means, such as mandatory access controls.

7. AN OVERVIEW OF DAC MECHANISMS

This section presents an overview of the most commonly used DAC mechanisms. Each mechanism will be described briefly. Section 9 explains how each of these mechanisms rates against the Criteria.

Implementing a complete DAC system requires retaining the information that is represented by the access control matrix model in some form. An access control matrix has users represented on the rows and protected objects on the columns (see figure 1). The entries in the matrix describe what type of access each user has to each object. Current operating systems have attempted to represent that information using five basic mechanisms:

1 Capabilities
2. Profiles
3. Access Control Lists (ACLs)
4. Protection Bits
5. Passwords

ACCESS CONTROL MATRIX

Objects Users:	KIMSFILE	DONSFILE	PAYROL1	PAYROL2	DOESFILE
Kim	rw	r	rw	r	
Joe		r			
Don		rw	r		
Jones			r		
Doe					rw
Mgr Jim	cp	cp	c	c	c
Jan			rw	rw	

Figure 1

The access control matrix such as the example in figure 1- above, is a pictorial view of a set of users and their access permissions to a set of protected objects. The access type ``r'' denotes read access and the access type ``w'' denotes write access. The access type ``c'' means control permission and the access type ``cp'' means control with passing ability. The access types are explained in Section 8.

Capabilities and profiles represent the access control matrix information by row, connecting the accessible objects to the user. ACLs and protection bits represent the access control matrix information by column, connecting a list of users to an object. As the balance of this section will demonstrate, ACLs are the most flexible and usable DAC mechanism that can be implemented with existing technology.

7.1 CAPABILITIES

In a capability-based system, access to protected objects such as files is granted if the would- be accessor possesses a capability for the object. The capability is a protected identifier that both identifies the object and specifies the access rights to be allowed to the accessor who possesses the capability. Two fundamental properties of capabilities are that they may be passed from one accessor (subject) to another, and that the accessor who possesses capabilities may not alter or fabricate capabilities without the mediation of the operating system TCB.

Capability-based systems [8] provide dynamically changeable domains (name spaces) for processes to run in. Ability to access an object is demonstrated when a process has a capability or ``ticket'' to the object. The capability also contains allowable access modes (e.g., read, write, execute). In some implementations, programs can contain capabilities or capabilities can be stored in files. They are protected by hardware and software mechanisms or by encryption. Capabilities can usually be passed along to other processes and can sometimes be increased or decreased in scope.

A pure capability system includes the ability for users to pass the capability to other users. Because this ability is not controlled and capabilities can be stored, determining all the users who have access for a particular object generally is not possible. This makes a complete DAC implementation, including revocation, very difficult. (Revocation may not be an issue, however, since a user who has access to an object can make a copy of the information in another object. Revoking the user's access on the original object does not revoke access to the information contained in the user's copy. After revocation, however, changes can be made to the original object without the knowledge of revoked users.) For a discussion on revokable capabilities, see Redell's paper [9].

Since capabilities implement dynamic domains they can ideally limit the objects accessible to any program. This would limit a Trojan horse's access to only the protected objects handed to it. 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 [10].

Capabilities could be useful in enforcing the least privilege principle and providing dynamically changeable domains, making discretionary access controls less vulnerable to Trojan horse attacks. However, in order to pass the class C2 and above DAC requirements, the ability for users to pass capabilities to other users must be sufficiently controlled. There could be some design difficulties in building capability-based mechanisms to satisfy the B3 DAC requirement because of difficulty in implementing precisely defined groups [11]. Also, at class B3 it is required that users be able to specify a list of users that have permission (or do not have permission) to access each object. Capability-based systems are row-based mechanisms and do not easily lend themselves to this function. Deletion of an object from the system and revocation of access present yet another problem. The problem is that row-based systems do not provide an efficient means of determining which users have access to a given object.

7.2 PROFILES

Profiles [12], which have been implemented in some form on several systems, use a list of protected objects associated with each user. Since object names are not consistent or amenable to grouping, their size and number are difficult to reduce. If a user has access to many protected objects, the profile can get very large and difficult to manage. Also, all protected object names must be unique so full pathnames must be used. Creating, deleting and changing access to protected objects requires many operations since multiple users' profiles must be updated. Timely revocation of access to an object is very difficult unless the user's profile is automatically checked each time the object is accessed. Deleting an object may require some method of determining every user who has the object in his profile. In general, with profiles as with capabilities, answering the question of who has access to a protected object is very difficult. Since this is usually an important question in a secure system and more efficient mechanisms exist, profiles are not a recommended implementation of DAC.

7.3 ACCESS CONTROL LISTS (ACLs)

ACLs allow any particular user to be allowed or disallowed access to a particular protected object. They implement the access control matrix by representing the columns as lists of users attached to the protected objects. The lists do not have to be excessively long if groups and wild cards (see below) are used. The use of groups raises the possibility of conflicts between group and individual user. As an example, the ACL entries "PAYROL rw" and "Jones.PAYROL r" appear to conflict, but can be resolved in the design of the DAC mechanism. The Apollo system has a multiple, hierarchical group mechanism. The ACL entry has the form ``user-id.group.organization .node." As in Multics, if the ACL specifies access rights for the user by user-id then group access rights are ignored. This allows a particular user to be excluded or restricted in access rights [13]. In the Apollo, if a user is not on the ACL by user-id, but is a member of a group, those rights are used and organization and node memberships are not examined. Multiple group mechanisms add more complexity and may facilitate administrative control of a system, but do not affect the utility of a DAC mechanism.

Access to ACLs should be protected just as other objects are protected. The creation of groups must be controlled, since becoming a member of a group can change the objects accessible to any member. In many systems, e.g., Multics [14], a user must be a member of at least one group. One detriment of the group mechanism is that changing the members of a group results in changes to an unknown set of ACLs for protected objects. Allocation of groups could be a Systems Administrator function only, or it could be distributed to a Project Administrator type function. Problems could result from allowing any user to create a group and then be "owner'' of that group. If users were prohibited from listing the members of groups they are not in because of covert channels and privacy, it would be difficult to determine if a group was the correct one to use. System or Project Administrator control is a preferred mechanism.

7.3.1 Wild Cards

A wild card mechanism allows a string replacement where the wild card is specified. For example, in the Multics system ```PAYROL rw'' gives read and write access to any user in the PAYROL group. ``Smith.* r'' gives Smith read access, no matter what group the user Smith belongs to. ``*.*'' gives any user access. The group and wild card mechanisms allow the ACL list to be kept to a reasonable size. The use of wild cards raises the possibility of conflicts if a user has multiple ACL entries for an object. In the above example, Smith has a possible conflict; as a member of any group he can read and as a member of the PAYROL group he can read and write. The system must make a decision as to which one of the ACL entries it will apply when granting Smith access to the object. Various systems have different rules for resolving conflicts. One approach might be to have the system enforce an ordering of the ACLs. Another approach might be to allow ordering of the ACLs by the users.

In any case, the users must understand the rules in order to create effective ACL entries. A wild card mechanism adds more complexity, but does not affect the utility of a DAC mechanism. An exclusion capability, which allows a group to be specified with certain members to be excluded, is also useful and is required at class B3 and above.

7.3.2 Default ACLs

There are many side issues in the implementation of access control lists. Default ACLs are usually necessary for the user friendliness of the DAC mechanism. At the very least, when an object is created by a user, the user should be placed on its ACL by default. Some of the other possible default mechanisms include a system-wide default, a user-associated default or if the file structure is a tree, a default associated with the directory.

A system-wide default could be used as the default in cases where no other default had been specified. A system-wide default might give access only to the creating user. A user-associated default might work well on a system with a flat file structure. When a user is first entered on the system, his default ACL would have to be specified.

For file structures that are trees, a default(s) associated with the directory could be most efficient. If the user organizes the directory structure to represent project work or areas of interest, then the ACLs for all objects in a sub-tree would be similar. One default ACL in the directory would be for children that are files. For children that are directories either a separate sub-directory default ACL should be specified or the default ACLs should have to be stated explicitly by the user. Otherwise, unless care is taken, those with access to the root sections of the storage hierarchy could by automatic default get access 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 user intended. In other words, to err on the conservative side is preferred. In all implementations some user(s) must have permission to change the ACLs after they have been set by default, and the ability to change the defaults is very useful. Defaults can be implemented in two ways: they can be copied to the ACL or they can be pointed to by the ACL. If they are copied, then changes to the default will not affect the ACL; otherwise, changes in the default may cause changes in many ACLs.

7.3.3 Named ACLs

Another possible user friendly feature is "named" ACLs. One implementation of this feature uses a named ACL as a template. If a user often sets ACLs to the same list of Users, the setting user may want to create a named ACL as a template which, when used, copies that list into the ACL. When the named ACL is changed, there is no effect on the ACLs already in existence. This use of named ACLs has no particular detriments and is of limited usefulness. The other implementation of named ACLs places a pointer in the real ACL to the named ACL. Now when the named ACL gets changed, all of the real ACLs that use it also get changed. This is very convenient for the user, but when a named ACL is changed the user has no way of determining all of the protected objects affected by the change. The named ACLs also have to be protected in the same way as the real ACLs. Most of the features of named ACLs can be replaced by some group and default mechanisms.

In summary, access control lists are the most desirable implementation of discretionary access control. ACLs conveniently lend themselves to specifying a list of named users who are allowed to access each object. Also, providing access to defined groups of users is easily done with ACL-based mechanisms. ACLs, if properly implemented, will satisfy the B3 DAC requirement. At class B1 and above, care must be taken so that the implementation does not conflict with mandatory controls.

7.4 PROTECTION BITS

Protection bits are an incomplete attempt to represent the access control matrix by column. Implementation of protection bits include systems such as UNIX [15], which use protection bits associated with objects instead of a list of users who may access an object. (UNIX is a registered trademark of AT&T.) In the UNIX case the protection bits indicate whether everyone, the object's group or only the owner has any of the access modes to the protected object. The user who created the object is the owner, and that can only be changed through superuser privileges. The owner is the only one (besides a superuser) who can change protection bits.

The problem with protection bits is that they are an incomplete implementation of the access control matrix model. The system cannot conveniently allow or disallow access to a protected object on any single user basis. It has been suggested that groups be set up so that any needed combination of users can be specified. But, for more than a few users, the combinatorics of such a solution are unrealistic. Also, groups are controlled by the system administrator, and such a scheme would require full-time attention.

7.5 PASSWORD DAC MECHANISMS

Password protection of objects attempts to represent the access control matrix by row. If each user possessed his own password to each object, then the password is a ticket to the object, similar to a capability system (except, of course, with no dynamic domains). In most implementations of password protection, only one password per object or one password per object per access mode exists. Passwords. on protected objects have been used in IBM's MVS [16] and with other mechanisms in CDC's NOS [17] to implement DAC.

Many problems are associated with using a password protected DAC system. The use of passwords prevents the TCB from controlling distribution of access permissions. The sharing of passwords takes place outside the system. For a user to remember a password for each protected object is virtually impossible, and if the passwords are stored in programs they are vulnerable. 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 all or none. In such implementations, revoking a user's access requires revoking access from all other users with similar access and then distributing a new password to those who are to retain access. This becomes almost Impossible when passwords are stored in programs. 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. Thus a new file in MVS is not protected until the password protection mechanism is invoked.

If passwords are used as in the CDC NOS system to supplement another DAC mechanism, they do have one positive aspect. If all objects are protected with different passwords, Trojan horses can be restricted to only the objects that are handed to them. The use of passwords for a complete DAC is strongly discouraged, because there is no way to determine who has access to an object, and because managing such a system properly is very difficult.

7.6 SUMMARY

Access control lists, if implemented properly, can satisfy the DAC requirement for any class. Protection bits lack the ability to conveniently control access to each object to the granularity of a single user. Row-based mechanisms, such as capabilities and profiles, may encounter design problems with revocation of access and group access to objects. Passwords should not be used for discretionary access controls, except as a supplement to another DAC mechanism, to reduce that mechanism's vulnerability to Trojan horses.

8. THE SET OF DAC ACCESS TYPES

In this section, control permissions and access modes for objects are discussed. In the access control matrix (see figure I, in the previous section), control permissions and access modes are the entries in the matrix that specify what kind of access the subject has to the object.

Control permissions define which subjects have the ability to grant and revoke access permissions and change access modes. Sometimes control permissions include the ability to pass these control permissions to other subjects.

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.

8.1 CONTROL PERMISSIONS

In many current systems, the concept of control permissions is not separated from access modes. In Multics, ``modify access'' on the directory is actually the ability to change the ACLs of the children of the directory. It is recommended that control permissions and access modes be kept conceptually separate on any DAC system. This separation allows control of the object to be separated from access to the object.

Two basic control permissions exist: control and control with passing ability. The difference between these permissions is the. ability of the holder to propagate control to other subjects.

- control - Subjects who have control permission to objects are able to pass access permission and to set access modes to those objects for other subjects. Control permission does not allow subjects who possess it to give control to other subjects.

- control with passing ability - This control permission is identical to control with the exception that the holder can pass his control permission to other subjects. In a system implementing this permission, objects may have more than one subject who are able to control them.

Since control permissions allow a subject to specify access to an object, they can be used to implement the control model for a DAC system.

8.1.1 Control Models

The Criteria requires DAC controls on the ability to assign access permission to an object by a user already possessing access permission. The ability to assign these permissions should be controlled with the same precision as the ability to access the objects themselves. For example, at class B3 it should be possible to permit or prohibit both users and groups the authorization to assign access permission. Four basic models for DAC control exist: hierarchical, concept of ownership, laissez-faire, and centralized.

8.1.1.1 Hierarchical

Permissions to specify control of objects can be organized so that control is hierarchical, similar to the way most businesses operate. A simple example would have the system administrator at the root of the hierarchical tree. The system administrator would eve control permission to all objects on the system. This control permission would include the ability to pass those control permissions to other subjects. The system administrator would divide everyone else into subsets (e.g., departments), with each subset being represented by a directory in the storage hierarchy. The default access for each department would give the department head control permission for that directory. The department heads would divide their subordinates into subsets (e.g., projects) and set the defaults so that for each project, they give the project heads control permission on their project directories. The users at the bottom of the hierarchy would have no control permissions on any object. Notice that those in the hierarchy who have control permission on an object would be able to give themselves any access mode on the object. A hierarchical or tree-structure file system is not required in order to implement a hierarchical control model.

The advantage of a hierarchical structure is that control can be placed in the most trusted hands and the control can mimic the organizational environment. This should not imply, however, that control should be distributed based on one's position in the hierarchy. It is sometimes appropriate that a manager have no authorization for a subordinate's objects. The previously defined hierarchical structure provides the ability for multiple users to have the ability to control an object. Other hierarchical organizations can be imagined and implemented using control permissions. Hierarchical organizations could also be programmed into the DAC system without using control permissions, but such a restrictive implementation would result in a specific, fixed hierarchical structure. Such inflexible hierarchical control is not recommended. The hierarchical control model would satisfy the DAC requirement for all classes.

8.1.1.2 Concept of Ownership

Another control model associates with each object an owner (usually the creator of the object) who is the only user that has control permission to this object. The owner is always in ``full'' control of the objects that he has created and has no ability to pass that control to any other subject. Hence the owner may change the access permissions at any time in order to grant and to deny the access rights of the objects under his control.

This near absolute control by owner can be implemented administratively. The system administrator could set up the system so that the default access would always contain control permission, but the ability to pass control permission to other users 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. Therefore, 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 and not implementing any control permissions. This will enforce the owner concept because the creator of an object is known to the DAC mechanism and can change the access, but has no method of passing control permission to another user. UNIX is an example of an operating system where only the owner of an object has the ability to change its access (protection bits).

A disadvantage of owner control is the difficulty it creates with regard to non-owner users changing their access modes for an object (i.e., it is harder to share the objects). In order to gain or lose any access to an object, users must ask the owner of the object to grant or revoke access for them. However, in some operating environments, this disadvantage is actually a desired system characteristic.

Owner control of objects eliminates confusion concerning who controls access to each object. This particular control model will meet the DAC requirement for all classes.

8.1.1.3 Laissez-Faire

In this scheme, any user who has control permission with passing ability to an object may pass that permission on to any other user. No concept of ownership exists. The ACL will show all the users who can change the ACL of an object. Once a user has passed on control permission with passing ability to other users, they may pass this permission to other users without the consent of the creator of the object. Once the access permissions are given away, keeping control of an object is difficult. So long as the passing of control permission is separate from the granting of access permission, the laissez-faire control model will meet the DAC requirement for all classes. However, if the model were to require the passing of control permission in order to grant an access permission to a user, then the model would not meet the C2 DAC requirement, because it could not control the propagation of access rights.

8.1.1.4 Centralized

This control model gives one user, typically the system administrator, control permission to all objects on the system. That user does not have the ability to pass control on to other users. Thus- all control permission is held by one user. To obtain access to any object, a user must request that the controlling user grant him access permission. This places a considerable burden on the controlling user. Additionally, if requests for access permission are numerous, delays would probably result from the inflexibility of this model. Although it has limited flexibility, this control model would satisfy the DAC requirements for all classes.

8.1.2 File Structures and Control Permissions

Use of control 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. (Directories are sometimes called nodes, and files are sometimes called segments.) Control permissions can be applied to both directories and files. Control permission on a file changes control only to that particular file, but control permission on a directory may be implemented as the ability to control not only the directory but also all of the subdirectories and files under it (let us call this extended directory control). This extended directory control implements a very hierarchical control structure, since control permission on a directory means that the subject can change any access on the whole subtree below that directory.

A pure owner control policy could be implemented without using the extended directory control. The subject who creates a file or a directory would receive the ability to control either. The hierarchical control scheme described above would use the extended directory control and the control permission with passing ability. Giving the system administrator, the department heads and the project leaders extended directory control to the appropriate directory in the storage structure would give them control over the appropriate portion of the file system. Laissez-faire could use the extended directory control, along with control permission with passing ability on both the directories and the files.

Depending on the situation, one of the above schemes or a combination of the above schemes could be used to implement the control model of the DAC mechanism for any evaluation class.

8.2 ACCESS MODES

A fairly wide range of access modes are available in various DAC mechanisms.

Some implementations may combine some access modes, while other implementations do not provide many access modes. How the access modes are combined should be given careful consideration, because combining access modes can limit the flexibility of the DAC mechanism. The names of any of the access modes may vary from system to system. Many types of write access modes have been implemented in different systems for controlling what changes can be done to an object. This section discusses various read, write, execute, and delete access modes.

-READ - allows an object to be read but not changed in any way. On most of the systems, the READ mode also allows the object to be copied. Thus READ gives all access to the object except WRITE, EXECUTE, and DELETE.

-WRITE-APPEND - allows a user to expand an object but does not allow a user to change the previous contents of or view an object. WRITE-APPEND is also known as WRITE-EXPAND.

-WRITE-CHANGE - allows a user to modify and to delete all or part of the contents of an object but does not allow a user to expand or view the object.

-WRITE-UPDATE - allows a user to modify the contents of an object but does not allow a user to add to, delete from, or view an object.

-WRITE - allows a user to modify, add, or delete the contents of an object in any manner but does not allow a user to view an object.

In most operating systems little is known about the semantics of the data stored in files, although some objects such as directories and mailboxes may have more operating system manipulated structure associated with them. In general write accesses such as WRITE- CHANGE and WRITE-UPDATE are very difficult for an operating system to implement reliably. WRITE-APPEND, which forces new data to be added only at the beginning or the end of an object, is in the realm of most operating system architecture. If an operating system could support it, WRITE-APPEND would be useful for logging and transaction type operations.

-EXECUTE - allows a subject to run the object as an executable file. On some systems EXECUTE access requires READ access. An EXECUTE without READ would be recommended if a correct implementation is possible.

-DELETE - access mode allows an object to be deleted.

-NULL - access mode grants no access permissions and is used to allow exclusion of a particular user 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.

-CONTROL - (See Section 8.1)
-CONTROL WITH PASSING ABILITY - (See Section 8.1)

8.2.1 File Structures and Access Modes

If the files are organized in a tree structure then the directories are used to represent the non-leaf nodes of the tree. Three methods are available for controlling the access to the directories and their associated files:

1. access controls on the files but not on the directories

2. access controls on the directories but not on the files

3. access controls on both the files and the directories.

If access mode controls are placed only on the directories then once a user is given any access to a directory, he has the same access to all files under this directory. Of course if one of the objects under this directory is another directory (which we call a subdirectory) then users need access mode permission to the subdirectory before they can access the objects under the subdirectory. Placing access controls only on directories requires users to group files according to access types. This requirement could be too constrictive and could conflict with other reasons for grouping files. If access mode controls are placed only on files then controls are of a finer granularity. Access to individual files is controlled independently of other files under the same directory. But if no access controls are placed on directories, users could browse through the storage structure looking at the names of other users' files. Also, there is no control on where in the tree structure a subject could place a file, and this is usually the main purpose of a tree structure. By having the access mode controls on both directories and files, a subject needs the correct access mode permissions to both the directory and the file before it is allowed to access the file. Access modes for files are the basic software and hardware implemented access modes, such as those listed in Section 8.2, since the operating system has no knowledge of the semantics of the files. The rest of the objects (directories, mailboxes, etc.) are data structures maintained by the operating system, and their access modes are generally more elaborate extensions of the basic access modes and are usually implemented by protected subsystems. In a system like UNIX, the lack of execute access implies no access to the existing entire sub-tree under that directory (ignoring an implementation of links as in UNIX and our earlier discussion of access permissions). A user cannot give another user access to a file without also giving the correct read access to the parent directories. In Multics, the philosophy chosen was to enable a user to grant access to a file just by setting the ACLs correctly on that file. Null access on a directory will prevent a user from seeing the names or ACLs of the children of that directory. But if the user has an access mode to a child, he can access the child (he must already know its name). This 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.

Write-append allows a user to add new objects to the directory (i.e., to create files and sub- directories under the directory). Null is implied if no access mode is specified.

In order to have more detailed access control, an implementor may add other access modes to the minimal set. But the final implementation of the access modes should not be so complicated that users cannot easily remember the implication of each mode. If users cannot distinguish the functions of each access mode, they will just grant to other subjects either the full access rights or no access rights to an object. Also, implementations of any access type must be consistent and guaranteed to always function correctly. In implementing any access type, the designer must consider how much complexity it adds to the trusted computing base, relative to the gain in functionality.

9. MEETING THE CRITERIA REQUIREMENTS

This section of the guide explains the DAC requirements in the Criteria and presents implementation examples that will satisfy each requirement. DAC requirements may be viewed as three levels. The first level is at the C1 Criteria class. The next level begins at class C2 and does not change through class B2. The highest level requirement begins at class B3 and remains the same at class Al. At each of these levels, this guide quotes the requirement from the Criteria, explains what the requirement means, describes the minimum functionality that is necessary to meet the requirement at that level (and the functionality that is not required at that level), and presents an implementation example which satisfies the requirement.

The reader should keep in mind that the implementations presented are not the only ones that will meet their respective requirement. The combinations possible with the mechanisms mentioned are too numerous to list here. Each implementation must be evaluated on an individual basis. The implementation selected should be as simple and user friendly as possible, while meeting the control objective. Since the mechanism is user-invoked, if it is overly complicated, users may avoid using it to its fullest extent.

9.1 THE C1 DAC REQUIREMENT

Following is the DAC requirement for Criteria class C1:

> "The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals or defined groups or both."

Note that the class C1 requirement names examples of DAC mechanisms. These examples are not the only mechanisms that will satisfy the requirement. Many mechanisms are adequate to satisfy the C1 DAC requirement.

At class C1 , users do not need special privileges to be authorized to control access to objects. Unprivileged users may control access to objects they can access. A user with access to an object can in turn grant access to that object to other users.

The class C1 DAC mechanism need only provide support to allow access/no access; separate modes for read and write need not be supported.

Objects may be shared by defined groups. That is, a group of users that may have access to an object must be defined to the TCB. Groups can be defined by naming the users to be included in each group.

9.1.1 Minimum Functionality

- The DAC mechanism should control access to individual named objects by named users.

- If the mechanism is not capable of the above function, it must at least control access to each individual named object by defined user groups.

- The DAC mechanism must recognize and control access to individual objects.

9.1.2 Functionality Not Required

- Class C1 does not require that users be uniquely identified to the TCB. The users could be members of a group and thereby lose uniqueness. Therefore, it is not necessary for the C1 DAC mechanism to control access based on individual user identities.

- Although it is not required, user groups should be predefined by authorized users; that is, users who define groups should have some sort of special privilege to do so.

  In definitions for user groups, defining the members of the group by unique user identifier is not necessary. For example, defining a group consisting of anyone whose name starts with "M'' by using a wildcard such as ``M*'' would be permissible.

- Newly created named objects do not need to be automatically protected. Default public access is permissible at C1

- The C1 DAC requirement does not mandate controls to limit the propagation of access rights. In other words, users possessing a certain access right may or may not be permitted to pass that right on to other users at their own discretion.

- It is not necessary to define different access modes, such as read and write. The mechanism need only provide support to allow access/no access to objects.

9.1.3 An Acceptable Implementation

This section describes a DAC mechanism which would satisfy the C1 DAC requirement. This example is included for purposes of illustrating the functionality necessary to meet this requirement.

This mechanism, called SEE-ONE, utilizes profiles, which contain a list of accessible objects for each subject. When a new object is created on this system, it is unprotected until the system administrator identifies it to the DAC mechanism. From that point on, a subject must have that object's name in his profile before he can access it. The system administrator must place the name of the object on the profiles of the subjects who should be allowed to access it. No access modes appear on the profiles. If an object is in a subject's profile, that subject has total access to that object (read, write, execute, delete). Subjects cannot pass access permissions on to other subjects. This particular mechanism does not support user groups.

While SEE-ONE will satisfy the C1 DAC requirement, it is not without drawbacks. Decentralizing control of protected objects from the system administrator to the user community may be desirable. Most systems have a considerable amount of object creation and deletion. Excessive object creation and deletion could overburden the system administrator. The implementation described here contains only the minimal functionality necessary to meet the C1 DAC requirement.

9.2 THE C2 THROUGH B2 DAC REQUIREMENT

Following is the DAC requirement for Criteria classes C2 through B2:

> "The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism. (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.''

At this level protection on newly created objects must not default to public access. Either the system must have a specific rule that limits access to each object to its creator or to a previously specified subset of users, or the system must require the creator of an object to specify who will be allowed to access it before allowing him to create that object. Limiting access to the creator of the object is the method that is preferred.

The mechanism must have the ability to include or exclude access to objects on a per user basis. If group access controls are provided, groups must be precisely defined by listing the unique names of the users in the group. Hence, groups are made of named users.

Users must possess special authorization to control access to objects. This special authorization is called control permission (see section 8.1). The authorization required may range from system administrator to users controlling access to objects that they have created. The latter is preferred because it does not require system administrator intervention. Either method is acceptable, but the former method could result in delays if the system administrator is not available.

9.2.1 Minimum Functionality

The minimum requirements specified here are in addition to the functionality required to satisfy the C1 DAC requirement.

- The mechanism must provide the capability of including or excluding access to each object on a per user (or per group if groups are implemented) basis.

- The mechanism must provide that all objects are protected. Protection by default must be provided for all newly created objects at creation time.

- Users must possess control permission to assign access permission to objects. This authorization usually takes the form of a special access type, sometimes called CONTROL or OWNER.

- If group access controls are provided, groups must be precisely defined by using the unique identifier of each member of the group.

9.2.2 Functionality Not Required

- Providing access to objects by groups of users is optional at this level.

- It is not necessary to define different access modes, such as read and write. The mechanism need only provide support to allow access/no access to objects.

9.2.3 An Acceptable Implementation

This section describes a DAC mechanism called SEE-TWO which would meet the C2-B2 DAC requirement. The purpose of this description is to illustrate the functionality that is necessary to meet the requirement.

This particular mechanism uses an ACL-like DAC mechanism. Each object has a list of users who are allowed to access the object, along with the modes of access the users have to the object. By including or excluding users' names on the list, access can be controlled on each object to the granularity of a single user.

When a creator requests the creation of an object, he must specify the desired access controls. Otherwise, the mechanism will enforce a restrictive default access control. The creator can grant other users the privilege of accessing the object as he wishes. Only the creator has the privilege of passing access on to other users. This concept is called ownership. Ownership is not transferrable to other users. If the creator wishes, he can specify a default protection list for all objects he creates. The default would contain a list of all users which will be added to the protection list of any new objects created by that subject.

This SEE-TWO implementation will satisfy the C2-B2 DAC requirement. Furthermore, it has an advantage over the SEE-ONE implementation; the management of an object is decentralized from the system administrator to the owner of that object. A disadvantage of the SEE-TWO implementation is that it does not provide the capability to grant access to groups of uniquely identified users. Using this mechanism could be cumbersome in some environments since all users must be explicitly listed. In other environments, however, requiring the users to be explicitly listed is desirable.

9.3 THE B3 THROUGH A1 DAC REQUIREMENT

Following is the DAC requirement for Criteria classes B3 and A1.

> "The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., access control lists) shall allow users to specify and control sharing of those objects, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object. Furthermore, for each such named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.''

At this level the requirement mandates that the DAC mechanism control access to each named object. Access must be controlled on a per user basis. In addition, access control based on groups of named users is required at this level. Authorized users who control access to objects must be able to list specific named users that are allowed to access each object they control. They also must have the capability to list groups of named users that will be allowed to access each object. These lists must also contain the respective access modes that each named user or group of named users has on each object.

Additionally, for each object, authorized users must be able to specify a list of named users and a list of groups of named users that are not to be allowed to access that object. Thus, for example, an object could exist on a system that is public read-only except for JOE. USER who is not allowed to access the object at all.

9.3.1 Minimum Functionality

In addition to the minimum requirements stated in this section for the C1 and C2-B2 DAC requirements, the following functionality must be supplied in order for a mechanism to satisfy the B3-A DAC requirement.

- For each named object, the mechanism must provide the ability for authorized users to specify a list of named users along with each user's respective modes of access to the object.

- The mechanism must provide support to allow different access modes, such as read and write. Merely providing for access/no access to objects is not sufficient.

- For each named object, the mechanism must provide the ability for authorized users to specify a list of groups of named users along with each group's respective modes of access to the object.

- For each named object, the mechanism must provide the ability for authorized users to specify a list of named users and groups of named users for which no access will be allowed to that object.

9.3.2 An Acceptable Implementation

Following is an example of a DAC mechanism, called BEE-THREE, that would be sufficient to satisfy the B3-A1 DAC requirement. This example is included for the purpose of illustrating the functionality necessary to meet the requirement.

When an object is created, an ACL that only permits access for the creator is associated with the object. All objects are protected. Each object's ACL describes the users or groups that can access it, along with their respective access modes to that object. Protected objects are segments and directories. The available access modes on segments are read, execute, and write. On directories the available access modes are search, modify, and append. A process must have modify access to the containing directory of an object in order to modify the ACL of the object.

Access groups are defined by the system projects, which are also used for accounting purposes. Only the system administrator can define projects by listing the unique identities of each user to be included in the projects. Users can be defined in more that one project, but can only be active in one project at a time. Changing projects requires that users logout of the system, and then login under the new project name.

If a user or group of users is listed on an object's ACL without an access mode, that individual or group has no access to that object. If a subject has more than one ACL entry on an object with conflicting access modes, the system takes the most restrictive access mode. This makes it possible to give everyone on a project access to an object, while denying certain members of that project any access at all. This can be done by including the project name on the ACL with the access mode desired while, at the same time, including those users to be excluded on the ACL with no access modes.

This BEE-THREE implementation will satisfy the B3-A1 DAC requirement. Furthermore, it has an advantage over the SEE-TWO implementation: it provides the capability to grant access to individual users or to groups of uniquely identified users. However, in this implementation it is impossible to allow a group some access to an object (such as read), but allow one group member some additional access (such as read, write) to that object.

10. OTHER TOPICS

10.1 PROTECTED SUBSYSTEMS

In order to provide users with access types finer than the basic read, write, execute, etc., some systems (actually a rather limited number) support protected subsystems. Saltzer and Schroeder [13] 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 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 more flexible 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 inflict on objects protected by the subsystem. Likewise, the lending user could also encapsulate the lent program in a protected subsystem of its own to protect it from the programs of the borrower. The danger of Trojan horse attacks and the inadequacy of most DAC mechanisms to protect against such attacks was discussed in Section 6. A protected subsystem which incorporates one of the DAC mechanisms discussed earlier is no less vulnerable to attacks on objects within that subsystem. Furthermore, readers should consider the effects of illusory security which may be inherent with a protected subsystem. Apollo's Domain [18] implements limited protected subsystems. Domain allows users to write their own subsystems, but it does not provide a mechanism for protected subsystems to interact with one another. A user has the 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 access control lists associated with the subsystem manager.

10.2 ADMINISTERING AND USING DAC

The Trusted Facility Manual (TFM) must explain to the system administrator how to set up the DAC mechanism and how to properly administer a DAC system. The manual must explain 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 manual 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's code. ACLs may need to be set on the supervisor entry points and the I/O daemons or device queues.

The TFM must be easy to understand and follow, since proper set-up and proper administration of the DAC mechanism is critical to its effectiveness. 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 and must indicate settings required for the product's evaluation class. The manual should explain Trojan horse attacks and should give an example of one that could defeat the DAC mechanism and be undiscovered by the audit mechanism.

The Security Features Users Guide (SFUG) must explain and give examples of how to use the DAC mechanism. The manual should be easy for the users to understand, since the mechanism is user-invoked. Like the manual for the system administrator, the users' manual should explain Trojan horse attacks and give an example of one that could defeat the DAC mechanism and be undiscovered by the audit mechanism.

10.3 AUDITING DAC

Auditing is an important part of a secure computer system. 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 auditable. Any change in any ACL, including the default ACLs, should be auditable. Any attempt to access a DAC-protected object should be auditable. Any changes to group definitions should be auditable. In general an audit log message should include a timestamp, the subject's identity, the object's identity, the type of action and any other pertinent information.

10.4 VERIFYING DAC

Verifying the correctness of the DAC mechanism is not well understood. As can be seen by this paper, DAC does not have well defined rules as does mandatory access control. In fact no formal model of the myriad of DAC implementations currently exists although some mention is made of the DAC mechanisms in the Bell and LaPadula [19] formal model for mandatory access control. In systems such as SCOMP [20] 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(s) for DAC and verification of systems where DAC is important should at least provide assurance by a structured argument that demonstrates the DAC implementation's sufficiency to implement its specification.

10.5 DAC ADD-ONS

For some operating systems (e.g.. IBM MVS), products which provide DAC mechanisms are available as `add-ons' (e.g., ACF2, Top Secret, RACF). 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 either 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 would receive a D rating against the Criteria if evaluated on their own. To demonstrate that an add-on product operates as claimed, the add- on, along with the underlying operating system, is evaluated as a total system. The security provided by the add-on relies on the correct operation of the underlying system.

11. SUMMARY OF DESIRED DAC FEATURES

Regardless of the Criteria class a DAC mechanism is intended to satisfy, the preferred DAC mechanism within the current state-of-the-art is the access control list. The deletion of subjects and objects is a potential problem for any DAC mechanism, including ACLs. Regardless of whether a subject or object is deleted, the mechanism should handle the deletion efficiently, making sure that any dangling references do not grant unintended access after the deletion. An asset of ACLs is their design, which represents the access control matrix by column. An object's ACLs are automatically purged when the object is deleted. Unlike row-based mechanisms, ACLs do not require searching the profiles of all subjects to determine who has access to the deleted object. However, when subjects are removed from the ADP system, ACLs pose the same problem; all system objects must be searched to purge that subject from all ACLs. Since objects are much more volatile than subjects, ACLs are more efficient than profiles.

At classes B3 through A1, access modes must be implemented, but they are highly recommended at all lower levels of the Criteria. The access modes read, execute, and write should be implemented at a minimum and write-append should be considered.

At class C2 and above, some sort of control permission must be implemented. The control models described in Section 8.1.1 are acceptable. The preferred model is the concept of ownership model. This model assures that some user is accountable for each object on the ADP system. Ownership should not be transferrable between users. If it is transferrable, ownership transfers must be audited.

Discretionary Access Control cannot deter hostile attempts to access sensitive information. DAC mechanisms can be designed to eliminate casual browsing through the storage system, to prevent accidental disclosure, modification, and destruction of data, and to provide for the sharing of data between users. A mandatory security policy must also be implemented to provide adequate protection if users are not allowed to share all data on the system. An effective audit trail mechanism is also necessary in order to detect hostile attacks on the DAC mechanism.

12. REFERENCES

1. Department of Defense Trusted Computer System Evaluation Criteria, DoD, DoD 5200.28-STD, 1985.

2. Schell, R. R., ``Security Kernels: A Methodical Design of System Security,'' in Technical Papers, USE Inc., Spring Conference, 5-9 March 1979, pp. 245-250.

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

4. Boebert, W.E., and Ferguson, C.T., ``A Partial Solution to the Discretionary Trojan Horse Problem,'' 9th Security Conference, DoD/NBS, September 1985, pp 141-144.

5. Downs, D, ``Discretionary Access Control Guideline,'' Aerospace Report, The Aerospace Corporation, September 1985.

6. Boebert, W.E., Kain, R.Y. and Young, W.D., ``Secure Computing: The Secure Ada Target Approach,'' Scientific Honeyweller, Vol. 6, No. 2 July 1985, pp 1-17.

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

8. Fabry, R.S., ``Capability-Based Addressing,'' Communications of the ACM, Vol. 17, No. 7, July 1974, pp. 403-411.

9. Redell, D.D., ``Naming and Protection in Extensible Operating Systems,'' AD-A001 721, published by M.I.T., Cambridge MA, November 1974.

10. Karger, P.A. and Herbert, A.J., ``Lattice Security and Traceability of Access,'' Symposium on Security and Privacy, IEEE, April 1984, pp. 13-23.

11. Gligor, V., Huskamp, J., Welke, S., Linn, C., and Mayfield, W., ``Traditional Capability-Based Systems: An Analysis of Their Ability to Meet the Trusted Computer Security Evaluation Criteria,'' IDA Paper Pl 935, October 1986.

12. Computer Associates, CA-SENTINEL Reference Guide, 1983.

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

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

15. UC Berkeley, UNIX Programmer's Manual, 7 ed., 1981.

16. IBM, Access Method Services, 1983.

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

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

19. 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.

20. Benzel Vickers, T., ``Overview of the SCOMP Architecture and Security Mechanisms,'' Tech. report MTR-9071, MITRE Corp., September 1983.